CODIFICADORES DE IMAGEM USANDO PLANOS DE BITS GENERALIZADOS

Lara Cristiana Rodrigues Lourenço Feio

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Aprovada por:

Prof. Eduardo Antônio Barros da Silva, Ph.D.

Prof. Gelson Vieira Mendonça, Ph.D.

Prof. Abraham Alcaim, Ph.D.

RIO DE JANEIRO, RJ - BRASIL
DEZEMBRO DE 2000

FEIO, LARA CRISTIANA RODRIGUES LOURENÇO
Codificadores de imagem usando planos de bits generalizados [Rio de Janeiro] 2000
xvii, 169 pp. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia Elétrica, 2000)
Tese - Universidade Federal do Rio de Janeiro, COPPE
1.Codificação de imagens 2.Codificação usando planos de bits 3.Quantização por aproximações sucessivas escalares 4.Quantização por aproximações sucessivas vectoriais
I.COPPE/UFRJ II.Título (série)

Ao meu marido Paulo.

Agradecimentos

Em primeiro lugar quero agradecer à COPPE pela oportunidade que me foi concedida para a realização deste Mestrado, disponibilizando todos os meios materiais e humanos para a sua concretização.

Quero também, de uma forma muito especial, agradecer a compreensão e colaboração do meu orientador, Professor Eduardo, que sempre mostrou total disponibilidade sempre que esta lhe foi solicitada.

Sem citar nomes para não correr o risco de algum esquecimento, quero também agradecer a todos os professores e colegas do Laboratório de Processamento de sinais, que ao longo do tempo de duração deste trabalho, contribuíram para a sua finalização com pequenas de aparência, mas na realidade grandes ajudas.

Quero agradecer aos meus pais, Zulmira e Oscar, e ao meu irmão Filipe que, embora longe, sempre me deram o máximo de apoio e são os principais responsáveis pela pessoa que sou hoje.

A todos os meus familiares que sempre acreditaram em mim e me incentivaram para a realização deste Mestrado quero agradecer muito sinceramente.

De uma forma especial, quero expressar a minha profunda gratidão ao meu marido Paulo que esteve sempre comigo, dando-me o maior carinho e apoio mesmo nas situações mais adversas.

A todos muito obrigado.

Lara

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

Codificadores de imagem usando planos de bits generalizados

Lara Cristiana Rodrigues Lourenço Feio

Dezembro/2000

Orientador: Eduardo Antônio Barros da Silva

Programa: Engenharia Elétrica

Os codificadores baseados em transformada *wavelet* e que usam planos de bits têm vindo a apresentar um excelente desempenho na compressão de imagens estáticas. Recentemente, este conceito de planos de bits foi generalizado com a introdução de planos de bits vectoriais. Nesta tese, são propostos novos algoritmos de codificação de imagens, que são generalizações de codificadores por planos de bits escalares já existentes. A generalização é baseada na substituição dos planos de bits escalares por planos de bits vectoriais. Também é feita uma análise comparativa do desempenho dos diversos codificadores usando tanto planos de bits escalares quanto vectoriais. Resultados experimentais mostram que o uso de planos de bits vectoriais em substituição aos escalares pode ser vantajoso sob o ponto de vista taxa $\times$ distorção.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

Image Coding Using Generalised Bitplanes

Lara Cristiana Rodrigues Lourenço Feio

Dezembro/2000

Advisor: Eduardo Antônio Barros da Silva

Department: Electrical Engineering

The encoders based on wavelet transform that use bitplanes present excellent results in the compression of static images. Recentely, this concept was generalised with the introdution of vector bitplanes. In this thesis, new algorithms for image coding are proposed, which are generalisations of the already existent scalar bitplane encoders. The generalisation is based on the substitution of the scalar bitplanes by vector bitplanes. It is also made a comparative analysis of the performance of the several encoders using scalar and vector bitplanes. Experimental results show that the use of vector bitplanes in substitution to scalar bitplanes can be advantageous in terms of rate $\times$ distortion performance.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Motivação

Na última década, cada vez mais as imagens digitais estão presentes em diversas aplicações do nosso dia-a-dia. Com isto, cresceu muito a necessidade de efectuar um processamento eficiente dessas imagens, de forma a que se adaptem ao tipo de aplicação. Dependendo da aplicação das imagens, pode-se ter que ser mais exigente ao nível da qualidade visual, da velocidade de transmissão ou mesmo ao nível da capacidade dos dispositivos de armazenamento. São estes diferentes níveis de exigências que têm vindo a motivar o desenvolvimento de novas técnicas que permitam diminuir o espaço necessário para guardar uma imagem e consequentemente aumentar a velocidade da sua transmissão, mantendo sempre o nível de qualidade visual exigido para a imagem. Isto é, dependendo do nível de qualidade visual que se pretende para a imagem digital esta deverá ser mais ou menos comprimida. A este processo chama-se *codificação da imagem*.

A codificação da imagem é essencialmente baseada no facto das imagens naturais conterem muitas redundâncias, o que significa que se esta característica for explorada de uma forma eficiente é possível representar a imagem usando muito menos informação.

Este processo de codificação é composto principalmente por três fases:

1. **Transformação**: Nesta fase o que se pretende é obter uma outra repre-

sentação dos pixeis da imagem, que permita que os novos valores gerados sejam mais facilmente codificáveis.

2. **Quantização**: Os valores obtidos na transformação são discretizados de forma a obter um conjunto finito de símbolos. É durante esta fase da compressão que são introduzidas perdas na imagem.

3. **Codificação**: A cada símbolo gerado na fase anterior é atribuído um determinado número de bits que pode ser variável.

Para a recuperação da imagem será percorrido o caminho inverso, ou seja, descodificação da imagem e em seguida a transformação inversa. A diferença entre a imagem recuperada e a original será tanto maior quanto maior for o nível de quantização introduzido.

Entre os métodos de compressão desenvolvidos actualmente os que mais se destacam são aqueles que, na primeira fase do processo de codificação, fazem a decomposição da imagem em sub-bandas, através da aplicação de uma *transformada wavelet* e na segunda fase fazem uma quantização por planos de bits escalares. Neste tipo de quantização, os coeficientes da imagem transformada são codificados através de iterações sucessivas, isto é, em cada iteração é codificado um plano de bits dos coeficientes da imagem.

Este tipo de codificadores, chamados de *codificadores por planos de bits*, tem apresentado um excelente desempenho na codificação de imagens estáticas, sendo, por isso, parte integrante de alguns padrões, entre eles o JPEG-2000 [13] e o procedimento de codificação de textura do MPEG4 [9].

Embora estes codificadores estejam geralmente associados a uma técnica de quantização escalar, onde cada coeficiente é individualmente decomposto em planos de bits, em [7] este conceito foi generalizado para o uso de vectores. Neste caso, cada vector, composto por vários coeficientes, é decomposto em planos de bits vectoriais.

A substituição da quantização escalar por uma quantização vectorial já foi realizada tendo apresentado resultados bastante animadores.

1.2 Objectivo do trabalho

O que motivou o desenvolvimento deste trabalho foi o interesse de obter uma resposta para a seguinte questão:

Será que a substituição dos planos de bits escalares por planos de bits vectoriais, em alguns dos algoritmos já existentes, pode levar a uma melhoria nos seus desempenhos?

Que factores influenciam o desempenho dos algoritmos?

O objectivo deste trabalho é, então, analisar o desempenho de um conjunto de algoritmos usando os dois tipos de quantização e entender o comportamento de cada um deles.

1.3 Organização do trabalho

Este trabalho encontra-se dividido em 6 capítulos principais.

No Capítulo 1 é feito um breve enquadramento do trabalho, sendo apresentadas as questões que levaram ao desenvolvimento do mesmo.

No Capítulo 2 é apresentado o conceito de transformada *wavelet* assim como a grande contribuição deste tipo de transformada na codificação de imagens.

O Capítulo 3 descreve o processo de quantização por aproximações sucessivas tanto de grandezas escalares como de grandezas vectoriais.

No Capítulo 4 são descritos os algoritmos que foram implementados para fazer a comparação do desempenho usando planos de bits escalares e planos de bits vectoriais. Os algoritmos comparados são o EZW, o EZW/C, o SPIHT, o MGE e o SWEET que usam planos de bits escalares e as suas extensões para planos de bits vectorias, denominados SAWVQ, SAWVQ/C, SPIHTVQ, MGEVQ e SWEETVQ.

No Capítulo 5 são apresentados os resultados através da análise dos bits gastos em cada uma das fases de codificação e da PSNR [1] obtida para cada um dos

[1]Definida como: $PSNR = 10 \log \frac{255^2}{ems}$, onde $ems = \sum_{i=1}^{M} \sum_{j=1}^{N} (x_{i,j} - \bar{x}_{i,j})^2$ em que $x_{i,j}$ é o valor do coeficiente da imagem original e $\bar{x}_{i,j}$ o valor do coeficiente da imagem reconstruída. Alguns

algoritmos propostos.

O Capítulo 6 apresenta as conclusões e faz algumas sugestões para futuros desenvolvimentos.

autores a denominam, em português, *razão pico-ruído* (RPR)

Capítulo 2

Codificação de imagens utilizando transformada *wavelet*

2.1 Introdução

Hoje em dia, os métodos mais utilizados na codificação de imagens são os que fazem codificação por transformada, cujo objectivo principal é a redução da correlação existente entre os diversos elementos de uma determinada imagem, de forma a concentrar a maior quantidade de energia possível no menor número de coeficientes, sendo depois estes quantizados separadamente. Para isto, os pixeis da imagem são projectados em uma base mais favorável, a qual condensa a energia em menos componentes e torna a imagem menos sensível a erros. Assim, o principal problema da codificação por transformada está na escolha da base que melhor cumpre o objectivo. Actualmente, os tipos de transformada que estão a ser mais utilizados na compressão de imagem são: Transformada Discreta de Cosseno (DCT) e Transformada *Wavelet* (DWT), sendo esta última descrita neste capítulo.

Tanto as transformadas baseadas na análise de Fourier, como é o caso da DCT, como aquelas que se baseiam na análise por *wavelets*, como é o caso da DWT, assentam na decomposição de um sinal em uma base de funções. Contudo, enquanto que a análise de Fourier parte do princípio de que qualquer função pode ser decomposta numa soma de funções sinusoidais com diferentes frequências, a

análise por *wavelet* utiliza uma função protótipo, designada por *wavelet mãe*, a partir da qual se obtêm as restantes *wavelets*, versões escaladas e deslocadas da função protótipo. Desta forma, com este último tipo de transformadas é possível explorar a redundância de toda a imagem, assim como aquela existente entre as diferentes bandas de frequência.

Neste capítulo são apresentados os conceitos básicos da transformada *wavelet* assim como as vantagens da utilização deste tipo de transformada na codificação de imagens.

2.2 Transformada *wavelet*

2.2.1 Definição

Nesta secção é feita uma breve apresentação de transformadas *wavelet* contínuas e discretas, bem como como a sua definição matemática.

2.2.1.1 Transformada *wavelet* contínua

A transformada *wavelet* de um sinal contínuo no tempo $f(t)$ pode ser decomposta em um conjunto de *wavelets*, onde cada uma destas é uma versão escalonada e deslocada da função mãe $\psi(t)$. A transformada *wavelet* contínua pode então ser definida como:

$$F_W(u,s) = \int_{-\infty}^{\infty} f(t)\frac{1}{\sqrt{s}}\psi^*(\frac{t-u}{s})dt \tag{2.1}$$

onde:

$$\psi_{s,u}(t) = \frac{1}{\sqrt{s}}\psi(\frac{t-u}{s}) \tag{2.2}$$

representa as funções base $\psi_{s,u}(t)$.

Os parâmetros s, u representam expansões e translações da função mãe, respectivamente, e '$*$' representa o complexo conjugado.

O sinal $f(t)$ pode ser recuperado através da sua transformada *wavelet* contínua $f_W(u,s)$ usando a seguinte equação:

$$f(t) = \frac{1}{C_\psi} \int_0^\infty \int_{-\infty}^\infty F_W(u,s)\psi_{u,s}(t)du\frac{ds}{s^2} \tag{2.3}$$

desde que a constante C_ψ seja tal que:

$$C_\psi = \int_0^\infty \frac{|\psi(w)|}{w}dw < \infty \tag{2.4}$$

2.2.1.2 Transformada *wavelet* discreta

Da mesma forma que para o tipo de transformada anterior, a transformada *wavelet* discreta pode decompor o sinal $f(t)$ em versões escalonadas e deslocadas da função mãe $\bar{\psi}(t)$, e é definida pela seguinte equação:

$$F_W(m,n) = \int_{-\infty}^\infty 2^{\frac{-m}{2}}\psi(2^{-m}t - n)f(t)dt \tag{2.5}$$

Da mesma forma, o sinal $f(t)$ pode ser recuperado através da sua transformada *wavelet* discreta $F_W(m,n)$ usando a seguinte equação:

$$f(t) = \sum_m \sum_n F_W(m,n)2^{\frac{-m}{2}}\bar{\psi}(2^{-m}t - n) \tag{2.6}$$

As funções representadas por $\psi(t)$ e $\bar{\psi}(t)$ representam as *wavelets* de análise e de síntese, respectivamente. Caso $\psi(t) = \bar{\psi}(t)$, a transformada *wavelet* discreta é ortogonal, se $\psi(t) \neq \bar{\psi}(t)$ então a transformada *wavelet* discreta é biortogonal.

2.2.2 Implementação usando Bancos de Filtros

O grande problema da transformada *wavelet* é a sua difícil implementação em termos computacionais, uma vez que o principal objectivo é projectar o sinal em infinitos espaços de detalhes. Contudo, em [16] foi mostrado que é possível calcular a transformada *wavelet* de um sinal usando um banco de filtros.

A figura 2.1 esquematiza o banco de filtros de análise que permite obter a transformada *wavelet discreta* do sinal $f(n)$. $H_0(z)$ representa a transformada Z do

filtro passa-baixo de análise e $H_1(z)$ representa a transformada Z do filtro passa-alto de análise.

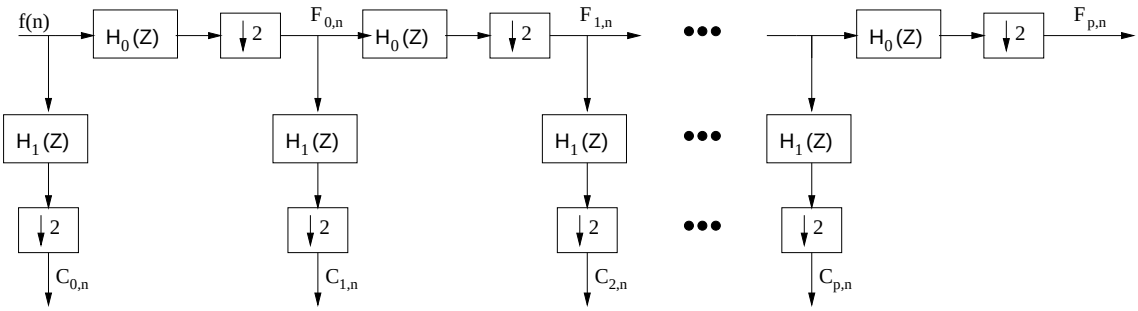


Figura 2.1: Codificação de um sinal $f(n)$ usando uma transformada *wavelet*

Pela figura pode-se ver que o sinal $f(n)$ é filtrado tanto pelo filtro passa-baixo como pelo passa-alto e o resultado de cada uma destas filtragens é subamostrado por um factor de 2. Assim, dependendo do número de filtros de análise que são colocados em série é possível obter uma transformada *wavelet* com o número de decomposição em sub-bandas que se pretende. Para isso, os sinais gerados em cada um dos filtros passa-baixo são filtrados o número de vezes desejado. Após p estágios de filtragem, têm-se p conjuntos de coeficientes $C_{0,n}$ até $C_{p,n}$ que representam os detalhes do sinal de entrada $f(n)$ nas diferentes resoluções, e um conjunto de coeficientes $F_{p,n}$ que representa o conteúdo do sinal $f(n)$ nas frequências mais baixas. Os índices $0, 1, ..., p$ representam o nível de detalhe de resolução do mais alto até ao mais baixo.

Para que o sinal $f(n)$ seja novamente recuperado, os coeficientes gerados no filtro de análise são novamente filtrados agora por um banco de filtros de síntese, como o que se encontra esquematizado na figura 2.2. Aqui, $G_0(z)$ representa a transformada Z do filtro passa-baixo de síntese e $G_1(z)$ representa a transformada Z do filtro passa-alto de síntese.

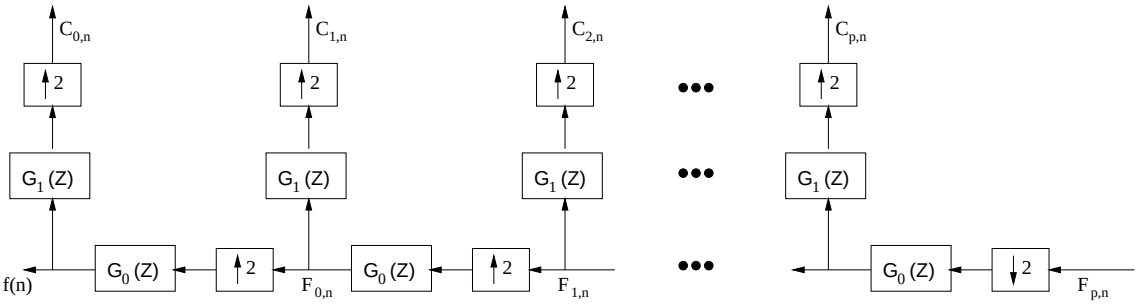


Figura 2.2: Descodificação de um sinal $f(n)$ usando uma transformada *wavelet*

Pode-se observar que o sinal é reconstruído interpolando os coeficientes $C_{0,n}$ até $C_{p,n}$ por um factor 2 e filtrando-os usando um filtro cuja resposta em frequência é $G_1(z)$, sendo em seguida somado o resultado dos coeficientes de detalhe igualmente interpolados por um factor de 2 e filtrados usando um filtro cuja resposta em frequência é G_0.

Contudo, para que a recuperação do sinal seja per

$$G_0(z) = z^{2m-1} H_1(-z) \tag{2.7}$$

$$G_1(z) = -z^{2m-1} H_0(-z) \tag{2.8}$$

Desta forma, com esta série de equações é possível implementar computacionalmente a transformada *wavelet* discreta.

2.2.3 Transformadas *wavelet* bidimensionais

A aplicação da transformada *wavelet* a imagens é feita em dois passos: aplica-se a transformada na direcção horizontal da imagem e em seguida na direcção vertical, como se se tratasse de um sinal de apenas uma dimensão. Cada estágio do banco de filtros é aplicado duas vezes. Na primeira vez, as linhas são filtradas pelo filtro passa-baixo e pelo passa-alto e o resultado subamostrado por um factor 2. Na segunda vez, as colunas do resultado são filtradas pelo filtro passa-baixo e pelo passa-alto e o resultado subamostrado por um factor 2. O resultado é uma imagem composta por 4 imagens com um quarto do tamanho da imagem original, cada uma correspondente às seguintes filtragens:

- filtro passa-baixo nas direcções horizontal e vertical;
- filtro passa-baixo na direcção horizontal e passa-alto na direcção vertical;
- filtro passa-alto na direcção horizonta e passa-baixo na direcção vertical;
- filtro passa-alto nas direcções horizontal e vertical.

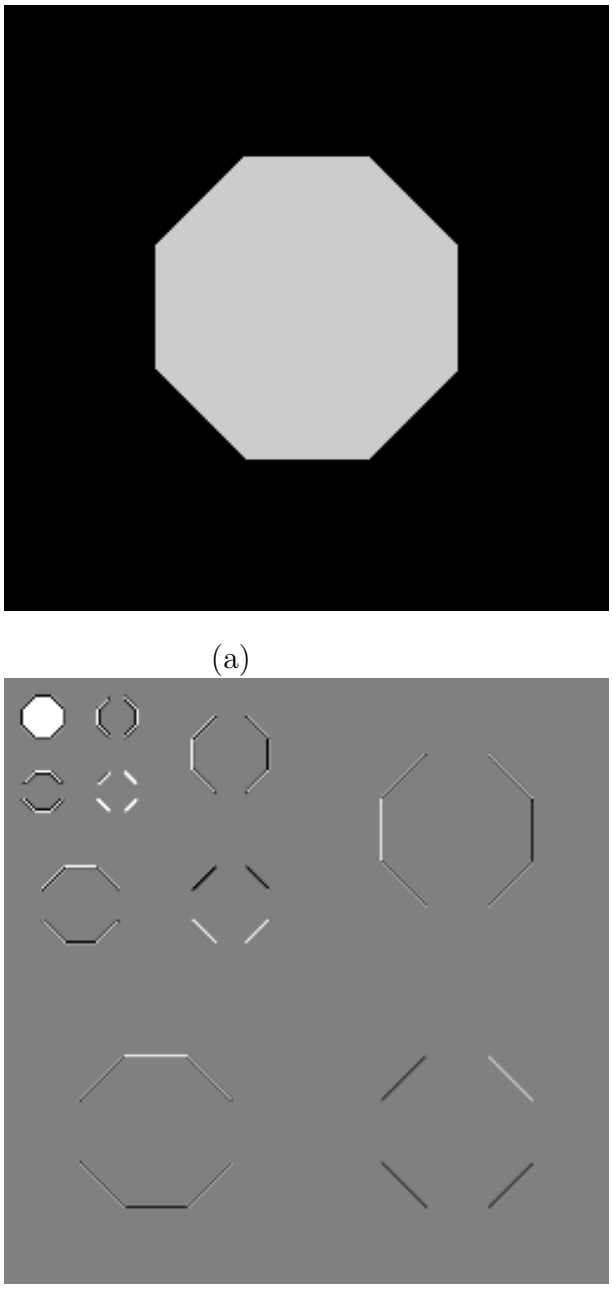
Em seguida, o processo é repetido recursivamente para a banda passa-baixo.

Nas figuras 2.3 e 2.4 são apresentadas as imagens LENA 256×256 e OCTÓGONO, respectivamente, e as transformadas *wavelet* de 3 estágios correspondentes. Aqui é possível observar a diferença das filtragens em cada uma das bandas de diferente orientação.

(a)

(b)

Figura 2.3: (a) Imagem LENA 256×256 original; (b) Transformada *wavelet* de 3 estágios da imagem LENA 256×256.



(a)

(b)

Figura 2.4: (a) Imagem OCTÓGONO original; (b) Transformada *wavelet* de 3 estágios da imagem OCTÓGONO.

2.3 Codificação de imagens usando transformada *wavelet*

Os mais recentes algoritmos de codificação de imagens estáticas fazem um pré-processamento aplicando uma transformada *wavelet* antes de iniciar a fase de codificação propriamente dita.

A transformada *wavelet*, como qualquer outra transformada, vai apenas gerar coeficientes que se encontram menos correlacionados do que os pixeis que compõem a imagem original.

Este tipo de codificadores tem apresentado muito bons resultados quando comparados com os métodos de transformada utilizados anteriormente, como é o caso da DCT que é uma transformada por blocos.

Nas figuras 2.3 e 2.4 são apresentadas duas imagens com as suas respectivas transformadas *wavelet*. Através da observação destas imagens é possível tirar algumas conclusões acerca dos factores que levam ao sucesso da utilização deste tipo de transformada. Assim:

1. **Grande quantidade de zeros**

   Quando é aplicada aos coeficientes da imagem uma transformada *wavelet*, com determinadas características, uma grande parte dos novos coeficientes que são gerados, chamados de coeficientes da transformada, têm o valor zero ou então muito próximo de zero. Isto significa que, esta transformada concentra, de uma forma muito eficiente, a maior parte da energia em muito poucos coeficientes, levando todos os restantes a terem valores com muito pouco significado. Esta característica pode ser facilmente visualizada através das figuras 2.3b e 2.4b, onde se verifica que a imagem está praticamente toda na banda de mais baixa frequência, ficando as restantes bandas apenas com os detalhes.

   O facto de se obter esta grande quantidade de coeficientes nulos ou quase nulos é muito vantajoso na hora de codificar cada um dos coeficientes da imagem, pois existem técnicas bastante eficientes para codificar os coeficientes iguais a zero.

2. **Similaridade entre as bandas**

   A transformada *wavelet* decompõe a imagem em diversas bandas, o que permite que uma análise em múltiplas escalas. A figura 2.5 apresenta uma transformada *wavelet* bidimensional de n estágios, onde L_i, V_i, H_i e D_i representam as bandas passa baixo, vertical, horizontal e diagonal, respectivamente.

   Uma das principais características deste tipo de transformada é o facto de existir uma forte relação entre as bandas de frequências com a mesma orientação. Assim, tendo informação sobre um coeficiente com uma determinada posição em uma banda de frequência é possivel fazer alguma previsão sobre os coeficientes correspondentes que se encontrem numa outra banda de frequência com a mesma orientação. A figura 2.6 mostra de que forma se encontram relacionados os coeficientes nas diferentes bandas de frequência.

   As figuras 2.3b e 2.4b mostram uma transformada *wavelet* com 3 níveis de decomposição para as imagens LENA 256×256 e OCTÓGONO, respectivamente. Através destas figuras é possível confirmar a característica de similaridade entre as diversas bandas de frequência, pois pode-se observar que as imagens dos coeficientes que se encontram nas bandas com a mesma orientação são muito parecidas entre si. Dependendo da orientação das bandas estas ressaltam mais os detalhes da imagem que têm essa orientação. É também importante notar que as imagens dos coeficientes da transformada são muito similares à imagem original.

3. **Ganho de codificação**:

   Para um dado número de bits é possível obter um ganho de codificação quando se aloca um diferente número de bits para as diferentes bandas de frequência. Além disso, as diferentes sensibilidades do sistema visual humano às diferentes faixas de frequência permitem uma optimização subjectiva da qualidade da imagem [11].

Estas características da transformada *wavelet* são muito úteis na codificação de imagens, o que faz com que este tipo de transformada seja utilizada tanto nos al-

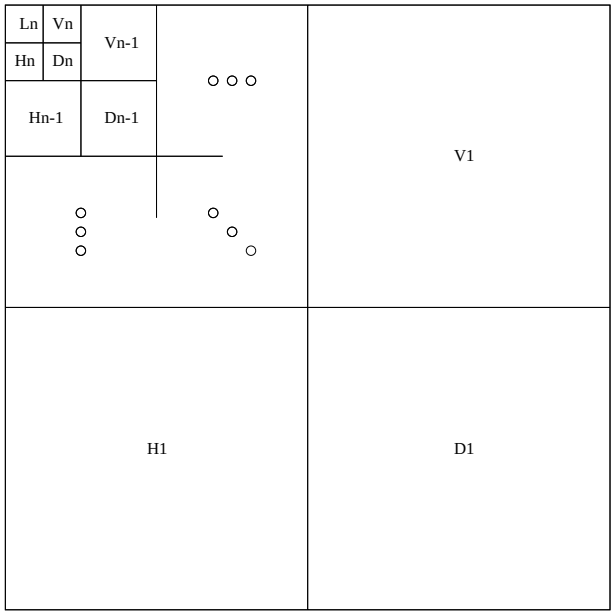


Figura 2.5: Transformada *Wavelet* bi-dimensional com n níveis de decomposição

goritmos já desenvolvidos e implementados no âmbito deste trabalho, como naqueles que são aqui desenvolvidos e igualmente implementados.

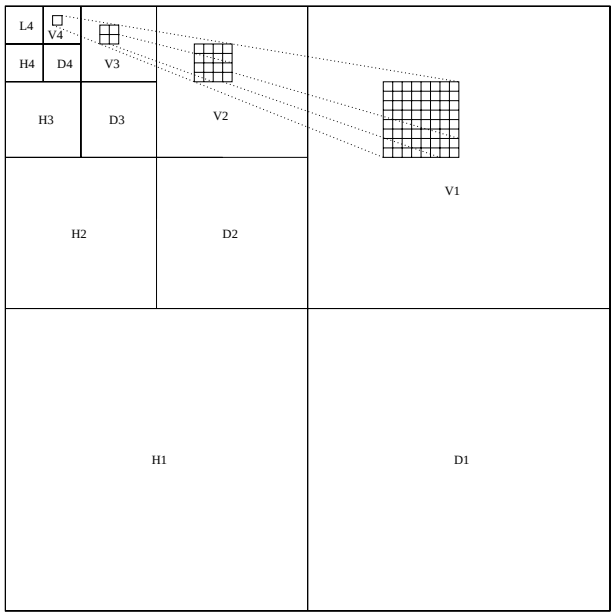


Figura 2.6: Coeficientes correspondentes nas bandas V_i, $i = 1, ..., 4$

Capítulo 3

Codificação de imagens usando planos de bits

3.1 Introdução

Após ser realizada a transformada da imagem, para uma base que seja mais conveniente para a compressão, é agora altura de quantizar os novos coeficientes que foram gerados. Ou seja, os coeficientes que até esta altura têm um grande número de valores distintos devem ser mapeados em um conjunto mais restrito de valores. Só após ser efectuada esta quantização é que é iniciada a codificação dos valores propriamente ditos.

A quantização dos coeficientes pode ser efectuada de forma escalar, onde cada coeficiente é mapeado individualmente em um novo valor, ou pode ser efectuada de forma vectorial, onde diversos coeficientes são mapeados conjuntamente.

Nos codificadores que têm vindo a ser desenvolvidos mais recentemente, estas duas técnicas de quantização encontram-se geralmente associadas a codificadores baseados no conceito de planos de bits.

Uma vez que na codificação por planos de bits os coeficientes da imagem transformada são codificados através de iterações sucessivas, isto é, em cada iteração é codificado um plano de bits dos coeficientes da imagem, a forma de quantização que mais se adequa a este método é aquela denominada *quantização por aproximações*

sucessivas, onde os coeficientes da transformada são comparados com um valor que vai diminuindo sucessivamente em cada iteração. Este tipo de quantização pode ser efectuado através de aproximações sucessivas de grandezas escalares ou vectoriais.

Neste capítulo, serão detalhadamente abordadas as técnicas de quantização usando planos de bits escalares e vectoriais, assim como será explicado o que se entende por codificadores que usam o conceito de planos de bits.

3.2 Quantização por aproximações sucessivas de escalares

O método de aproximações sucessivas de um escalar pode ser entendido observando a figura 3.1. Nesta figura vê-se que o segmento original de comprimento L é aproximado através da soma/subtracção de diversos segmentos de comprimento cada vez menor (no exemplo ilustrado o comprimento dos segmentos vai sendo multiplicado por 0,5 em cada iteração).

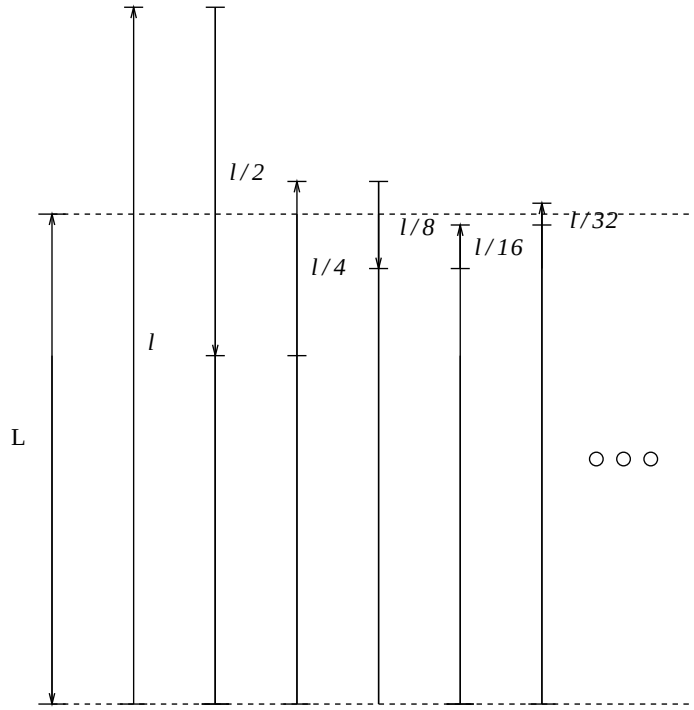


Figura 3.1: Aproximação sucessiva de um coeficiente para o caso escalar.

Assim, de acordo com a figura 3.1 o comprimento aproximado do segmento L é dado por:

$$L = l - \frac{l}{2} + \frac{l}{4} - \frac{l}{8} + \frac{l}{16} + \frac{l}{32} + ... + \frac{l}{2^n} \quad (3.1)$$

$$L = l - l_1 + l_2 - l_3 + l_4 + l_5 + ... + l_n \quad (3.2)$$

onde n representa o número de aproximações que se pertende fazer para representar o comprimento do segmento L original.

Matematicamente este método de quantização por aproximação sucessiva de um escalar L pode ser definido da seguinte forma:

$$L = \sum_{n=1}^{\infty} l_n b_n \quad (3.3)$$

$$l_n = l\alpha^n \quad (3.4)$$

onde:

- l representa o comprimento do segmento inicial para a aproximação;
- α representa o factor de escala dos segmentos l_n;
- n indica o número de iterações de aproximação efectuadas;
- $b_n \in \{-1, 1\}$.

Pode-se concluir, tanto através da observação da figura 3.1 como pela expressão matemática 3.4, que o erro final de aproximação, e_n vai depender de n, o número de iterações de aproximação que são efectuadas, e que quanto maior for esse número menor irá ser o erro na representação do comprimento do segmento original L.

Em [6] é mostrado que para que haja convergência ($\lim_{n\to\infty} e_n = 0$) o valor do factor escala deverá estar compreendido entre os seguintes valores:

$$\frac{1}{2} \leq \alpha < 1 \quad (3.5)$$

O erro de quantização em cada aproximação n depende então do factor de escala que é utilizado e tem a sua amplitude limitada a $0 < e_n \leq l\alpha^n$.

Contudo, para a obtenção da máxima eficiência na codificação, o valor do factor escala α deve ser o menor possível, pois quanto menor for este valor, menor será o número n de aproximações necessárias para se atingir o nível de erro desejado.

Pode-se concluir então que, para uma codificação mais eficiente que usa quantização por aproximação sucessivas de escalares, o valor escolhido para o factor de escala deve ser o menor que permita convergência, ou seja, $\alpha = 0,5$.

Como conclusão deste método de quantização pode-se dizer que, dado um segmento inicial l, é possível representar o segmento original L através de uma série de símbolos '+'e '−' e que o erro de quantização e_n vai diminuindo à medida que mais símbolos vão sendo adicionados.

Codificadores que utilizam este método de quantização por aproximação sucessiva de escalares em conjunto com a transformada *wavelet*, têm motivado muitos estudos, fazendo parte do estado da arte da codificação de imagens estáticas. Desta forma, têm vindo a ser desenvolvidos diversos codificadores de alto desempenho com estas caraterísticas.

O primeiro codificador deste tipo foi o **EZW** (*Embedded Zerotree Wavelet*) [12] e, seguindo o seu sucesso, foram desenvolvidos diversos algoritmos. Entre estes, podemos destacar dois deles. Em [1] foi apresentada uma variação do EZW que utiliza condicionamento estatístico e, de seguida, Said and Pearlman [15] propuseram o codificador **SPIHT** (*Set Partitional in Hierarchical Trees*) que apresenta desempenho significativamente superior em relação ao **EZW**.

Em [2] foi também apresentado o codificador **SWEET** (*Structured Wavelet Embedded Encoding Trees*) que, embora inspirado nos algoritmos **EZW** e **SPIHT** tem caraterísticas bastantes diferentes e que serão apresentadas na secção seguinte.

Mais recentemente, em [14] foi proposto um codificador, chamado de **MGE** (*Multigrid Embedding*) que possui um desempenho similar ao do **SPIHT**.

Estes algoritmos são descritos mais detalhadamente no capítulo 4.

3.3 Quantização por aproximações sucessivas de vectores

Muito recentemente, a ideia de quantização por aproximações sucessivas de um escalar foi generalizada e surgiu a chamada quantização por aproximações sucessivas de vectores. Isto é, em vez de aproximar sucessiva e individualmente cada coeficiente da transformada *wavelet*, passa-se a aproximar um conjunto de coeficientes dessa transformada, tratando-os como se fossem uma só unidade.

Em [4] e [8] é apresentado um método para implementar a quantização por aproximações sucessivas de um vector, que é uma simples extensão da aproximação sucessiva de escalares. Lá, os componentes que o compõem são aproximados sucessivamente por uma série de vectores cujas amplitudes são cada vez menores, ou seja, é aplicado a cada um dos componentes do vector o processo de aproximação sucessiva de um escalar.

A figura 3.2 mostra um exemplo de aproximações sucessivas de um vector $\vec{v}$ de duas dimensões com componentes v_1 e v_2 que podem ser expressas da seguinte forma:

$$\vec{v} = (v_1, v_2) \tag{3.6}$$

$$v_1 = +l_0 - \frac{l_0}{2} + \frac{l_0}{4} + \frac{l_0}{8} - \frac{l_0}{16} + \frac{l_0}{32} + ... \tag{3.7}$$

$$v_2 = -l_0 - \frac{l_0}{2} - \frac{l_0}{4} + \frac{l_0}{8} + \frac{l_0}{16} + \frac{l_0}{32} + .. \tag{3.8}$$

Na forma vectorial pode representar-se da seguinte forma:

$$\vec{v} = 1\begin{pmatrix} +l_0 \\ -l_0 \end{pmatrix}^t + \frac{1}{2}\begin{pmatrix} -l_0 \\ -l_0 \end{pmatrix}^t + \frac{1}{4}\begin{pmatrix} +l_0 \\ -l_0 \end{pmatrix}^t \tag{3.9}$$

$$+ \frac{1}{8}\begin{pmatrix} +l_0 \\ +l_0 \end{pmatrix}^t + \frac{1}{16}\begin{pmatrix} -l_0 \\ +l_0 \end{pmatrix}^t + \frac{1}{32}\begin{pmatrix} +l_0 \\ +l_0 \end{pmatrix}^t + ... \tag{3.10}$$

Generalizando tem-se:

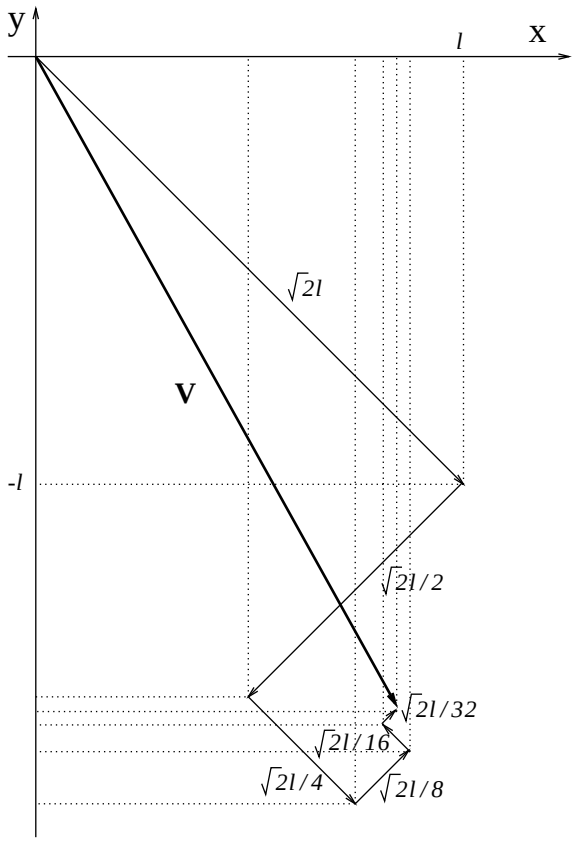


Figura 3.2: Exemplo de uma aproximação sucessiva de um vector bi-dimensional

$$\vec{v} = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} \sum_{n=1}^{\infty} b_{1,n} l_0 \alpha^n \\ \sum_{n=1}^{\infty} b_{2,n} l_0 \alpha^n \end{pmatrix} \tag{3.11}$$

$$\vec{v} = \sum_{n=1}^{\infty} \begin{pmatrix} b_{1,n} \\ b_{2,n} \end{pmatrix} l_0 \alpha^n = \sum_{n=1}^{\infty} \vec{b}_n l_0 \alpha^n \tag{3.12}$$

onde:

- l_0 indica o tamanho inicial do primeiro vector usado na aproximação;
- $\vec{b}_n$ representa a direcção do vector usado para a aproximação na iteração n;
- α representa o factor de escala usado na aproximação;
- n indica o número da iteração de aproximação.

Das formulações anteriores, pode-se mostrar que o vector $\vec{v}$ pode ser aproximado através de uma série de vectores, com amplitudes que vão diminuindo pro-

gressivamente e cujas orientações são variáveis e escolhidas entre um conjunto de vectores de orientação, chamado de **dicionário de vectores**. Assim, para o caso apresentado, um vector $\vec{v}$ com duas componentes v_1 e v_2, o dicionário escolhido para fazer a aproximação seria composto pelos seguintes vectores orientação:

$$u_1 = \{1, 1\}; \tag{3.13}$$

$$u_2 = \{1, -1\}; \tag{3.14}$$

$$u_3 = \{-1, 1\}; \tag{3.15}$$

$$u_4 = \{-1, -1\}; \tag{3.16}$$

Como cada vector orientação tem amplitude $\sqrt{2}$, na iteração n o vector $\vec{v}$ é aproximado por um vector cuja amplitude é $\frac{l_0\sqrt{2}}{2^{n-1}}$.

Assim, como conclusão deste exemplo, pode-se dizer que, dado um vector com uma amplitude inicial l_0, é possível representar o vector $\vec{v}$ enviando a cada iteração n apenas um símbolo c_n que vai representar o índice do vector que melhor representa o vector aproximado no conjunto de vectores de orientação $\{u_1, u_2, u_3, u_4\}$ existentes no dicionário.

Este processo pode ser generalizado para um vector $\vec{v}$ de dimensão k com componentes $\{v_1, v_2, ..., v_k\}$. Da mesma forma que para o exemplo apresentado, o vector $\vec{v}$ pode ser aproximado através de uma série de vectores, com amplitudes que vão diminuindo progressivamente e cujas orientações são variáveis e escolhidas entre um conjunto de vectores de orientação $\vec{u}_i$, definidos da seguinte forma:

$$\vec{u}_i = ((-1)^{\beta_{i1}}...(-1)^{\beta_{ik}}) \tag{3.17}$$

onde $\beta_{ij} \in \{0, 1\}$, com $j = 1, 2, ...k$.

É importante referir que os dicionários gerados a partir da equação 3.17 representam apenas uma extensão trivial do caso escalar. Contudo, existem outros dicionários que representam os seus vectores de orientação de forma diferente e que, provavelmente, tornam o método de quantização por aproximações sucessivas de

vectores bem mais eficiente. A forma como estes dicionários são gerados encontra-se descrita no apêndice B.

Generalizando, o vector $\vec{v}$ é representado da seguinte forma:

$$\vec{v} = \sum_{i=1}^{\infty} \vec{y}_{ni} V_0 \alpha^i \tag{3.18}$$

Há contudo, algumas considerações que necessitam ser feitas para que o método de quantização por aproximações sucessivas de vectores convirja ($\lim_{n\to\infty} e_n = 0$). A saber:

1. O dicionário Y tem que ser um conjunto finito de vectores de orientação com dimensão k e energia unitária, tal que:

$$Y = \{y_m : \|y_m\| = 1, m = 1, 2, ..., M\} \tag{3.19}$$

onde:

- y_m representa o vector de orientação de índice m dentro do dicionário Y;
- M é o número de vectores de orientação que o compõem.

Em cada iteração, a amplitude do novo vector é calculada como sendo o produto da amplitude do vector actual pelo vector de orientação $\vec{y}_i$ pertencente ao dicionário que melhor o representa. Ou seja:

$$v_n = \alpha^n V_0 \vec{y}_i \tag{3.20}$$

onde:

- v_n representa o novo vector que está a ser aproximado na iteração n;
- V_0 indica a amplitude do vector inicialmente escolhida (a amplitude deste vector tem que ser maior ou igual à amplitude do vector

original v);

- $\vec{y}_n$ representa o vector de orientação pertencente ao dicionário.

2. A amplitude do vector em cada iteração é escalada por um factor constante α pertencente ao intervalo $0,5 \leq \alpha < 1$.

Da mesma forma que para o caso escalar, pode-se assumir que, este método converge se, para um dado número de iterações de aproximação, o vector original puder ser aproximado, independentemente da amplitude que foi escolhida para o vector de aproximação inicial $\vec{V}_0$.

O dicionário tem que ser de tal forma, que o ângulo entre qualquer vector e o seu vector orientação é limitado por um $\Theta(Y)$. Desta forma, o máximo de erro que pode ser introduzido a cada iteração é atingido quando o vector residual, dado pela diferença entre o vector que se pretende codificar e o seu vector de aproximação, é aproximado por um vector que forma com ele um ângulo $\Theta(Y)$.

Utilizando as considerações expostas anteriormente, tem-se que as condições suficientes para garantir a convergência do método de aproximações sucessivas de vectores por um conjunto de vectores de orientação com amplitudes descendentes podem ser derivadas avaliando-se o pior caso, ou seja, quando os ângulos entre os vários vectores residuais e os correspondentes vectores orientação forem máximos ($\theta_1 = \theta_2 = ... = \Theta(Y)$). A figura 3.3 apresenta este caso. Definindo $\|\vec{r}_0\| = V_0$ a amplitude do vector residual a cada iteração n é dada por:

$$\|\vec{r}_n\|^2 = \|\vec{r}_{n-1}\|^2 + \alpha^{2n}\|\vec{V}_0\|^2 - 2\|\vec{r}_{n-1}\|\alpha^n\|\vec{V}_0\|\cos(\Theta(Y)) \tag{3.21}$$

onde:

- $\|\vec{r}_n\|$ é a norma do vector residual $\vec{r}_n$ na iteração i;
- $\|\vec{V}_0\|$ é a norma do vector inicial;
- α é o factor escalar;
- $\Theta(Y)$ é o ângulo máximo entre qualquer vector e o vector orientação que melhor o representa.

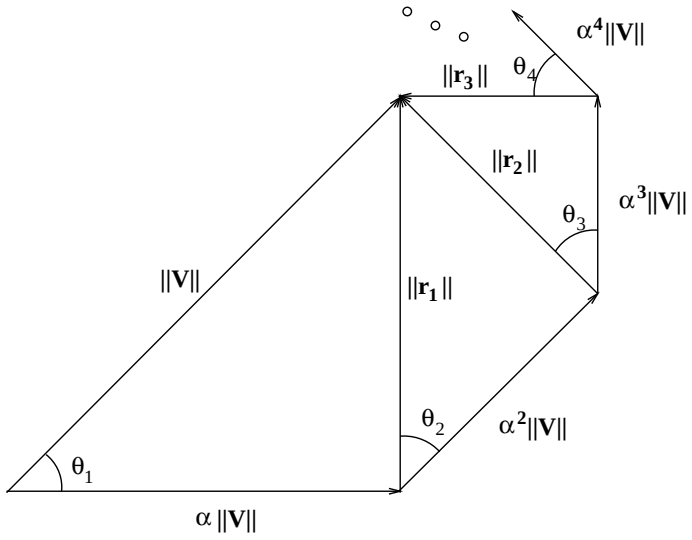


Figura 3.3: Análise de convergência para o caso de n-dimensões

Em 3.18 foi apresentada a equação geral para representação do vector $\vec{V}$, contudo, na realidade o número de aproximações é limitada por um valor N, assim:

$$\vec{v}_N = \sum_{i=1}^{N} \vec{y}_{ni} V_0 \alpha^i \tag{3.22}$$

onde:

- $\vec{v}_N$ representa o vector resultante da quantização por aproximações sucessivas de vectores;
- N representa o número de aproximações efectuadas para obter $\vec{V}_N$.

Usando a fórmula recursiva 3.21 e considerando a condição inicial $\|\vec{r}_0\| = V_0$, podem ser calculadas as amplitudes dos vectores residuais após cada iteração i, $\|\vec{r}_i\|, i = 1, 2, ..., n$ para qualquer par $(\alpha, \Theta(Y))$. Assim sendo, a convergência do método de aproximações sucessivas de vectores é dada por:

$$\lim_{n\to\infty} \|\vec{r}_n\| = 0 \tag{3.23}$$

Em [7] é mostrado que, se α for escolhido convenientemente, o erro $\|\vec{v} - \vec{v}_N\|$ tende a zero à medida que o número de aproximações tende a infinito. Assim, os valores de α para os quais isto se verifica são dados por:

$$\frac{1}{2\cos(\Theta(Y))} \leq \alpha < 1, \quad \Theta(Y) \leq 45^o \tag{3.24}$$

$$\sin(\Theta(Y)) \leq \alpha < 1, \quad \Theta(Y) \geq 45^o \tag{3.25}$$

Em [4] é mostrado que desde que haja convergência para um intervalo de valores de α o erro é dado por $\|r_n\| \leq V_0\alpha^n$

Como conclusão do acima apresentado, pode-se dizer que o método de quantização por aproximações sucessivas de vectores está fortemente relacionado com o valor de $\Theta(Y)$, valor este que depende do tipo de dicionário que está a ser utilizado. Assim sendo, a escolha do dicionário é muito importante para o sucesso ou insucesso deste método. Contudo, os dicionários contêm outros parâmetros que igualmente influenciam os resultados obtidos com este método, como é o caso da dimensão dos vectores e do número de vectores que o compõem.

No apêndice B são apresentados alguns reticulados regulares que são bons exemplos de dicionários para o uso com o SAWVQ.

3.4 Codificação de imagens usando planos de bits

Como foi referido anteriormente, os algoritmos para compressão de imagens estáticas desenvolvidos mais recentemente associam o método de transformada *wavelet* à codificação por planos de bits. A este tipo de codificação está normalmente associada a técnica de quantização de grandezas escalares e vectoriais apresentadas na secção anterior.

Após terem sido definidos os dois métodos de quantização por aproximação sucessiva, tanto de grandezas escalares como vectorias, é agora altura de passar a explicar o conceito de planos de bits quando associados a qualquer uma destas duas técnicas de quantização.

Numa codificação usando o conceito de planos de bits, os coeficientes da imagem transformada são codificados através de iterações sucessivas, isto é, em

cada iteração é codificado um plano de bits dos coeficientes da transformada da imagem. A figura 3.4 apresenta o conceito de planos de bits. Então, inicialmente é codificado o plano de bits mais significativo, n, para os coeficientes da transformada, depois o $n-1$ e assim sucessivamente até todos terem sido codificados ou até se ter atingido um outro critério de paragem.

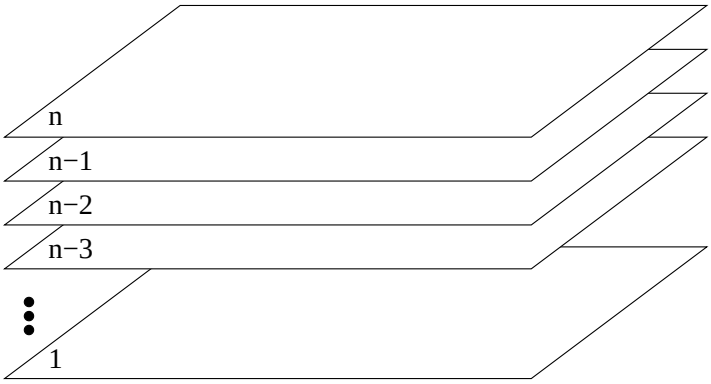


Figura 3.4: Vários planos de bits de uma imagem.

A quantização de um escalar $-1 \leq L \leq 1$ usando planos de bits pode ser representada por uma sequência $\{s, b_1,\ b_2 \ldots,\ b_n \ldots\}$ tal que:

$$L = s \sum_{i=1}^{\infty} b_i 2^{-i} \tag{3.26}$$

onde $s \in \{-1, 1\}$ representa o sinal de c e $b_i \in \{0, 1\}$.

Alternativamente, pode-se considerar que s seja sempre igual a 1 e $b_i \in \{-1, 1\}$. Assim sendo vem que:

$$L = \sum_{i=1}^{\infty} b_i 2^{-i} \tag{3.27}$$

Em aplicações práticas de codificação o somatório não é infinito, mas sim limitado a um valor P, que representa o número de planos de bits, e que se encontra diretamente relacionado com o nível de distorção pretendido para a imagem reconstruída.

Como foi apresentado na secção anterior, segundo [7] a representação usando planos de bits vectoriais envolve a decomposição do vetor $\vec{v}$ da seguinte forma:

$$\vec{v}_Q = \sum_{i=1}^{Q} \vec{y}_{ni} \alpha^{-i} \tag{3.28}$$

onde $\vec{y}_{ni}$ é um vetor unitário que pertence a um dado dicionário Y.

Como referido anteriormente, para que haja convergência, o valor do factor de escala tem que ser escolhido usando a equação 3.25.

A codificação por planos de bits é muito eficiente devido às características que são apresentadas em seguida.

1. **Controlo preciso da taxa de bits**:

   Através da codificação por planos de bits é possível escolher a altura exacta em que a codificação da imagem deve parar. Isto acontece porque, para cada plano de bits todos os coeficientes são varridos e a sua significância codificada. É importante referir que os primeiros coeficientes que são varridos são aqueles que contêm mais informação. Esta característica permite, além de um controlo preciso da taxa de bits, que uma imagem codificada para uma taxa de bits possa só ter uma parte descodificada, ou seja, pode-se descodificar apenas parte da imagem, dependendo da taxa de distorção que se pretende.

2. **Prioridade aos coeficientes mais importantes**:

   Usando planos de bits, a codificação dos coeficientes é efectuada de acordo com os seus valores absolutos. Ou seja, como a codificação é feita por planos de bits, aquele plano que é codificado inicialmente é aquele relacionado com as maiores amplitudes. Desta forma, os coeficientes que têm maior valor absoluto, que consequentemente são os que contêm mais informação, são codificados sempre primeiro. A grande vantagem desta caraterística é o facto de, quando for atingida a taxa de bits máxima pretendida, os coeficientes que foram codificados foram aqueles de maior importância.

3. **Igual distorção em todas as bandas**:

   Este tipo de codificação permite que em qualquer uma das bandas da transformada se tenha, a qualquer altura, aproximadamente a mesma distorção média, fazendo uma alocação de bits perfeita.

4. **Exploração hierárquica**:

   Com a associação da transformada *wavelet* com a codificação usando planos de bits é possível fazer uma exploração hierárquica da imagem. Isto porque este tipo de transformada decompõe a imagem a ser codificada em bandas de frequência onde, à medida que se vai descendo, a informação contida nessas bandas vai-se tornando menos importante. Assim, a codificação por planos de bits é iniciada na banda de mais baixa frequência, que é aquela que contém, em geral, as informações mais relevantes para a codificação da imagem, e segue codificando aquela banda que contém importância imediatamente inferior.

No capítulo seguinte faz-se uma descrição detalhada de alguns algoritmos que apresentam estas características.

Capítulo 4

Descrição dos Algoritmos

4.1 Introdução

Neste capítulo são descritos, detalhadamente, cada um dos algoritmos que foram implementados, tanto nas suas versões usando planos de bits escalares como nas suas adaptações para planos de bits vectoriais, segundo a tabela 4.1.

ESCALARES	**VECTORIAIS**
EZW	SAWVQ
EZW/C	SAWVQ/C
SPIHT	SPIHTVQ
MGE	MGEVQ
SWEET	SWEETVQ

Tabela 4.1: Algoritmos escalares e vectoriais.

De forma a melhor se entender a descrição que é feita para cada algoritmo, apresentam-se em seguida algumas das palavras chaves que são utilizadas.

- **Valor de referência**: É o valor que define um plano de bits. O bit referente a um coeficiente é dado pela comparação do valor absoluto do coeficiente com este valor de referência. Este valor é escolhido antes da codificação seguindo alguns critérios. Ele vai diminuindo à medida que vão sendo codificados os vários planos de bits, ou seja, cada plano de bits (n) tem um valor de referência menor do que o anterior ($n-1$).

- **Coeficiente** [1] **Significante**: Quando o valor absoluto de um dado coeficiente é maior do que um determinado valor de referência, representado aqui por "SS";

- **Coeficiente Insignificante**: Quando o valor absoluto de um dado coeficiente é menor do que um determinado valor de referência, representado aqui por "IS";

- **Coeficiente pai**: O coeficiente pai de um determinado coeficiente é aquele correspondente na banda da mesma orientação com frequência imediatamente inferior. Ver figura 2.6.

- **Coeficiente filho**: Os coeficientes filhos de um determinado coeficiente são aqueles correspondentes na banda da mesma orientação, mas com frequência imediatamente superior. Ver figura 2.6.

- **Árvore de Zeros**: O uso das árvores de zeros permite que sejam exploradas as similaridades entre as bandas da mesma orientação na transformada *wavelet*. Assim, caso se esteja perante uma árvore de zeros para um dado coeficiente, significa que tanto este como todos os correspondentes nas bandas de mesma orientação e com maior frequência são insignificantes para o valor referência em questão.

- **Decomposição em *quadtree***: A parte da imagem que está a ser codificada é dividida em 4 partes iguais, caso tenha pelo menos um coeficiente maior do que o valor de referência em questão.

- **Passo Dominante**: Este passo é comum a todos os algoritmos e é nele que os coeficientes são classificados como significantes ou insignificantes. Ou seja, para o valor de referência do plano de bits que está a ser codificado actualmente, o coeficiente é **significante** se o seu valor for maior que o valor

[1] Coeficiente é usado para o caso de planos de bits escalares; para planos de bits vectoriais são usados vectores de coeficientes.

referência, e é **insignificante** se o seu valor for menor ou igual ao valor referência.

- **Passo de Refinamento**: Este passo é igualmente comum a todos os algoritmos e é aqui que é enviado mais um plano de bits para todos aqueles coeficientes que já foram considerados significantes em algum dos planos de bits anteriores.

- **Codificador de símbolos**: Método como vai ser codificado o *bitstream* que vai sendo gerado.

- **Alfabeto**: Conjunto de diferentes símbolos que podem ser gerados durante a codificação da imagem.

Depois destas definições pode-se passar a descrever cada um dos algoritmos detalhadamente. Para cada um deles será apresentada a sua estrutura e de seguida a forma como foram implemetados, tanto na sua versão escalar como na sua adaptação para a forma vectorial.

Os resultados obtidos com os vários algoritmos são apresentados e comparados no capítulo 5.

4.2 Algoritmo EZW e SAWVQ

Nesta secção são descritos o algoritmo EZW e a sua extensão para planos de bits vectoriais, aqui denominado por SAWVQ.

4.2.1 Algoritmo EZW [12]

4.2.1.1 Estrutura do algoritmo

A estrutura do algoritmo EZW é a seguinte:

- **Transformada da imagem**: transformada *wavelet*;

- **Método de quantização**: aproximação sucessiva de escalares;

- **Unidade de codificação**: coeficiente;
- **Codificação do mapa de significâncias**: árvore de zeros;
- **Alfabeto**: 2 símbolos, 3 símbolos ou 4 símbolos;
- **Codificação dos símbolos**: codificador aritmético adaptativo.

O uso de árvores de zeros na codificação do mapa de significâncias é bastante eficiente, uma vez que poupa muitos bits ao enviar uma grande região onde todos os coeficientes são insignificantes para um dado plano de bits. Os coeficientes que são insignificantes nas diferentes escalas da transformada *wavelet* apresentam um grande factor de correlação. Desta forma, um agrupamento eficiente em árvores de zero permite um bom desempenho na codificação.

A figura 4.1 apresenta um esquema de árvores de zeros. Aqui pode-se ver que, para um dado coeficiente, à medida que se vai caminhando nas bandas de frequência, o número de coeficientes que se encontram correlacionados com ele vai aumentando. Assim, caso todos estes coeficientes que se encontram relacionados forem insignificantes, basta enviar um símbolo que indique esta característica, em vez de estar a ser enviado um símbolo individualmente.

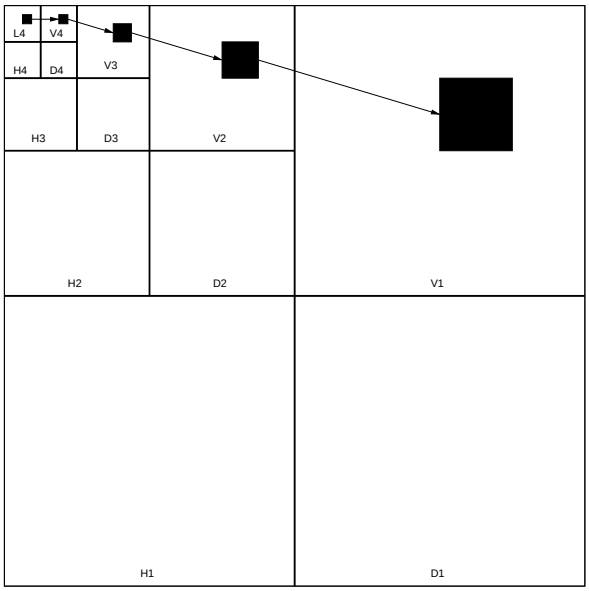


Figura 4.1: Agrupamento de coeficientes em árvore de zeros.

4.2.1.2 Linhas gerais do algoritmo

Os parâmetros de entrada deste algoritmo são os seguintes:

- Nome da imagem a codificar;
- Número de níveis de decomposição para a *wavelet* utilizada;
- Taxa de bits pretendida na codificação (bits/pixel).

Em termos gerais, a codificação de uma imagem estática usando este algoritmo é implementada da seguinte forma:

É calculada a transformada *wavelet* da imagem e as coordenadas dos coeficientes da transformada são colocadas em uma lista, chamada de **lista dominante**. Esta lista é varrida e os seus coeficientes codificados através de um processo de aproximação sucessiva de escalares.

No passo dominante, é enviado o mapa de significância dos coeficientes da transformada usando o conceito de árvore de zeros. Isto é efectuado da seguinte forma: em cada iteração os coeficientes da transformada são comparados com um dado valor de referência. Caso o valor absoluto do coeficiente seja significante as coordenadas correspondentes ao mesmo são colocadas em uma lista, denominada **lista subordinada** e é gerado um símbolo '+'ou '−' dependendo do sinal do coeficiente. Caso contrário, as coordenadas do coeficiente são colocadas numa lista, chamada de **lista temporária**. Aqui, podem ser gerados dois símbolos diferentes: primeiro, de raiz de árvore de zeros, 'ZT', quando todos os coeficientes correspondentes em todas as bandas de mais alta frequência também são considerados insignificantes, devendo todas as suas coordenadas serem colocadas na mesma lista temporária; segundo, o símbolo de zero isolado, 'Z' quando não se verifica a presença de uma árvore de zeros. A coordenada relativa a este coeficiente é igualmente colocada na lista temporária..

No passo subordinado, os coeficientes da transformada que já foram considerados significantes em alguma iteração anterior, são refinados de forma a enviar os planos de bits propriamente ditos.

O processo repete-se até ser atingida a taxa de bits pretendida.

4.2.1.3 Implementação do algoritmo

O algoritmo de codificação EZW pode ser descrito mais detalhadamente da seguinte forma:

1. A média da imagem é calculada e subtraída.

2. Aplica-se uma transformada *wavelet* com R estágios à imagem com média zero.

3. Dependendo da banda que está a ser considerada, o varrimento dos coeficientes de cada banda é feito de forma diferente. É efectuado um varrimento vertical nas bandas verticais, um horizontal nas bandas horizontais e em *zig-zag* nas bandas diagonais. A ordem de varrimento dos coeficientes nas diferentes bandas de frequência encontra-se ilustrada na figura 4.2.

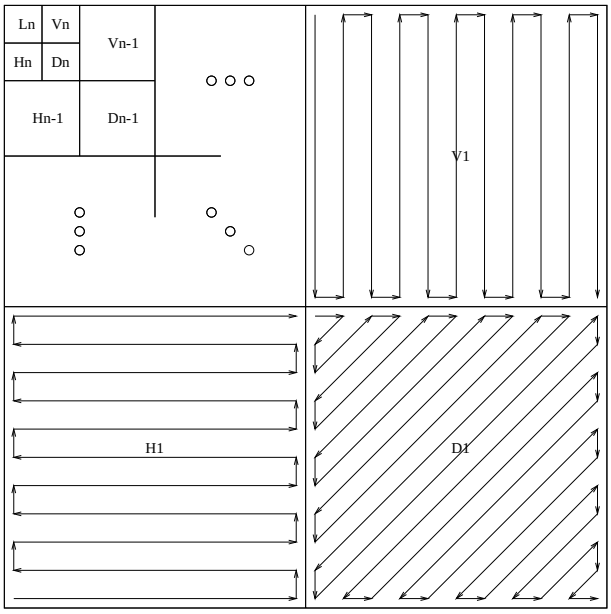


Figura 4.2: Ordem de varrimento dos coeficientes nas diferentes bandas de frequência.

4. Escolhe-se um valor de referência inicial l_0 como sendo metade do maior valor absoluto dos coeficientes da transformada.

5. É criada uma lista de coordenadas dos coeficientes, chamada de **lista dominante**, que define a ordem segundo a qual os coeficientes são varridos. Esta

ordem é tal que os coeficientes das bandas com mais baixa frequência são sempre varridos primeiro do que aqueles que se encontram em bandas com frequência mais alta. São criadas duas listas vazias, a **lista subordinada** e a **lista temporária**.

6. A transformada *wavelet* da imagem é varrida, e se um coeficiente da *wavelet* é menor que o valor de referência l_n para a iteração actual n este é reconstruído como tendo o valor zero. Caso contrário, é reconstruído com o valor $\pm\frac{3l_n}{2}$, de acordo com o seu sinal.

7. *Passo dominante*: Os coeficientes reconstruídos são varridos novamente, de acordo com a ordem com que se encontram na lista dominante, gerando o chamado mapa de significâncias. Este processo é efectuado da seguinte forma: Se um coeficiente foi reconstruído, no passo anterior, como sendo positivo ou negativo, um símbolo '+'ou '−' é gerado e adicionado a uma *string* e em seguida as coordenadas deste coeficiente são adicionadas no final da lista subordinada. Caso um coeficiente tenha sido reconstruído com o valor zero, no passo anterior, as suas coordenadas são adicionadas à lista temporária e podem ser gerados dois símbolos diferentes:

   - Caso todos os coeficientes correpondentes nas bandas com a mesmo orientação mas com frequência mais alta forem insignificantes, um símbolo de *raiz de árvore de zeros*, 'ZT', é gerado e adicionado à *string*, e as coordenadas correspondentes a todos estes coeficientes são removidos da lista dominante e adicionados à lista temporária (uma vez que já se sabe que são insignificantes para o valor de referência l_n, não necessitam de ser varridos);
   - Se a situação anterior não se verificar, ou seja, não se esteja perante uma árvore de zeros, um símbolo de *zero isolado*, 'Z', é gerado e adicionado à *string*. A coordenada referente ao coeficiente é removida da lista dominante e colocada na lista temporária.

   À medida que vai sendo gerada a *string* de símbolos '+', '−', 'ZT'e 'Z', para

codificar o mapa de significâncias, ela vai sendo codificada em um *bitstream* usando o codificador aritmético adaptativo apresentado em [17]. Como neste algoritmo podem ser gerados 4 símbolos distintos, o codificador aritmético adaptativo é actualizado em cada passo para o uso desses 4 símbolos. Contudo, se estiverem a ser varridos os coeficientes que se encontram na banda de mais alta frequência (H_1, V_1 e D_1 na figura 4.2), deixam de existir raízes de árvores de zeros, devendo por isso, neste caso, o codificador aritmético ser actualizado para apenas 3 símbolos.

8. O valor de referência l_n é dividido por 2, ficando agora com o valor l_{n+1}.

9. *Passo subordinado ou de refinamento*: É durante este passo que os coeficientes são refinados, ou seja, os coeficientes que nos passos anteriores foram reconstruídos com valores diferentes de zero, são novamente varridos de acordo com a ordem com que se encontram na lista subordinada. A cada um destes coeficientes é adicionado o valor $+\frac{l_{n+1}}{2}$ ou então $-\frac{l_{n+1}}{2}$, com o objectivo de diminuir o erro de reconstrução. Caso o valor somado seja $+\frac{l_{n+1}}{2}$, é gerado um símbolo '+' e enviado para a *string*, caso contrário é gerado e enviado para a *string* o símbolo '−'. No final do passo subordinado a lista subordinada é reordenada de forma a que as coordenadas dos coeficientes que têm maior amplitude fiquem sempre primeiro. Os símbolos '+'e '−' gerados neste passo são também codificados em um *bitstream* usando o codificador aritmético. Assim, como neste caso apenas existem 2 símbolos diferentes, no início deste passo o codificador aritmético deve ser actualizado para 2 símbolos.

10. A lista dominante é substituida pela temporária e esta é novamente esvaziada.

11. O valor de n é incrementado e o processo é repetido desde o passo 6, e é interrompido em qualquer ponto quando o tamanho do *bitstream* exceder um limite pré-estabelecido.

O *bitstream* possui um cabeçalho de 10 bytes com informação extra necessária ao algoritmo de descodificação. Ele contém:

- o número de estágios (1 byte);
- as dimensões da imagem (4 bytes);
- a média da imagem (1 byte);
- o valor de referência inicial (4 bytes).

A lista dominante é inicializada da mesma forma para o codificador e para o descodificador. As restantes listas vão sendo actualizadas pelo descodificador à medida que a imagem é reconstruída. Desta forma, o descodificador segue exactamente os mesmos passos do codificador, descodificando correctamente o *bitstream* que foi gerado.

4.2.1.4 Análise do algoritmo

O algoritmo descrito acima possui algumas características importantes que o tornam bastante eficiente. Pode-se citar:

- O uso de árvores de zeros permite explorar de uma forma muito eficiente a similaridade entre as diferentes bandas de frequência da transformada *wavelet*;
- O uso de um alfabeto muito pequeno para codificar uma imagem permite que o codificador aritmético se adapte muito rapidamente a qualquer mudança nas estatísticas do símbolos;
- O facto de os coeficientes serem codificados de acordo com o seu valor absoluto, dando sempre prioridade aqueles que são maiores. Esta característica faz com que o nível de distorção seja sempre o menor possível pois, quando o processo é interrompido, é garantido que os coeficientes não codificados são sempre aqueles que apresentam menor valor absoluto.
- É possível interromper o processo em qualquer altura, dependendo da taxa de bits que se pretende para a codificação da imagem.

4.2.2 Algoritmo SAWVQ [6]

4.2.2.1 Estrutura do algoritmo

As principais mudanças que existem entre o EZW e o SAWVQ são duas. A primeira é que passam a ser processados vectores de coeficientes em vez de cada coeficiente individualmente. A segunda modificação é o facto do valor de referência deixar de ser dividido por 2 e passar a ser multiplicado por um factor de escala $(0,5 \leq \alpha < 1)$.

A estrutura do algoritmo SAWVQ é a seguinte:

- **Transformada da imagem**: transformada *wavelet*;
- **Método de quantização**: aproximação sucessiva de vectores;
- **Unidade de codificação**: vector de coeficientes com dimensão MN;
- **Codificação do mapa de significâncias**: árvore de zeros.
- **Alfabeto**: 3 símbolos ou $m + 1$ símbolos, onde m corresponde ao número de vectores de orientação que constituem o dicionário;
- **Codificação dos símbolos**: codificador aritmético adaptativo.

4.2.2.2 Linhas gerais do algoritmo

Os parâmetros de entrada deste algoritmo são os seguintes:

- Nome da imagem a codificar;
- Número de níveis de decomposição para a *wavelet* utilizada;
- Taxa de bits pretendida na codificação (bits/pixel);
- Valor do factor de escala α.

Em termos gerais, a codificação de uma imagem estática, usando este algoritmo, é implementada da seguinte forma:

É calculada a transformada *wavelet* da imagem e as coordenadas dos vectores de coeficientes da transformada são colocadas em uma lista, chamada de **lista dominante**. Esta lista é varrida e os seus vectores codificados através de um processo de aproximação sucessiva de grandezas vectoriais.

No passo dominante, é enviado o mapa de significância dos vectores de coeficientes da transformada usando o conceito de árvore de zeros. Isto é efectuado da seguinte forma: em cada iteração os módulos das amplitudes dos vectores são comparados com um dado valor de referência. Caso este valor seja maior, as coordenadas correspondentes ao mesmo são colocadas em uma lista, denominada **lista subordinada** e é gerado um símbolo 'C_j' correspondente ao índice do vector de orientação mais próximo no dicionário de vectores escolhido. Caso contrário, as coordenadas do vector são colocadas numa lista, chamada de **lista temporária**. Aqui podem ser gerados dois símbolos diferentes. Primeiro, de raiz de árvore de zeros, 'ZT', quando todos os vectores de coeficientes correspondentes em todas as bandas de mais alta frequência também são considerados insignificantes, devendo todas as suas coordenadas serem colocadas na mesma lista temporária. Segundo, o símbolo de zero isolado, 'Z' quando não se verifica a presença de uma árvore de zeros. A coordenada correspondente a este vector é igualmente colocada na lista temporária.

No passo subordinado, os vectores que já foram considerados significantes em alguma iteração anterior, são refinados de forma a enviar mais um plano de bits.

O processo repete-se até ser atingida a taxa de bits pretendida.

4.2.2.3 Implementação do algoritmo

O algoritmo de codificação SAWVQ pode ser descrito da seguinte forma:

1. A média da imagem é calculada e subtraída.

2. Aplica-se uma transformada *wavelet* com R estágios à imagem com média zero.

3. Cada banda dos coeficientes da *wavelet* é dividida em blocos $M \times N$, formando vectores de dimensão MN. Dependendo da banda que está a ser considerada, o varrimento dos blocos é feito de forma diferente. É efectuado um varrimento

vertical nas bandas verticais, um horizontal nas bandas horizontais e em *zig-zag* nas bandas diagonais. A ordem de varrimento dos blocos nas diferentes bandas de frequência encontra-se ilustrada na figura 4.3.

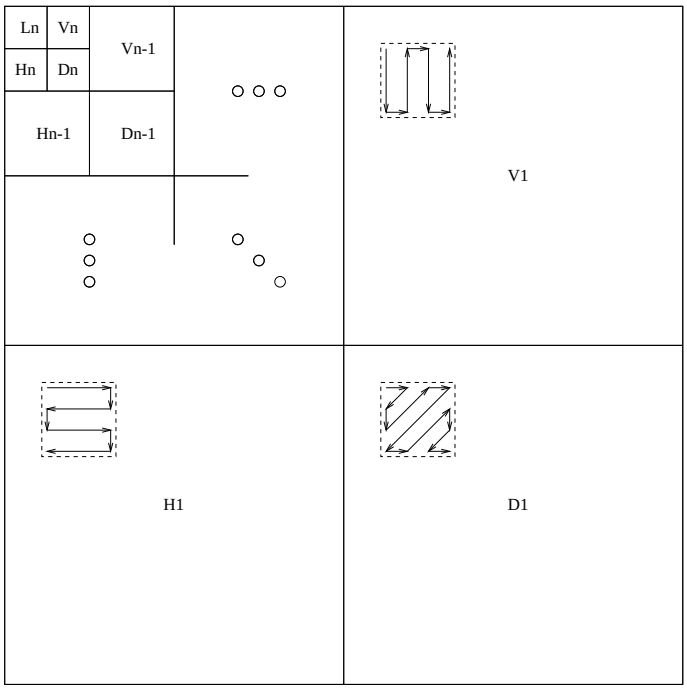


Figura 4.3: Varrimento de cada bloco de coeficientes para formar os vectores em cada banda de diferentes orientações.

4. É encontrado o vector de coeficientes da transformada que tem maior amplitude $\|\vec{V}\|_{max}$.

5. O valor de referência inicial, l_0, é dado por $\alpha\|\vec{V}\|_{max}$, onde α depende do dicionário de vectores de orientação que é utilizado.

6. É criada uma lista com as coordenadas dos vectores, chamada de **lista dominante**, que define a ordem segundo a qual estes são varridos. Esta ordem é tal que os vectores das bandas com mais baixa frequência são sempre varridos primeiro do que aqueles que se encontram em bandas com frequência mais alta. São criadas duas listas vazias, a **lista subordinada** e a **lista temporária**.

7. A transformada *wavelet* da imagem é varrida, e se a amplitude de um vector de coeficientes da *wavelet* é menor que o valor de referência l_n para a iteração

actual n, este é reconstruído como tendo o valor zero. Caso contrário, é reconstruído com o vector de orientação mais próximo escalado pelo valor de referência da iteração actual, l_n.

8. *Passo dominante*: Os vectores de coeficientes reconstruídos são varridos novamente, de acordo com a ordem com que se encontram na lista dominante, gerando o chamado mapa de significâncias. Este processo é efectuado da seguinte forma: Se um vector de coeficientes não foi reconstruído com o valor zero, no passo anterior, um símbolo 'C' é gerado e adicionado à *string* e em seguida as coordenadas deste vector são adicionadas no final da lista subordinada. É enviado para a *string* o índice do vector mais próximo pertencente ao dicionário. Assim, no início deste passo, o codificador aritmético deve ser actualizado de forma a permitir tantos símbolos como o número de vectores de orientação que constituem o dicionário. Caso um vector coeficiente tenha sido reconstruído com o valor zero, no passo anterior, as suas coordenadas são adicionadas à lista temporária e podem ser gerados dois símbolos diferentes:

   - Caso todos os vectores de coeficientes correpondentes nas bandas com a mesmo orientação mas com frequência mais alta forem insignificantes, um símbolo de *raiz de árvore de zeros*, 'ZT', é gerado e adicionado à *string*, e as coordenadas correspondentes a todos estes vectores são removidas da lista dominante e adicionadas à lista temporária (uma vez que já se sabe que são insignificantes para o valor de referência actual l_n, não necessitam de ser varridos);
   - Se a situação anterior não se verificar, ou seja, não se esteja perante uma árvore de zeros, um símbolo de *zero isolado*, 'Z', é gerado e adicionado à *string*. A coordenada referente ao vector de coeficientes é removida da lista dominante e colocada na lista temporária.

À medida que vai sendo gerada a *string* de símbolos 'C', 'ZT'e 'Z', para codificar o mapa de significâncias, ela vai sendo codificada em um *bitstream* usando um codificador aritmético adaptativo [17]. Como neste algoritmo podem ser

gerados 3 símbolos distintos, o codificador aritmético adaptativo é actualizado em cada passo para o uso desses 3 símbolos. Contudo, se estiverem a ser varridos os coeficientes que se encontram na banda de mais alta frequência (H_1, V_1 e D_1 na figura 4.3), deixam de existir raízes de árvores de zero, devendo por isso, neste caso, o codificador aritmético ser actualizado para apenas 2 símbolos.

9. O valor de referência l_n é multiplicado pelo factor de escala α, ficando agora com o valor l_{n+1}.

10. *Passo subordinado ou de refinamento*: É durante este passo que os vectores de coeficientes são refinados, ou seja, os vectores que nos passos anteriores foram reconstruídos com valores diferentes de zero, são novamente varridos de acordo com a ordem com que se encontram na lista subordinada. Neste processo de refinamento, é codificada a diferença entre o vector original e o vector reconstruído até ao momento, utilizando-se o novo valor de referência, l_{n+1}, e um vector de orientação escolhido do dicionário. Os índices destes novos vectores de orientação são codificados em um *bitstream* usando um codificador aritmético, que no início do passo 9 já foi inicializado para permitir o número de vectores que constitui o dicionário. No final do passo subordinado a lista subordinada é reordenada de forma a que as coordenadas dos vectores de coeficientes que têm maior amplitude fiquem sempre primeiro.

11. A lista dominante é substituída pela temporária e esta é novamente esvaziada.

12. O valor n é incrementado e o processo é repetido desde o passo 7, e é interrompido em qualquer ponto quando o tamanho do *bitstream* exceder um limite pré-estabelecido.

O *bitstream* possui um cabeçalho de 11 bytes com informação extra necessária ao algoritmo de descodificação. Ele contém:

- o valor de α (1 byte);
- o número de estágios (1 byte);

- as dimensões da imagem (4 bytes);
- a média da imagem (1 byte);
- o valor de referência inicial (4 bytes).

Tal como para o caso do algoritmo EZW, a lista dominante é inicializada da mesma forma para o codificador e para o descodificador. As restantes listas vão sendo actualizadas pelo descodificador à medida que a imagem é reconstruída. Desta forma, o descodificador segue exactamente os mesmos passos do codificador, descodificando correctamente o *bitstream* que foi gerado.

4.2.2.4 Análise do algoritmo

O SAWVQ apresenta quase as mesmas características que as apresentadas para o algoritmo EZW. Pode-se citar:

- O uso de árvores de zeros permite explorar de uma forma muito eficiente a similaridade entre as diferentes bandas de frequência da transformada *wavelet*;
- O facto de os vectores coeficientes serem codificados de acordo com o módulo da sua amplitude, dando sempre prioridade aqueles que são maiores. Esta característica faz com que o nível de distorção seja sempre o menor possível pois, quando o processo é interrompido é garantido que os vectores não codificados são sempre aqueles que apresentam menor módulo de amplitude.
- É possível interromper o processo em qualquer altura, dependendo da taxa de bits que se pretende para a codificação da imagem.

4.2.3 Resumo das principais características do EZW e do SAWVQ

Na tabela 4.2 é apresentado um resumo das principais características dos algoritmos EZW e SAWVQ.

	EZW	**SAWVQ**
Transformada da imagem	Transformada *wavelet*	Transformada *wavelet*
Método de quantização	Aproximação sucessiva de escalares	Aproximação sucessiva de vectores
Unidade de codificação	Coeficiente	Vector de coeficientes
Codificação mapa de significâncias	Árvore de zeros	Árvore de zeros
Alfabeto	2, 3 ou 4 símbolos	3 ou $m+1$ símbolos
Codificação dos símbolos	Codificador aritmético adaptativo	Codificador aritmético adaptativo
Valor de referência	Dividido por 2 a cada iteração	Multiplicado por α a cada iteração, $0,5 \leq \alpha < 1$

Tabela 4.2: Principais características da implementação dos algoritmos EZW e SAWVQ.

Sobre as diferenças apresentadas é importante ressaltar o número de símbolos que é necessário usar para a representação da imagem para cada um dos algoritmos. Enquanto, que o EZW usa no máximo 4 símbolos, o SAWVQ usa bastantes mais, sendo que este número vai depender do número de vectores de orientação que constituem o dicionário, m. Assim, existe uma perda de eficiência clara na parte de codificação por entropia que é consequência do facto do aumento significativo do número de símbolos. No entanto, uma vez que a quantização é efectuada de forma vectorial, ou seja, de cada vez que é codificado o mapa de significâncias de um vector na realidade estão a ser codificados MN coeficientes da transformada, existe uma economia muito grande nos símbolos cujo envio é necessário para codificar esse mapa de significâncias (MN vezes menos símbolos).

4.3 Algoritmo EZW/C e SAWVQ/C

Nesta secção são descritos o algoritmo EZW/C e a sua extensão para planos de bits vectoriais, aqui denominado por SAWVQ/C.

Este algoritmo, tal como o próprio nome indica, é uma adaptação do EZW onde a principal diferença diz respeito à forma como é codificado o mapa de significâncias. Aqui deixa de ser feita uma previsão sobre os coeficientes que se encontram nas bandas de mais altas frequências e passa a ser feito um condicionamento estatístico na codificação destes coeficientes, ou seja, a codificação por árvores de zeros é substituída por uma codificação dependente de contextos da informação de significância. Assim, o contexto usado pelo codificador dos símbolos tem que alternar levando em consideração os seguintes parâmetros:

- passo corrente (envio do mapa de significâncias ou do plano de bits propriamente dito);
- nível de decomposição da *wavelet* onde se encontra o coeficiente;
- significância do coeficiente vizinho;
- significância do coeficiente pai.

No caso da codificação referente aos sinais dos coeficientes, é igualmente feito um condicionamento estatístico, agora dos seguintes parâmetros:

- nível de decomposição da *wavelet* onde se encontra o coeficiente cujo sinal está a ser enviado;
- sinal do coeficiente vizinho;
- sinal do coeficiente pai.

Como exemplo, pode-se dizer que para *wavelet* com 6 níveis de decomposição são usados 102 contextos distintos.

Outra diferença que pode ser ressaltada entre este algoritmo e o EZW é o facto de neste deixar de existir o passo dominante e o passo subordinado separadamente.

Ou seja, durante o mesmo passo são enviados o mapa de significâncias e os planos de bits propriamente ditos.

4.3.1 Algoritmo EZW/C [1]

4.3.1.1 Estrutura do algoritmo

A estrutura do algoritmo EZW/C é a seguinte:

- **Transformada da imagem**: transformada *wavelet*;
- **Método de quantização**: aproximação sucessiva de escalares;
- **Unidade de codificação**: coeficiente;
- **Codificação do mapa de significâncias**: condicionamento estatístico.
- **Alfabeto**: 2 símbolos;
- **Codificação dos símbolos**: codificador aritmético adaptativo.

4.3.1.2 Linhas gerais do algoritmo

Os parâmetros de entrada deste algoritmo são os seguintes:

- Nome da imagem a codificar;
- Número de níveis de decomposição para a *wavelet* utilizada;
- Taxa de bits pretendida na codificação (bits/pixel).

Em traços gerais, o algoritmo EZW/C pode ser descrito da seguinte forma:

É calculada a transformada *wavelet* da imagem e as coordenadas dos coeficientes da transformada são colocadas em uma lista, chamada de **lista dominante** e todas classificadas como sendo insignificantes, 'IS'. Assim, na primeira vez que esta lista é varrida, todos os coeficientes estão marcados como 'IS' e por isso todos são comparados com um dado valor de referência de forma a ser enviado o mapa de significâncias. Este mapa de significâncias é agora enviado usando um condicionamento

estatístico implementado da seguinte forma: Caso o coeficiente seja significante, ele é marcado na lista dominante como 'SS' e gerado o símbolo correspondente a coeficiente significante. Este símbolo é então codificado dependendo do contexto em que se encontra. Neste caso, tem que ser igualmente enviado o sinal do coeficiente, tendo também por base o contexto onde este se encontra. No caso do coeficiente continuar a ser insignificante, ele é marcado como 'IS' e gerado o símbolo correspondente a esta situação. Este símbolo é também codificado tendo em consideração o contexto em que o coeficiente se encontra.

No entanto, na segunda vez que a lista dominante é varrida, passam a existir coeficientes 'IS' e 'SS'. Para aqueles que se encontram marcados por 'IS' todo o processo é repetido, sendo que agora o valor de referência já é menor (foi dividido por 2). Os restantes, marcados com 'SS', são refinados e os símbolos gerados codificados conforme o contexto em que se encontram.

O processo termina quando a taxa de bits pretendida é atingida.

4.3.1.3 Implementação do algoritmo

O algoritmo de codificação EZW/C pode ser descrito mais detalhadamente da seguinte forma:

1. A média da imagem é calculada e subtraída.
2. Aplica-se uma transformada *wavelet* com R estágios à imagem com média zero.
3. Dependendo da banda que está a ser considerada, o varrimento dos coeficientes de cada banda é feito de forma diferente. É efectuado um varrimento vertical nas bandas verticais, um horizontal nas bandas horizontais e em *zig-zag* nas bandas diagonais. A ordem de varrimento dos coeficientes nas diferentes bandas de frequência encontra-se ilustrada na figura 4.2.
4. Escolhe-se um valor de referência inicial l_0 como sendo metade do maior valor absoluto dos coeficientes da transformada.
5. É criada uma lista de coordenadas dos coeficientes, chamada de **lista dominante**, que define a ordem segundo a qual os coeficientes são varridos. Esta

ordem é tal que os coeficientes das bandas com mais baixa frequência são sempre varridos primeiro do que aqueles que se encontram em bandas com frequência mais alta.

6. Inicialmente a lista dominante é varrida e todos os coeficientes da transformada são marcados como sendo insignificantes, 'IS'.

7. *Passo dominante e de refinamento*: Este passo deve ser dividido em duas fases, uma correspondente à primeira vez que a lista dominante é varrida e a outra para as restantes vezes. Assim sendo, tem-se:

   (a) A primeira vez que a lista é varrida, todos os coeficientes estão marcados com sendo insignificantes. Então, todos são comparados com o valor de referência de forma a ver quais os que se tornaram significantes nesta iteração. Podem surgir duas situações distintas:

      i. Quando o valor absoluto do coeficiente é menor que o valor de referência, o coeficiente continua marcado como sendo 'IS' na lista dominante e o coeficiente é reconstruído com o valor zero. É gerado um símbolo '0' que é adicionado à *string*, tendo em conta os parâmetros usados para o condicionamento estatístico que foram apresentados acima.

      ii. Quando o valor absoluto do coeficiente é maior que o valor de referência, ele é marcado como sendo 'SS' na lista dominante e é gerado o símbolo '1' que é adicionado à *string* dependendo do contexto em que se encontra. Neste caso, é também enviado o sinal '1'ou '0' dependendo se o valor do coeficiente é positivo ou negativo sendo o coeficiente reconstruído com o valor $\pm\frac{3l_n}{2}$, dependendo do sinal. O símbolo gerado também é enviado para a *string* levando em consideração o condicionamento estatístico definido para os sinais.

   (b) A partir da segunda vez que a lista dominate é varrida, nela já vão existir coeficientes marcados como sendo 'IS' e outros como sendo 'SS'. Assim tem-se:

i. Para o caso dos coeficientes que se encontram marcados como 'IS' o processo é efectuado da mesma forma como se fosse a primeira vez que a lista está a ser varrida, sendo a unica diferença o facto do valor de referência ter diminuído.

ii. Para o caso dos coeficientes que estão marcados como 'SS', estes são refinados, através do envio de mais um plano de bits, de forma a que o seu erro de reconstrução diminua. Desta forma, é adicionado a este coeficiente na imagem reconstruída o valor de $\pm\frac{l_n}{2}$ dependendo se o valor do coeficiente é maior ou menor do que o valor que se encontra na imagem reconstruída, gerando os símbolos '+' e '−' respectivamente. Estes símbolos também são codificados dependendo do contexto onde se encontram.

Tanto para codificar o mapa de significâncias como os planos de bits propriamente ditos, é suficiente utilizar os símbolos '1' e '0'. Desta forma, a *string* de símbolos gerada vai ser codificada utilizando o mesmo codificador aritmético apresentado em [17], que tem que ser actualizado a cada iteração para o uso de 2 símbolos.

8. O valor de referência é dividido por 2.

9. o valor n é incrementado e o processo é repetido desde o passo 7, e é interrompido em qualquer ponto quando o tamanho do *bitstream* exceder um limite pré-estabelecido.

O *bitstream* possui um cabeçalho de 10 bytes com informação extra necessária ao algoritmo de descodificação. Ele contém:

- o número de estágios (1 byte);
- as dimensões da imagem (4 bytes);
- a média da imagem (1 byte);
- o valor de referência inicial (4 bytes).

O codificador e o descodificador têm as suas listas dominantes inicializadas da mesma forma e por isso encontram-se perfeitamente sincronizados, permitindo que o *bitstream* gerado seja descodificado correctamente. É igualmente necessário que, durante o processo de descodificação, seja levado em consideração o contexto em que os coeficientes foram codificados. Desta forma, tem que existir uma cópia dos contextos do codificador no descodificador.

4.3.1.4 Análise do algoritmo

As características que tornam este algoritmo eficiente são muito idênticas às do EZW, sendo a única diferença a substituição das árvores de zeros pelos contextos. Desta forma, para este algoritmo pode-se citar o seguinte:

- O uso de contextos distintos permite que nunca aconteça o facto de um estar demasiadamente saturado e outro quase vazio, permitindo que haja um equilíbrio grande entre todos.
- O uso de um alfabeto com apenas dois símbolos para codificar uma imagem, permite que o codificador aritmético se adapte muito rapidamente a qualquer mudança nas estatísticas dos símbolos;
- O facto de os coeficientes serem codificados de acordo com o seu valor absoluto, dando sempre prioridade àqueles que são maiores. Esta característica faz com que o nível de distorção seja sempre o menor possível, pois, quando o processo é interrompido, é garantido que os coeficientes não codificados são sempre aqueles que apresentam menor valor absoluto.
- É possível interromper o processo em qualquer altura, dependendo da taxa de bits que se pretende para a codificação da imagem.

4.3.2 Algoritmo SAWVQ/C

4.3.2.1 Estrutura do algoritmo

Este algoritmo é uma adaptação do EZW com condicionamento estatístico (EZW/C) e foi desenvolvido no âmbito deste trabalho.

As principais mudanças que existem entre o EZW/C e a sua adaptação a planos de bits vectoriais, o SAWVQ/C dizem respeito à unidade de codificação pois deixam de ser codificados coeficientes individualmente e passam a ser codificados vectores de coeficientes. Outra mudança importante reside no facto do valor de referência deixar de ser dividido por 2 e passar a ser multiplicado por um factor de escala $(0,5 \leq \alpha < 1)$.

A estrutura do algoritmo SAWVQ/C é a seguinte:

- **Transformada da imagem**: Transformada *wavelet*;
- **Método de quantização**: aproximação sucessiva de vectores;
- **Unidade de codificação**: vectores de coeficientes com dimensão MN;
- **Codificação do mapa de significâncias**: condicionamento estatístico.
- **Alfabeto**: 2 símbolos, $m + 1$ símbolos, onde m corresponde ao número de vectores de orientação que constituem o dicionário;
- **Codificação dos símbolos**: codificador aritmético adaptativo.

4.3.2.2 Linhas gerais do algoritmo

Os parâmetros de entrada deste algoritmo são os seguintes:

- Nome da imagem a codificar;
- Número de níveis de decomposição para a *wavelet* utilizada;
- Taxa de bits pretendida na codificação (bits/pixel);
- Valor do factor de escala α.

Em traços gerais, o algoritmo SAWVQ/C pode ser descrito da seguinte forma:

É calculada a transformada *wavelet* da imagem e as coordenadas correspondentes aos vectores de coeficientes da transformada são colocadas em uma lista, chamada de **lista dominante**, e todas classificadas como sendo insignificantes, 'IS'. Assim, na primeira vez que esta lista é varrida, todos os vectores estão marcados como 'IS' e por isso todos são comparados com um dado valor de referência de forma a ser enviado o mapa de significâncias. Este mapa de significâncias é agora enviado usando um condicionamento estatístico implementado da seguinte forma: Caso o vector de coeficientes seja significante, ele é marcado na lista dominante como 'SS' e gerado o símbolo correspondente a vector de coeficientes significante. Este símbolo é então codificado dependendo do contexto em que se encontra. Neste caso, tem que ser também enviado o índice do vector de orientação pertencente ao dicionário que mais se aproxima. No caso do vector de coeficientes continuar a ser insignificante, ele é marcado como 'IS' e gerado o símbolo correspondente a esta situação. Este símbolo é também codificado tendo em consideração o contexto em que o vector se encontra.

No entanto, na segunda vez que a lista dominante é varrida, passam a existir vectores de coeficientes 'IS' e 'SS'. Para aqueles que se encontram marcados por 'IS' todo o processo é repetido, sendo agora o valor de referência menor (foi dividido por 2). Os restantes, marcados com 'SS', são refinados de forma a serem enviados os planos de bits propriamente ditos.

O processo termina quando a taxa de bits pretendida for atingida.

4.3.2.3 Implementação do algoritmo

O algoritmo de codificação SAWVQ/C pode ser descrito mais detalhadamente da seguinte forma:

1. A média da imagem é calculada e subtraída.

2. Aplica-se uma transformada *wavelet* com R estágios à imagem com média zero.

3. Cada banda dos coeficientes da *wavelet* é dividida em blocos $M \times N$, de modo

a formar vectores de dimensão MN. Dependendo da banda que está a ser considerada, o varrimento dos blocos é feito de forma diferente. É efectuado um varrimento vertical nas bandas verticais, um horizontal nas bandas horizontais e em *zig-zag* nas bandas diagonais. A ordem de varrimento dos blocos nas diferentes bandas de frequência encontra-se ilustrada na figura 4.3.

4. É encontrado o vector de coeficientes da transformada que tem maior amplitude, $\|\vec{V}\|_{max}$.

5. O valor de referência inicial l_0 é dado por $\alpha\|\vec{V}\|_{max}$, onde α depende do dicionário de vectores de orientação que será utilizado.

6. É criada uma lista de coordenadas dos vectores, chamada de **lista dominante**, que define a ordem segundo a qual os coeficientes são varridos. Esta ordem é tal que os coeficientes das bandas com mais baixa frequência são sempre varridos primeiro do que aqueles que se encontram em bandas com frequência mais alta.

7. Inicialmente a lista dominante é varrida e todos os vectores de coeficientes da transformada são marcados como sendo insignificantes, 'IS'.

8. *Passo dominante e de refinamento*: Este passo deve ser dividido em duas fases, uma correspondente à primeira vez que a lista dominante é varrida e a outra para as restantes vezes. Assim sendo tem-se:

   (a) A primeira vez que a lista é varrida, todos os vectores que nela se encontram estão marcados com sendo insignificantes. Assim, todos são comparados com o valor de referência de forma a ver quais o que se tornaram significantes nesta iteração. Podem surgir duas situações distintas:

      i. Quando o módulo da amplitude do vector é menor que o valor de referência, o vector continua marcado como sendo 'IS' na lista dominante e o vector é reconstruído com o valor zero. É gerado um símbolo '0' que é adicionado à *string*, tendo em conta os parâmetros

usados para o condicionamento estatístico que foram apresentados acima.

ii. Quando o módulo da amplitude do vector é maior que o valor de referência, ele é marcado como sendo 'SS' na lista dominante e é gerado o símbolo '1' que é adicionado à *string* dependendo do contexto em que se encontra. Nesta situação, é também gerado e enviado para a *string* um símbolo 'C_j' que corresponde ao índice do vector de orientação mais próximo, pertencente ao dicionário. O vector é em seguida reconstruído com o vector de orientação mais próximo, escalado pelo valor de referência da iteração actual, l_n.

(b) A partir da segunda vez que a lista dominante é varrida, nela já vão existir vectores marcados como sendo 'IS' e outros como sendo 'SS'. Assim tem-se:

i. Para os vectores que se encontram marcados como 'IS' o processo é efectuado da mesma forma como se fosse a primeira vez que a lista está a ser varrida, sendo a única diferença o facto do valor de referência ter diminuído.

ii. Os vectores que estão marcados como 'SS' são refinados, através do envio de mais um plano de bits, de forma a que o seu erro de reconstrução diminua. Desta forma, é calculada a diferença entre o vector original e o reconstruído até ao momento, utilizando-se o valor de referência l_n e um vector de orientação escolhido do dicionário. Os índices destes novos vectores de orientação são gerados e enviados para a *string*.

A codificação da *string* gerada é, neste caso, um pouco mais complicada. Para enviar o mapa de significâncias do vector podem ser gerados 2 símbolos diferentes, '1' ou '0'. Desta forma, o codificador aritmético adaptativo responsável por fazer a codificação da *string* deve neste momento ser actualizado para o uso de 2 símbolos. Contudo, caso o coeficiente passe a significante é necessário o envio do índice do vector de orientação mais próximo. Assim, nesta altura, o

codificador aritmético adaptativo tem que ser actualizado para o uso de uma quantidade de símbolos igual áquela que constitui o dicionário de vectores de orientação. No caso dos vectores que já se encontram marcados com 'SS' na lista dominante, o codificador aritmético adaptativo tem que ser logo actualizado para o uso de uma quantidade de símbolos igual aquela que constitui o dicionário.

9. O valor de referência l_n é multiplicado pelo factor de escala α, ficando agora com o valor l_{n+1}.

10. o valor n é incrementado e o processo é repetido desde o passo 8, e é interrompido em qualquer ponto quando o tamanho do *bitstream* exceder um limite pré-estabelecido.

O *bitstream* possui um cabeçalho de 11 bytes com informação extra necessária ao algoritmo de descodificação. Ele contém:

- o valor de α (1 byte);
- o número de estágios (1 byte);
- as dimensões da imagem (4 bytes);
- a média da imagem (1 byte);
- o valor de referência inicial (4 bytes).

O codificador e o descodificador têm as suas listas dominantes inicializadas da mesma forma e por isso encontram-se perfeitamente sincronizados, permitindo que o *bitstream* gerado seja descodificado correctamente. É igualmente necessário que, durante o processo de descodificação, seja levado em consideração o contexto em que os vectores foram codificados. Desta forma, tem que existir uma cópia dos contextos do codificador no descodificador.

4.3.2.4 Análise do algoritmo

As características que tornam este algoritmo eficiente são muito idênticas às do EZW, sendo a única diferença a substituição das árvores de zeros pelos contextos. Desta forma, para este algoritmo pode-se citar o seguinte:

- O ganho que se observa com este algoritmo deve-se ao facto do condicionamento estatístico se apresentar mais eficiente do que a predição que é feita com árvores de zeros. [2].
- O facto de os vectores serem codificados de acordo com o módulo da sua amplitude, dando sempre prioridade àqueles que são maiores. Esta característica faz com que o nível de distorção seja sempre o menor possível, pois, quando o processo é interrompido, é garantido que os vectores não codificados são sempre aqueles que apresentam menor módulo.
- É possível interromper o processo em qualquer altura, dependendo da taxa de bits que se pretende para a codificação da imagem.

4.3.3 Resumo das principais características do EZW/C e do SAWVQ/C

Na tabela 4.3 é apresentado um resumo das principais características dos algoritmos EZW/C e SAWVQ/C.

Sobre as diferenças apresentadas é importante ressaltar o número de símbolos que é necessário usar para a representação da imagem para cada um dos algoritmos. Enquanto, que o EZW/C usa 2 símbolos, o SAWVQ/C usa bastante mais, sendo que este número vai depender do número de vectores de orientação que constituem o dicionário, m. Assim sendo, existe uma perda de eficiência clara na parte de codificação por entropia que é consequência do facto do aumento significativo do número de símbolos. No entanto, uma vez que a quantização é efectuada de forma vectorial, ou seja, de cada vez que é codificado o mapa de significâncias de um vector na realidade estão a ser codificados MN coeficientes da transformada, existe

	EZW/C	**SAWVQ/C**
Transformada da imagem	Transformada *wavelet*	Transformada *wavelet*
Método de quantização	Aproximação sucessiva de escalares	Aproximação sucessiva de vectores
Unidade de codificação	Coeficiente	Vector de coeficientes
Codificação mapa de significâncias	Condicionamento estatístico	Condicionamento estatístico
Alfabeto	2 símbolos	2 ou $m+1$ símbolos
Codificação dos símbolos	Codificador aritmético adaptativo	Codificador aritmético adaptativo
Valor de referência	Dividido por 2 a cada iteração	Multiplicado por α a cada iteração $0,5 \leq \alpha < 1$

Tabela 4.3: Principais características da implementação dos algoritmos EZW/C e SAWVQ/C.

uma economia muito grande nos símbolos que é necessário enviar para codificar esse mapa de significâncias (MN vezes menos símbolos).

4.4 Algoritmo SPIHT e SPIHTVQ

Nesta secção será descrito o algoritmo SPIHT e a sua extensão para planos de bits vectoriais, aqui denominado por SPIHTVQ.

Este algoritmo apareceu depois do EZW como sendo uma adaptação a este. A principal diferença diz respeito à forma como é inicializada a codificação dos coeficientes da transformada. Ao contrário do algoritmo EZW que inicialmente considera que todos os coeficientes são significantes e indica quais são insignificantes, o algoritmo SPIHT considera que inicialmente todos são insignificantes e indica quais se tornam significantes.

4.4.1 Algoritmo SPIHT [15]

4.4.1.1 Estrutura do algoritmo

A estrutura do algoritmo é a seguinte:

- **Transformada da imagem**: transformada *wavelet*;
- **Método de quantização**: aproximação sucessiva de escalares;
- **Unidade de codificação**: coeficiente;
- **Codificação do mapa de significâncias**: estrutura de listas;
- **Alfabeto**: 2 símbolos;
- **Codificação dos símbolos**: codificador aritmético adaptativo.

4.4.1.2 Linhas gerais do algoritmo

Os parâmetros de entrada deste algoritmo são os seguintes:

- Nome da imagem a codificar;
- Número de níveis de decomposição para a *wavelet* utilizada;
- Taxa de bits pretendida na codificação (bits/pixel).

Para melhor compreensão do algoritmo são definidos em seguida uma série de conjuntos de coordenadas de coeficientes que são usados por este algoritmo.

- $\mathcal{O}(i,j)$: conjunto das coordenadas de todos os coeficientes que são descendentes directos do coeficiente de coordenadas (i,j), ou seja, coordenadas de todos os coeficientes filhos;
- $\mathcal{D}(i,j)$: conjunto das coordenadas de todos os coeficientes que são descendentes do coeficiente de coordenadas (i,j);

- $\mathcal{H}$: conjunto das coordenadas de todos os coeficientes que são raízes das árvores de orientação espacial, ou seja, as coordenadas de todos os coeficientes que se encontram na banda de mais baixa frequência.

- $\mathcal{L}(i,j)$: conjunto das coordenadas de todos os coeficientes que são descendentes, mas não são descendentes directos do coeficiente de coordenadas (i,j), ou seja,

$$\mathcal{L}(i,j) = \mathcal{D}(i,j) - \mathcal{O}(i,j). \tag{4.1}$$

Este algoritmo usa também três listas, o que permite implementar de uma forma muito eficiente o conceito de árvore de zeros. As listas são as seguintes:

- **Lista de Pixeis Insignificantes**, aqui denominada por LIP;

- **Lista de Pixeis Significantes**, aqui denominada por LSP;

- **Lista de Conjuntos Insignificantes**, aqui denominada por LIS;

É importante referir que cada entrada nestas listas é identificada pela coordenada (i,j), que no caso das listas LIP e da LIS representam coeficientes individuais e na lista LIS tanto podem representar o conjunto de todos os descendentes de um dado coeficiente (conjunto $\mathcal{D}(i,j)$), como podem representar todos os coeficientes que não são descendentes directos (conjunto $\mathcal{L}(i,j)$). Para diferenciar estes dois casos, diz-se que uma entrada da LIS é do tipo A se ela representa $\mathcal{D}(i,j)$ e do tipo B caso represente $\mathcal{L}(i,j)$.

Em termos gerais, a codificação de uma imagem estática, usando este algoritmo, é implementada da seguinte forma:

É calculada a transformada *wavelet* da imagem. São inicializadas as três listas. A lista LSP é inicializada como zero, a lista LIP guarda as coordenadas que pertencem ao conjunto $\mathcal{H}$ e a lista LIS é inicializada com as coordenadas pertencentes ao conjunto $\mathcal{H}$ mas que têm descendentes nas outras bandas. Para uma melhor compreeder a inicialização destas listas é apresentada a figura 4.4. Todos os coeficientes que se encontram na banda de mais baixa frequência são colocados na LIP

e todos aqueles que se encontram nesta mesma banda, mas não estão marcados de preto são colocados na LIS.

No passo dominante a lista LIP é varrida e se um coeficiente se torna significante as suas coordenadas são movidas para a lista LSP. Em seguida é percorrida a lista LIS e se um dado coeficiente tiver um seu descendente significante todos eles têm que ser testados e caso sejam significantes colocados na LSP.

No passo subordinado, toda a lista LSP é percorrida e é enviado mais um plano de bits para os coeficientes que nela se encontram.

O processo repete-se até ser atingida a taxa de bits pretendida ou os planos de bits serem completamente codificados.

4.4.1.3 Implementação do algoritmo

O algoritmo de codificação SPIHT pode ser descrito mais detalhadamente da seguinte forma:

1. A média da imagem é calculada e subtraída.

2. Aplica-se uma transformada *wavelet* com R estágios à imagem com média zero.

3. É calculado o maior plano de bits que é necessário para representar o maior coeficiente da transformada da imagem, ou seja:

$$n = \lfloor \log_2(\max_{(i,j)} \|C_{i,j}\|) \rfloor \tag{4.2}$$

   onde:

   - n representa o maior plano de bits;
   - (i, j) as coordenadas do coeficiente;
   - $C_{i,j}$ a amplitude do coeficiente para as coordenadas (i, j).

4. São criadas e inicializadas as seguintes listas: a **lista de pixeis significantes**, LSP, como sendo uma lista vazia, a **lista de pixeis insignificantes**, LIP, onde são colocadas as coordenadas referentes a todos os coeficientes que

se encontram na banda de mais baixa frequência, ou seja, coordenadas pertencentes ao conjunto $\mathcal{H}$ (todos os coeficientes que se encontram na banda de mais baixa frequência da figura 4.4), e finalmente a **lista de conjuntos insignificantes** onde são colocadas as coordenadas de todos os coeficientes que estão na banda de mais baixa frequência, mas que têm descendentes, ou seja, coordenadas pertencentes ao conjunto $\mathcal{H}$ que têm descendentes nas bandas de frequência superior (todos os coeficientes que se encontram na banda de mais baixa frequência da figura 4.4, mas que não estão marcados de preto). No caso da lista LIS é também preciso referir se as coordenadas (i, j) representam todos os descendentes de um dado coeficiente (conjunto $\mathcal{D}(i, j)$) ou se representam apenas os descendentes que não são directos (conjunto $\mathcal{L}(i, j)$), sendo classificadas como sendo do tipo A ou B, respectivamente.

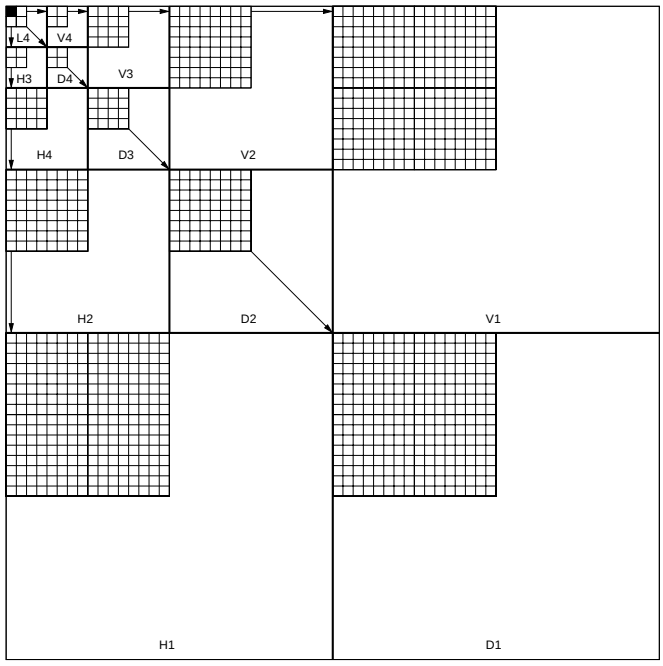


Figura 4.4: Conjunto de coeficientes do SPIHT para uma imagem com uma transformada *wavelet* de 4 estágios.

5. *Passo dominante*: Este passo pode ser dividido em duas partes:

   (a) É varrida toda a lista LIP e se o valor do coeficiente com coordenadas

(i, j) é menor que 2^n, ele vai ser reconstruído com o valor zero e é gerado e enviado para a *string* o símbolo '0', que corresponde a coeficiente insignificante. Caso contrário é gerado e enviado para a *string* o símbolo '1', correspondente a coeficiente significante, e o seu valor reconstruído com $\pm 2^n$, de acordo com o seu sinal. O sinal também é enviado para a *string*. Caso o coeficiente se tenha tornado significante, as suas coordenadas (i, j) devem ser removidas da LIP e colocadas na LSP.

(b) É varrida toda a lista LIS e, para o conjunto de coordenadas (i, j) pode-se ter o seguinte:

i. Caso as coordenadas (i, j) sejam do tipo A, ou seja, se representarem o conjunto de todos os descendentes do coeficiente que tem coordenadas (i, j), é enviado para a *string* o símbolo correspondente à significância de cada um dos coeficientes descendentes, ou seja, a significância de todos os coeficientes que pertencem ao conjunto $\mathcal{D}(i, j)$. Como no caso anterior, o '0' significa que o coeficiente é insignificante e '1' que ele é significante.

Se entre os vectores enviados existe pelo menos um descendente, pertencente ao conjunto $\mathcal{D}(i, j)$, que é significante é feito o seguinte:

- é enviada para a *string* a significância de todos os coeficientes que são filhos daquele que tem coordenadas (i, j), ou seja, de todas as coordenadas (k, l) que pertencem ao conjunto $\mathcal{O}(i, j)$. Como nos casos anteriores, '0' corresponde a coeficiente insignificante e '1' a coeficiente significante;
- Caso as coordenadas (k, l) correspondam a um coeficiente significante, estas devem ser adicionadas à lista LSP e deve ser enviado para a *string* o sinal do coeficiente de coordenadas (k, l). O valor deste coeficiente deve ser reconstruído com o valor $\pm 2^n$, dependendo do seu sinal;
- Caso as coordenadas (k, l) correspondam a um coeficiente insignificante, estas devem ser colocadas no final da lista LIP de forma

a que voltem ainda a ser lidas durante esta iteração.

Se algum dos coeficientes que não são descendentes directos do coeficiente que tem coordenadas (i, j), ou seja pertençam ao conjunto $\mathcal{L}(i, j)$, for significante então, as coordenadas (i, j) são colocadas no final da LIS e passam a ser do tipo B, ou seja significa que existe pelo menos um coeficiente que não é descendente directo do coeficiente de coordenadas (i, j) que é significante. Caso contrário, as coordenadas (i, j) são removidas da LIS.

ii. Caso as coordenadas (i, j) sejam do tipo B, ou seja, se representarem o conjunto de todos os descendentes não directos do coeficiente que tem coordenadas (i, j), é enviado para a *string* o símbolo correspondente à significância de cada um destes coeficientes, ou seja, a significância de todos os coeficientes que pertencem ao conjunto $\mathcal{L}(i, j)$. Como no caso anterior, o '0' significa que o coeficiente é insignificante e '1' que ele é significante.

Se entre os vectores enviados existe pelo menos um descendente, pertencente ao conjunto $\mathcal{L}(i, j)$, que é significante é feito o seguinte:

- todos os coeficientes que são filhos daquele que tem coordenadas (i, j), ou seja, todas as coordenadas (k, l) que pertençam ao conjunto $\mathcal{O}(i, j)$, são adicionadas no final da LIS e passam a ser do tipo A;
- as coordenadas (i, j) são removidas da LIS.

Como pode ser visto, para codificar o mapa de significâncias é suficiente utilizar os símbolos '1' e '0'. Desta forma, a *string* de símbolos gerada vai ser codificada utilizando o mesmo codificador aritmético apresentado em [17], que tem que ser actualizado a cada iteração para o uso de 2 símbolos.

6. *Passo de refinamento*: Para cada uma das coordenadas (i, j) que se encontram na LSP, menos aquelas que foram incluídas durante esta iteração, é enviado mais um plano de bit n de forma a que o erro de reconstrução seja diminuído.

Assim, a cada coeficiente que corresponde a estas coordenadas, é adicionado o valor $+2^n$ ou então -2^n, dependendo do sinal. Então, no primeiro caso, é gerado um símbolo '+' e enviado para a *string*, caso contrário é gerado e enviado para a *string* o símbolo '−'.

Tal como para a codificação do mapa de significâncias, para codificar os planos de bits propriamente ditos é suficiente utilizar os símbolos '1' e '0'. Assim, basta que o codificador aritmético seja iniciado no princípio de cada iteração.

7. n é decrementado.

8. O processo é repetido desde o passo 5, e é interrompido em qualquer ponto quando o tamanho do *bitstream* exceder um limite pré-estabelecido.

O *bitstream* possui um cabeçalho de 7 bytes com informação extra necessária ao algoritmo de descodificação. Ele contém:

- o número de estágios (1 byte);
- as dimensões da imagem (4 bytes);
- a média da imagem (1 byte);
- maior plano de bits necessário para representar os coeficientes (1 byte).

As diversas listas são inicializadas da mesma forma para o codificador e para o descodificador, fazendo com que o descodificador esteja sincronizado com o codificador em cada instante, permitindo que o *bitstream* seja descodificado correctamente.

Em [10] é mostrado que para reduzir o erro médio de reconstrução da imagem é necessário que, no final do processo de descodificação, seja somado o correspondente a $2^{(i-1)}$, (i corresponde ao último plano de bits que foi comparado com o coeficiente) a cada coeficiente quantizado. Assim, no caso do algoritmo SPIHT faz-se:

1. Caso o processo tenha sido interrompido quando estavam a ser refinados os coeficientes da LSP, é feito o seguinte:

(a) Caso já tenha sido somado durante este passo o valor $\pm 2^n$ a um dado coeficiente, é agora somado o valor $\pm 2^{(n-1)}$, dependendo do seu sinal.

(b) Aos restantes coeficientes que se encontram na lista LSP, menos aqueles que nela foram colocados durante o passo dominate anterior, é somado o valor $\pm 2^n$, dependendo do seu sinal.

2. Caso o processo tenha sido interrompido durante o passo dominante, é feito o seguinte:

(a) A todos os coeficientes que se tornaram significantes durante este passo é somado o valor $\pm 2^{(n-1)}$, dependendo do seu sinal.

(b) A todos os coeficientes que antes do início deste passo já se encontravam na LSP é somado o valor $\pm 2^n$, dependendo do sinal.

4.4.1.4 Análise do algoritmo

O algoritmo descrito acima possui algumas características importante que o tornam bastante eficiente. A saber:

- O uso de uma estrutura de listas permite que a similaridade entre as diferentes bandas de frequência da transformada *wavelet* seja aproveitada de uma forma muito eficiente;

- O facto de inicialmente todos os coeficientes serem considerados insignificantes aumenta em muito a eficiência do algoritmo, uma vez que eles vão ser considerados sempre insignificantes até prova em contrário;

- Durante o passo dominante vão sendo colocadas novas coordenadas na lista LIS que serão avaliadas mesmo antes do passo dominante terminar. Ou seja, quando é dito que "é varrida toda a LIS" está-se a falar também nas coordenadas que estão a ser colocadas no final desta lista durante o passo dominante;

- O uso de um alfabeto muito pequeno para codificar uma imagem permite que o codificador aritmético se adapte muito rapidamente a qualquer mudança nas estatísticas do símbolos;

- O facto de os coeficientes serem codificados de acordo com o seu valor absoluto, dando sempre prioridade aqueles que são maiores. Esta característica faz com que o nível de distorção seja sempre o menor possível pois, quando o processo é interrompido é garantido que os coeficientes não codificados são sempre aqueles que apresentam menor valor absoluto.
- É possível interromper o processo em qualquer altura, dependendo da taxa de bits que se pretende para a codificação da imagem.

4.4.2 Algoritmo SPIHTVQ

4.4.2.1 Estrutura do algoritmo

Este algoritmo é uma adaptação do SPIHT e foi desenvolvido no âmbito deste trabalho.

São duas as principais mudanças que existem entre o SPIHT e o SPIHTVQ. A primeira é que passam a ser processados vectores de coeficientes em vez de cada coeficiente individualmente. A segunda modificação reside no facto do valor de referência deixar de ser dado por 2^n e passar a ser o valor do coeficiente máximo multiplicado a cada iteração n, onde n representa o plano de bits actual, por um factor de escala α $(0,5 \leq \alpha < 1)$, ou seja, $\alpha^i \|\vec{V}\|_{max}$ onde i representa o número de iterações que já foram efectuadas.

A estrutura do algoritmo é a seguinte:

- **Transformada da imagem**: transformada *wavelet*;
- **Método de quantização**: aproximação sucessiva de vectores;
- **Unidade de codificação**: vector de coeficientes;
- **Codificação do mapa de significâncias**: estrutura complexa de listas;
- **Alfabeto**: 2 símbolos ou $m + 1$ símbolos, onde m corresponde ao número de vectores de orientação que constituem o dicionário;
- **Codificação dos símbolos**: codificador aritmético adaptativo.

4.4.2.2 Linhas gerais do algoritmo

Os parâmetros de entrada deste algoritmo são os seguintes:

- Nome da imagem a codificar;
- Número de níveis de decomposição para a *wavelet* utilizada;
- Taxa de bits pretendida na codificação (bits/pixel);
- Factor de escala α.

Para melhor compreensão do algoritmo são definidos em seguida uma série de conjuntos de coordenadas de vectores de coeficientes que são usados por este algoritmo.

- $\mathcal{O}_v(i,j)$: conjunto das coordenadas de todos os vectores de coeficientes que são descendentes directos do vector de coordenadas (i,j), ou seja, coordenadas de todos os vectores filhos;
- $\mathcal{D}_v(i,j)$: conjunto das coordenadas de todos os vectores de coeficientes que são descendentes do vector de coordenadas (i,j);
- $\mathcal{H}_v$: conjunto das coordenadas de todos os vectores de coeficientes que são raízes das árvores de orientação espacial, ou seja, as coordenadas de todos os vectores que se encontram na banda de mais baixa frequência.
- $\mathcal{L}_v(i,j)$: conjunto das coordenadas de todos os vectores de coeficientes que não são descendentes directos do vector de coordenadas (i,j), ou seja,

$$\mathcal{L}_v(i,j) = \mathcal{D}_v(i,j) - \mathcal{O}_v(i,j). \tag{4.3}$$

Este algoritmo usa também três listas o que permite implementar de uma forma muito eficiente o conceito de árvore de zeros. As listas são as seguintes:

- **Lista de Vectores Insignificantes**, aqui denominada por LIV;
- **Lista de Vectores Significantes**, aqui denominada por LSV;

- **Lista de Conjuntos Insignificantes**, aqui denominada por LIS;

É importante referir que cada entrada nestas listas é identificada pela coordenada (i, j), que no caso das listas LIV e da LIS representam vectores de coeficientes individuais e na lista LIS tanto podem representar o conjunto de todos os descendentes de um dado vector de coeficientes (conjunto $\mathcal{D}_v(i, j)$), como podem representar todos os vectores de coeficientes que não são descendentes directos (conjunto $\mathcal{L}_v(i, j)$). Para diferenciar estes dois casos, diz-se que uma entrada da LIS é do tipo A se ela representa $\mathcal{D}_v(i, j)$ e do tipo B caso represente $\mathcal{L}_v(i, j)$.

Em termos gerais, a codificação de uma imagem estática, usando este algoritmo, é implementada da seguinte forma:

É calculada a transformada *wavelet* da imagem. São inicializadas as três listas. Este algoritmo tem como principal característica o facto de inicialmente considerar que todos os vectores de coeficientes da imagem são insignificantes e estes permanecem assim até que seja demonstrado o contrário. Assim, a lista LSV é inicializada como zero, a lista LIV guarda as coordenadas que pertencem ao conjunto $\mathcal{H}_v$ e a lista LIS é inicializada com as coordenadas pertencentes ao conjunto $\mathcal{H}_v$ mas que têm descendentes nas outras bandas. Para melhor compreender a inicialização destas listas é apresentada a figura 4.5. Todos os vectores de coeficientes que se encontram na banda de mais baixa frequência são colocados na LIV e todos aqueles que se encontram nesta mesma banda, mas não estão marcados de preto são colocados na LIS.

No passo dominante a lista LIV é varrida e se um vector se torna significante as suas coordenadas são movidas para a lista LSV. Em seguida é percorrida a lista LIS e caso, para o dado vector, exista um de seus descendentes que seja significante, todos eles têm que ser testados, e caso sejam significantes colocados na LSV.

No passo subordinado, toda a lista LSV é percorrida e é enviado mais um plano de bits para os vectores que nela se encontram.

O processo repete-se até ser atingida a taxa de bits pretendida ou os planos de bits serem completamente codificados.

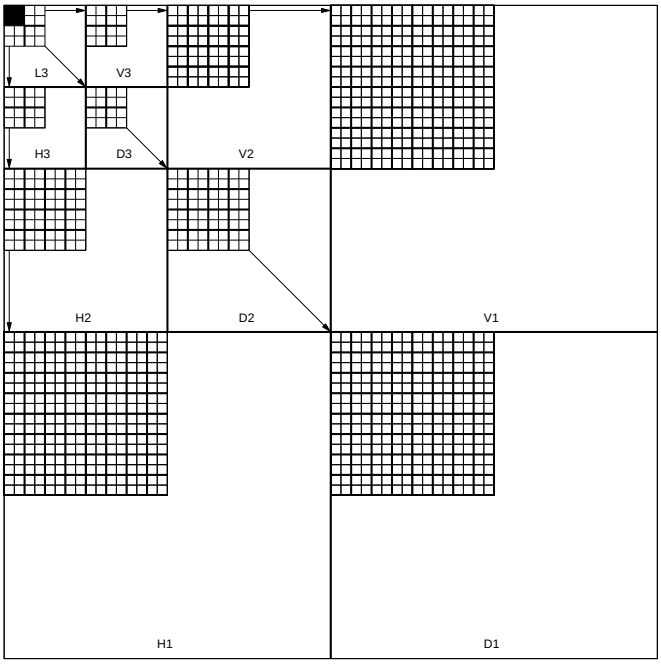


Figura 4.5: Conjunto de vectores de coeficientes do SPIHTVQ para uma imagem com uma transformada *wavelet* de 3 estágios.

4.4.2.3 Implementação do algoritmo

O algoritmo de codificação SPIHTVQ pode ser descrito mais detalhadamente da seguinte forma:

1. A média da imagem é calculada e subtraída.

2. Aplica-se uma transformada *wavelet* com R estágios à imagem com média zero.

3. É encontrado o vector de coeficientes da transformada que tem maior amplitude $\|\vec{V}\|_{max}$.

4. O valor de referência inicial, l_0, é dado por $\alpha\|\vec{V}\|_{max}$, onde α depende do dicionário de vectores de orientação que será utilizado.

5. São criadas e inicializadas as seguintes listas: a **lista de vectores significantes**, LSV, como sendo uma lista vazia, a **lista de vectores insignificantes**, LIV, onde são colocadas as coordenadas referentes a todos os vectores que se

encontram na banda de mais baixa frequência, ou seja, coordenadas pertencentes ao conjunto $\mathcal{H}_v$ (todos os vectores de coeficientes que se encontram na banda de mais baixa frequência da figura 4.5), e finalmente a **lista de conjuntos significantes** onde são colocadas as coordenadas de todos os vectores que estão na banda de mais baixa frequência mas que têm descendentes, ou seja, coordenadas pertencentes ao conjunto $\mathcal{H}_v$ que têm descendentes nas bandas de frequência superior (todos os vectores de coeficientes que se encontram na banda de mais baixa frequência da figura 4.5, mas que não estão marcados de preto). No caso da lista LIS é também necessário referir se as coordenadas (i, j) representam todos os descendentes de um dado vector (conjunto $\mathcal{D}_v(i, j)$) ou se representam apenas os descendentes que não são directos (conjunto $\mathcal{L}_v(i, j)$), sendo classificadas como sendo do tipo A ou B, respectivamente.

6. *Passo dominante*: Este passo pode ser dividido em duas partes:

   (a) É varrida toda a lista LIV e se o módulo da amplitude do vector com coordenadas (i, j) é menor que o valor de referência l_n, ele vai ser reconstruído com o valor zero e é gerado e enviado para a *string* o símbolo '0', que corresponde a coeficiente insignificante. Caso contrário é gerado e enviado para a *string* o símbolo '1', correspondente a coeficiente significante. Neste caso é também enviado para a *string* o símbolo 'C_j' que corresponde ao índice do vector de orientação mais próximo pertencente ao dicionário.

   Caso o vector se tenha tornado significante, as suas coordenadas (i, j) devem ser removidas da LIV e colocadas na LSV.

   (b) É varrida toda a lista LIS e, para conjunto de coordenadas (i, j) pode-se ter o seguinte:

      i. Caso as coordenadas (i, j) sejam do tipo A, ou seja, se representarem o conjunto de todos os descendentes do vector que tem coordenadas (i, j), é enviado para a *string* o símbolo correspondente à significância de cada um dos vectores descendentes, ou seja, a significância de

todos os vectores que pertencem ao conjunto $\mathcal{D}_v(i,j)$. Como no caso anterior, o '0' significa que o vector é insignificante e '1' que ele é significante.

Se entre os vectores enviados existe pelo menos um descendente, pertencente ao conjunto $\mathcal{D}_v(i,j)$, que é significante é feito o seguinte:

- é enviada para a *string* a significância de todos os vectores que são filhos daquele que tem coordenadas (i,j), ou seja, de todas as coordenadas (k,l) que pertencem ao conjunto $\mathcal{O}_v(i,j)$. Como nos casos anteriores, '0' corresponde a vector insignificante e '1' a vector significante;
- Caso as coordenadas (k,l) correspondam a um vector significante, estas devem ser adicionadas à lista LSV e deve ser enviado para a *string* o símbolo 'C_j' correspondente ao vector de orientação mais próximo no dicionário.
- Caso as coordenadas (k,l) correspondam a um vector insignificante, estas devem ser colocadas no final da lista LIS de forma a que voltem ainda a ser lidas durante esta iteração.

Se algum dos vectores que não são descendentes directos do vector que tem coordenadas (i,j), ou seja pertençam ao conjunto $\mathcal{L}_v(i,j)$, for significante então, as coordenadas (i,j) são colocadas no final da LIS e passam a ser do tipo B, ou seja, significa que existe pelo menos um vector que não é descendente directo daquele de coordenadas (i,j) que é significante. Caso contrário, as coordenadas (i,j) são removidas da LIS.

ii. Caso as coordenadas (i,j) sejam do tipo B, ou seja, se representarem o conjunto de todos os descendentes não directos do vector que tem coordenadas (i,j), é enviado para a *string* o símbolo correspondente à significância de cada um destes vectores, ou seja, a significância de todos aqueles que pertencem ao conjunto $\mathcal{L}_v(i,j)$. Como no caso anterior, o '0' significa que o vector é insignificante e '1' que ele é

significante.

Se entre os vectores enviados existe pelo menos um descendente, pertencente ao conjunto $\mathcal{L}_v(i,j)$, que é significante é feito o seguinte:

- todos os vectores que são filhos daquele que tem coordenadas (i,j), ou seja, todas as coordenadas (k,l) que pertençam ao conjunto $\mathcal{O}_v(i,j)$, são adicionadas no final da LIS e passam a ser do tipo A;
- as coordenadas (i,j) são removidas da LIS.

A codificação da *string* gerada é, neste caso, um pouco mais complicada. Para enviar o mapa de significâncias do vector podem ser gerados 2 símbolos diferentes '1' ou '0'. Desta forma, o codificador aritmético adaptativo responsável por fazer a codificação da *string* deve neste momento ser actualizado para o uso de 2 símbolos. Contudo, caso o coeficiente passe a significante é necessário o envio do índice do vector de orientação mais próximo. Assim, nesta altura, o codificador aritmético adaptativo tem que ser actualizado para o uso de uma quantidade de símbolos igual áquela que constitui o dicionário de vectores de orientação.

7. *Passo de refinamento*: Para cada uma das coordenadas (i,j) que se encontram na LSV, menos aquelas que foram incluídas durante esta iteração, é enviado mais um plano de bits n de forma a que o erro de reconstrução seja diminuído. Assim, neste processo de refinamento, é codificada a diferença entre o vector original e o vector reconstruído até ao momento, utilizando-se o valor de referência actual, l_n, e um vector de orientação escolhido do dicionário. Os índices destes novos vectores de orientação são codificados em um *bitstream* usando um codificador aritmético, que no início deste passo é inicializado para permitir o número de vectores que constitui o dicionário.

8. O valor de referência l_n é multiplicado pelo factor de escala α.

9. O valor n é icrementado e o processo é repetido desde o passo 6, e é interrompido em qualquer ponto quando o tamanho do *bitstream* exceder um limite

pré-estabelecido.

O *bitstream* possui um cabeçalho de 8 bytes com informação extra necessária ao algoritmo de descodificação. Ele contém:

- o valor de α (1 byte);
- o número de estágios (1 byte);
- as dimensões da imagem (4 bytes);
- a média da imagem (1 byte);
- maior plano de bits necessário para representar os coeficientes (1 byte).

As diversas listas são inicializadas da mesma forma para o codificador e para o descodificador, fazendo com que o descodificador esteja sincronizado com o codificador em cada instante, permitindo que o *bitstream* seja descodificado correctamente.

4.4.2.4 Análise do algoritmo

O algoritmo descrito acima possui as mesmas características que aquelas que foram apresentadas para a sua versão implementada com planos de bits escalares. Pode-se citar:

- O uso de uma estrutura de listas permite que a similaridade entre as diferentes bandas de frequência da transformada *wavelet* seja aproveitada de uma forma muito eficiente;
- O facto de inicialmente todos os vectores de coeficientes serem considerados insignificantes aumenta em muito a eficiência do algoritmo, uma vez que eles vão ser considerados sempre insignificantes até prova em contrário;
- Durante o passo dominante vão sendo colocadas novas coordenadas na lista LIS que serão avaliadas mesmo antes do passo dominante terminar. Ou seja, quando é dito que "é varrida toda a LIS" está-se a falar também nas coordenadas que estão a ser colocadas no final desta lista durante o passo dominante;

- O facto de os vectores coeficientes serem codificados de acordo com o módulo da sua amplitude, dando sempre prioridade àqueles que são maiores. Esta característica faz com que o nível de distorção seja sempre o menor possível pois, quando o processo é interrompido é garantido que os vectores não codificados são sempre aqueles que apresentam menor módulo de amplitude.

- É possível interromper o processo em qualquer altura, dependendo da taxa de bits que se pretende para a codificação da imagem.

4.4.3 Resumo das principais características do SPIHT e do SPIHTVQ

Na tabela 4.4 é apresentado um resumo das principais características dos algoritmos SPIHT e SPIHTVQ.

	SPIHT	**SPIHTVQ**
Transformada da imagem	Transformada *wavelet*	Transformada *wavelet*
Método de quantização	Aproximação sucessiva de escalares	Aproximação sucessiva de vectores
Unidade de codificação	Coeficiente	Vector de coeficientes
Codificação mapa de significâncias	Estrutura de listas	Estrutura de listas
Alfabeto	2 símbolos	2 ou $m+1$ símbolos
Codificação dos símbolos	Codificador aritmético adaptativo	Codificador aritmético adaptativo
Valor de referência	Dividido por 2 a cada iteração	Multiplicado por α a cada iteração, $0,5 \leq \alpha < 1$

Tabela 4.4: Principais características da implementação dos algoritmos SPIHT e SPIHTVQ.

Sobre as diferenças apresentadas é importante ressaltar o número de símbolos que é necessário usar para a representação da imagem para cada um dos algoritmos.

Enquanto que o SPIHT usa 2 símbolos, o SPIHTVQ usa bastante mais, sendo que este número vai depender do número de vectores de orientação que constituem o dicionário, m. Assim sendo, existe uma perda de eficiência clara na parte de codificação por entropia que é consequência do facto do aumento significativo do número de símbolos. No entanto, uma vez que a quantização é efectuada de forma vectorial, ou seja, de cada vez que é codificado o mapa de significâncias de um vector na realidade estão a ser codificados MN coeficientes da transformada, existe uma economia muito grande nos símbolos que é necessário enviar para codificar esse mapa de significâncias (MN vezes menos símbolos).

4.5 Algoritmo MGE e MGEVQ

Nesta secção são descritos o algoritmo MGE e a sua extensão para planos de bits vectoriais, aqui denominado por MGEVQ.

Neste algoritmo, o paradigma das árvores de zeros é substituído por uma decomposição em *quadtree*, para fazer a codificação dos coeficientes da transformada de uma imagem em planos de bits. A principal diferença entre este algoritmo e os métodos baseados em árvores de zero está na forma como este localiza os coeficientes que são significantes para um dado valor de referência. O MGE usa uma "grelha" que se vai dividindo sobre o domínio da transformada da imagem, e vai encontrando, de uma forma muito eficiente, os coeficientes que são significantes para a codificação.

Para melhor entender como é esta decomposição em *quadtree*, um exemplo é apresentado na figura 4.6. Aí considera-se uma imagem com apenas um coeficiente significante e é apresentada a forma como a imagem é dividida até esse coeficiente ser encontrado.

Os resultados apresentados no capítulo 5 mostram que este algoritmo apresenta um desempenho comparável ao apresentado pelo algoritmo SPIHT.

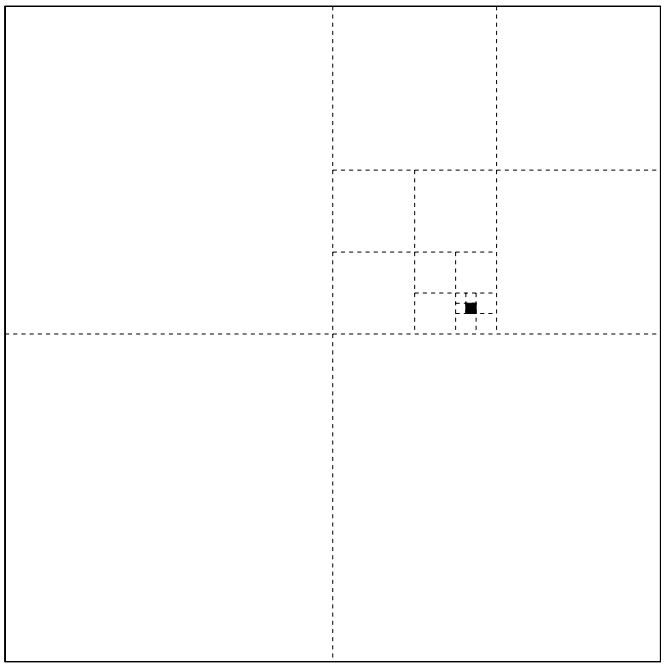


Figura 4.6: Exemplo de uma decomposição em *quadtree*.

4.5.1 Algoritmo MGE [14]

4.5.1.1 Estrutura do algoritmo

A estrutura do algoritmo MGE é a seguinte:

- **Transformada da imagem**: transformada *wavelet*;
- **Método de quantização**: aproximação sucessiva de escalares;
- **Unidade de codificação**: coeficiente;
- **Codificação do mapa de significâncias**: decomposição em *quadtree*;
- **Alfabeto**: 2 símbolos;
- **Codificação dos símbolos**: codificador aritmético adaptativo.

4.5.1.2 Linhas gerais do algoritmo

Os parâmetros de entrada deste algoritmo são os seguintes:

- Nome da imagem a codificar;
- Número de níveis de decomposição para a *wavelet* utilizada;
- Taxa de bits pretendida na codificação (bits/pixel);
- Nível máximo de decomposição em *quadtree*.

Este algoritmo usa duas listas que são de grande importância durante o passo de refinamento:

- **Lista de Coeficientes Significantes**, aqui denominada por SCL.
- **Lista de Coeficientes Insignificantes**, aqui denominada por ICL.

Em termos gerais, a codificação de uma imagem estática, usando este algoritmo, é implementada da seguinte forma:

É feita a transformada *wavelet* da imagem e obtidos os novos coeficientes. As listas ICL e SCL são inicializadas como estando vazias.

Tal como no caso do EZW, a codificação também é feita em dois passos:

No passo dominante, a região a codificar é testada para ver se tem algum coeficiente significante e caso haja, a região é dividida em 4 subregiões iguais, caso contrário deixa de ser dividida. Esta divisão pára quando é atingido um nível máximo de decomposição pré-definido, ou seja, quando é atingido um determinado tamanho para a região. Dependendo deste tamanho pode-se ter regiões que contêm apenas um coeficiente ou então que contêm mais. Assim, quando é definido que a região tem tamanho mínimo maior do que um coeficiente pode-se ter as seguintes situações: apenas um dos coeficientes que constituem a região é significante ou mais do que um o são. Assim, na lista ICL são colocadas todas as coordenadas dos coeficientes que se encontram na região, mas que nesta altura do processo de codificação ainda continuam a ser insignificantes e é apenas enviado o mapa de significância, e para a SCL são colocadas as coordenadas correspondentes aos coeficientes que nesta altura já passaram a significantes, sendo que neste caso é enviado o plano de bits actual.

Estas listas são varridas durante o passo de refinamento, mas apenas são testados aqueles coeficientes que já se encontravam na lista antes do início do passo dominante. Para os coeficientes da SCL é enviado mais um plano de bits, enquanto que os da ICL são testados para ver se se tornaram significantes ou não. Caso já sejam significantes é enviado o plano de bits actual, caso contrário é enviado mais um símbolo para o mapa de significância.

O processo repete-se até ser atingida a taxa de bits pretendida ou os planos de bits serem completamente codificados.

É importante referir que na implementação deste algoritmo é sempre guardado e actualizado, a cada plano de bits que vai sendo enviado, um espelho da decomposição em *quadtree*, de forma a que não seja enviada para a *string* informação redundante acerca da significância das regiões. Este princípio tem como base que, se uma dada região é significante para um determinado plano de bit l_n continuará a sê-lo para o plano de bits seguinte, l_{n-1}.

4.5.1.3 Implementação do algoritmo

O algoritmo de codificação MGE pode ser descrito mais detalhadamente da seguinte forma:

1. A média da imagem é calculada e subtraída.
2. Aplica-se uma transformada *wavelet* com R estágios à imagem com média zero.
3. É calculado o maior plano de bits que é necessário para representar o maior coeficiente da transformada da imagem, ou seja:

$$n = \lfloor \log_2(\max_{(i,j)} \|C_{i,j}\|) \rfloor \tag{4.4}$$

onde:

- n representa o maior plano de bits;
- (i, j) as coordenadas do coeficiente;
- $\|C_{i,j}\|$ a amplitude do coeficiente para as coordenadas (i, j).

4. São inicializadas como vazias duas listas de coeficientes, chamadas de **lista de coeficientes significantes** (SCL) e **lista de coeficientes insignificantes** (ICL).

5. A região a codificar (G) tem o tamanho da imagem original e é marcada como sendo uma "região zero" ($T = 0$).

6. *Passo dominante*: Este passo pode ser dividido da seguinte forma:

   (a) Para cada "região zero" ($T = 0$), é enviado para a *string* um símbolo que indica se essa região é significante ou não, ou seja, é enviado:

   $$T^n_{(i,j)\in(G)}(C_{(i,j)}) \tag{4.5}$$

   onde:

   • T é definido como '1' ("região um") se o valor absoluto de pelo menos um coeficiente que pertence à região é maior que 2^n, caso contrário T é definido como sendo zero;
   • n é o plano de bits que está a ser codificado;
   • G é a região que está a ser analisada;
   • $C_{(i,j)}$ diz respeito ao coeficiente de coordenadas (i, j) que pertence à região G.

   Pode-se então ter uma das seguintes situações:

   i. Caso se trate de uma "região um" ($T = 1$) podem acontecer duas situações distintas:
      - A região ainda não atingiu o tamanho mínimo pré-definido. Neste caso ela é dividida em 4 novas regiões do mesmo tamanho e cada uma delas marcada como sendo "região zero" ($T = 0$). O processo continua no ponto 6a.
      - O tamanho da região é igual ao mínimo pré-definido. Neste caso podem acontecer duas situações:
        – Caso o tamanho da região seja 1, é apenas enviado para a *string* o símbolo correspondente ao sinal do coeficiente, uma

vez que o símbolo que indica a significância do coeficiente já foi enviado como sendo aquele que indica a significância da região. O coeficiente é reconstruído com o valor $\pm 2^n$, dependendo do seu sinal. As coordenadas do coeficiente são colocadas na lista SCL.

- Caso este tamanho seja maior que 1, na região pode haver coeficientes significantes e insignificantes (pelo menos um tem que ser significante). Assim, é enviado para a *string* o símbolo que indica a significância de cada um dos coeficientes que compõem a região. Caso o coeficiente seja significante ele é reconstruído com o valor $\pm 2^n$, dependendo do seu sinal. O símbolo correspondente ao sinal do coeficiente também é enviado para a *string*. Por fim, as coordenadas dos coeficientes que foram reconhecidos como sendo insignificantes são colocadas na ICL e as dos significantes na SCL.

ii. Caso se trate de uma "região zero" ($T = 0$) não é feito mais nada e passa-se a analizar a região seguinte.

Como pode ser visto, para codificar o mapa de significâncias é suficiente utilizar os símbolos '1' e '0'. Desta forma, a *string* de símbolos gerada vai ser codificada utilizando o mesmo codificador aritmético apresentado em [17], que tem que ser actualizado a cada iteração para o uso de 2 símbolos.

7. *Passo de refinamento* Este passo também tem que ser dividido em duas partes:

   (a) A lista SCL é varrida e, para todos os coeficientes que no início do passo dominate anterior já se encontravam nesta lista, é enviado mais um símbolo correspondenete ao plano de bits propriamente dito. Dependendo do sinal é somado ao coeficiente da imagem o valor $\pm 2^n$.

   (b) A lista IGL é varrida e, para cada coeficiente que no início do passo dominante anterior já se encontrava nesta lista, é enviado para a *string* o símbolo que indica se ainda se trata de um coeficiente insignificante ou se

já se tornou um coeficiente significante. Se este último caso se verificar, o coeficiente é reconstruído com o valor $\pm 2^n$ dependendo do seu sinal. O símbolo correspondente ao sinal do coeficiente também deve ser enviado para a *string*. Todas as coordenadas dos coeficientes que se tornaram significantes devem ser removidas da ICL e colocadas na SGL.

Tal como para a codificação do mapa de significâncias, para codificar os planos de bits propriamente ditos é suficiente utilizar os símbolos '1' e '0'. Assim, basta que o codificador aritmético seja inicializado no principio de cada iteração.

8. n é decrementado.

9. O processo é repetido desde o passo 5, e é interrompido em qualquer ponto quando o tamanho do *bitstream* exceder um limite pré-estabelecido.

O *bitstream* possui um cabeçalho de 8 bytes com informação extra necessária ao algoritmo de descodificação. Ele contém:

- o número de estágios (1 byte);
- as dimensões da imagem (4 bytes);
- a média da imagem (1 byte);
- maior plano de bits necessário para representar os coeficientes (1 byte);
- tamanho mínimo da região a codificar (1 byte).

As listas são inicializadas da mesma forma para o codificador e para o descodificador. Ambos vão ter um espelho da árvore quaternária que vai sendo gerada, podendo assim o descodificador seguir exactamente os mesmos passos do codificador, descodificando correctamente o *bitstream* que tinha sido gerado.

Em [10] é mostrado que para reduzir o erro médio de reconstrução da imagem é necessário que, no final do processo de descodificação, seja somado o correspondente a $2^{(i-1)}$ (i corresponde ao último plano de bits que foi comparado com o coeficiente) a cada coeficiente quantizado. Assim, no caso do algoritmo MGE faz-se:

1. Caso o processo tenha sido interrompido quando estavam a ser refinados os coeficientes da SCL, é feito o seguinte:

   (a) Caso já tenha sido somado durante este passo o valor $\pm 2^n$ a um dado coeficiente, é agora somado o valor $\pm 2^{(n-1)}$, dependendo do sinal.

   (b) Aos restantes coeficientes que se encontram na lista SCL, menos aqueles que nela foram colocados durante o passo dominate anterior, é somado o valor $\pm 2^n$, dependendo do seu sinal.

2. Caso o processo tenha sido interrompido durante o passo dominante, é feito o seguinte:

   (a) A todos os coeficientes que se tornaram significantes durante este passo é somado o valor $\pm 2^{(n-1)}$, dependendo do seu sinal.

   (b) A todos os coeficientes que antes do início deste passo já se encontravam na SCL é somado o valor $\pm 2^n$, dependendo do sinal.

4.5.1.4 Análise do algoritmo

O algoritmo descrito acima possui algumas características importante que o tornam bastante eficiente. Pode-se citar:

- A decomposição em *quadtree* permite uma rápida detecção dos coeficientes significantes na imagem;

- O uso de um alfabeto muito pequeno para codificar uma imagem permite que o codificador aritmético se adapte muito rapidamente a qualquer mudança nas estatísticas do símbolos;

- O facto de os coeficientes serem codificados de acordo com o seu valor absoluto, dando sempre prioridade àqueles que são maiores. Esta característica faz com que o nível de distorção seja sempre o menor possível pois, quando o processo é interrompido é garantido que os coeficientes não codificados são sempre aqueles que apresentam menor valor absoluto.

- É possível interromper o processo em qualquer altura, dependendo da taxa de bits que se pretende para a codificação da imagem.

4.5.2 Algoritmo MGEVQ

4.5.2.1 Estrutura do algoritmo

Este algoritmo é uma adaptação do MGE e foi desenvolvido no âmbito deste trabalho.

São duas as principais mudanças que existem entre o MGE e o MGEVQ. A primeira é que passam a ser processados vectores de coeficientes em vez de cada coeficiente individualmente. A segunda modificação reside no valor de referência deixar de ser dado por 2^n, onde n representa o plano de bits actual, e passar a ser o valor do coeficiente máximo multiplicado a cada iteração por um factor de escala α $(0,5 \leq \alpha < 1)$, ou seja, $\alpha^i \|\vec{V}\|_{max}$ onde i representa o número de iterações que já foram efectuadas.

A estrutura do algoritmo MGEVQ é a seguinte:

- **Transformada da imagem**: transformada *wavelet*;
- **Método de quantização**: aproximação sucessiva de vectores;
- **Unidade de codificação**: vectores de coeficientes;
- **Codificação do mapa de significâncias**: decomposição em *quadtree*;
- **Alfabeto**: 2 símbolos ou $m + 1$ símbolos, onde m corresponde ao número de vectores de orientação que constituem o dicionário;
- **Codificação dos símbolos**: codificador aritmético adaptativo.

4.5.2.2 Linhas gerais do algoritmo

Os parâmetros de entrada deste algoritmo são os seguintes:

- Nome da imagem a codificar;

- Número de níveis de decomposição para a *wavelet* utilizada;
- Taxa de bits pretendida na codificação (bits/pixel);
- Nível máximo de decomposição em *quadtree*;
- Valor do factor de escala α.

Este algoritmo usa duas listas que são de grande importância durante o passo de refinamento:

- **Lista de Vectores Significantes**, aqui denominada por SVL.
- **Lista de Vectores Insignificantes**, aqui denominada por IVL.

Em termos gerais, a codificação de uma imagem estática, usando este algoritmo, é implementada da seguinte forma:

É feita a transformada *wavelet* da imagem. As listas IVL e SVL são inicializadas como estando vazias.

Usando este algoritmo, a codificação é feita em dois passos:

No passo dominante, a região a codificar é testada para ver se existe algum vector de coeficientes significante e caso haja, a região é dividida em 4 partes iguais, caso contrário deixa de ser dividida. Esta divisão pára quando é atingido um nível máximo de decomposição pré-definido, ou seja, quando é atingido um determinado tamanho para a região. Dependendo deste tamanho pode-se ter regiões que contêm apenas um vector ou então que contêm mais. Assim, quando é definido que a região tem tamanho mínimo maior do que um vector pode-se ter as seguintes situações: apenas um dos vectores que constituem a região é significante ou mais do que um vector são significantes. Assim, são colocadas na lista IVL as coordenadas de todos os vectores que pertencem à região, mas que nesta altura do processo de codificação ainda continuam sendo insignificantes e é apenas enviado o mapa de significância e na SVL são colocadas as coordenadas correspondentes àqueles vectores que nesta altura se apresentam como sendo significantes, sendo que neste caso é enviado o plano de bits actual.

Estas listas são varridas durante o passo de refinamento, mas apenas são testados aqueles vectores que já se encontravam na lista antes do início do passo dominante. Para os vectores da SVL é enviado mais um plano de bits, enquanto que os da IVL são testados para ver se se tornaram significantes ou não. Caso já sejam significantes é enviado o plano de bits actual, caso contrário é enviado mais um símbolo para o mapa de significâncias.

O processo repete-se até ser atingida a taxa de bits pretendida ou os planos de bits serem completamente codificados.

É importante referir que na implementação deste algoritmo é sempre guardado e actualizado, a cada plano de bits que vai sendo enviado, um espelho da decomposição em *quadtree*, de forma a que não seja enviada para a *string* informação redundante acerca da significância das regiões. Este princípio tem como base, que, se uma dada região é significante para um determinado plano de bits l_n, continuará a sê-lo para o plano de bits seguinte, l_{n-1}.

4.5.2.3 Implementação do algoritmo

O algoritmo de codificação MGEVQ pode ser descrito mais detalhadamente da seguinte forma:

1. A média da imagem é calculada e subtraída.
2. Aplica-se uma transformada *wavelet* com R estágios à imagem com média zero.
3. É encontrado o vector de coeficientes da transformada que tem maior amplitude $\|\vec{V}\|_{max}$.
4. O valor de referência inicial, l_0, é dado por $\alpha\|\vec{V}\|_{max}$, onde α depende do dicionário de vectores de orientação que será utilizado.
5. São inicializadas como vazias duas listas de vectores de coeficientes, chamadas de **lista de vectores significantes** (SVL) e **lista de vectores insignificantes** (IVL).

6. A região a codificar (G) tem o tamanho da imagem original e é marcada como sendo uma "região zero" ($T = 0$).

7. *Passo dominante*: Este passo pode ser dividido da seguinte forma:

   (a) Para cada "região zero" ($T = 0$), é enviado para a *string* um símbolo que indica se essa região é significante ou não, ou seja, é enviado:

   $$T^{n}_{(i,j)\in(G)}(C_{(i,j)}) \tag{4.6}$$

   onde:

   • T é definido como '1' ("região um") se o valor absoluto de pelo menos um vector de coeficientes que pertence à região é maior que 2^n, caso contrário T é definido como sendo zero;
   • n é o plano de bits que está a ser codificado;
   • G é a região que está a ser analisada;
   • $C_{(i,j)}$ diz respeito ao vector de coeficientes de coordenadas (i, j) que pertence à região G.

   Pode-se então ter uma das seguintes situações:

   i. Caso se trate de uma "região um" ($T = 1$) duas situações distintas podem acontecer:
      - A região ainda não atingiu o tamanho mínimo pré-definido. Neste caso ela é dividida em 4 novas regiões do mesmo tamanho e cada uma delas marcada como sendo "região zero" ($T = 0$). O processo continua no ponto 7a.
      - O tamanho da região é igual ao mínimo pré-definido. Neste caso são possíveis duas situações:
        – Caso o tamanho da região seja 1, é apenas enviado para a *string* o símbolo 'C_j' correspondente ao vector de orientação mais próximo, de entre aqueles que constituem o dicionário, uma vez que o símbolo que indica a significância do vector já foi enviado como sendo aquele que indica a significância da

região. As coordenadas do coeficiente são colocadas na lista SVL.

- Caso este tamanho seja maior que 1, na região pode haver vectores significantes e insignificantes (pelo menos um tem que ser significante). Assim, é enviado para a *string* o símbolo que indica a significância de cada um dos vectores que compõem a região. Caso o vector seja significante é também enviado para a *string* o símbolo 'C_j' correspondente ao vector de orientação mais próximo de entre aqueles que constituem o dicionário. Por fim, as coordenadas dos vectores que foram reconhecidos como sendo insignificantes são colocadas na IVL e as dos significantes na SVL.

ii. Caso se trate de uma "região zero" ($T = 0$) nada mais é feito e passa-se a analisar a região seguinte.

A codificação da *string* gerada é, neste caso, um pouco mais complexo. Para enviar o mapa de significâncias do vector podem ser gerados 2 símbolos diferentes '1' ou '0'. Desta forma, o codificador aritmético adaptativo responsável por fazer a codificação da *string* deve neste momento ser actualizado para o uso de 2 símbolos. Contudo, caso o coeficiente passe a significante, é necessário o envio do índice do vector de orientação mais próximo. Assim, nesta altura, o codificador aritmético adaptativo tem que ser actualizado para o uso de uma quantidade de símbolos igual àquela que constitui o dicionário de vectores de orientação.

8. *Passo de refinamento*: Este passo também tem que ser dividido em duas partes:

   (a) A lista SVL é varrida e, para todos os vectores que no início do passo dominante anterior já se encontravam nesta lista, é enviado mais um símbolo correspondente ao plano de bits propriamente dito. Desta forma, neste passo de refinamento é codificada a diferença entre o vector original e o vector reconstruído até ao momento, utilizando-se o valor de referência ac-

tual, l_n, e um vector de orientação escolhido do dicionário. Os índices destes novos vectores de orientação são codificados em um *bitstream* usando um codificador aritmético, que no início deste passo é inicializado para permitir o número de vectores que constitui o dicionário.

(b) A lista IGL é varrida e, para cada vector que no início do passo dominante anterior já se encontrava nesta lista, é enviado para a *string* o símbolo que indica se ainda se trata de um vector insignificante ou se já se tornou um coeficiente significante. Caso este último caso se verifique, é também enviado para a *string* o símbolo 'C_j' correspondente ao vector de orientação mais próximo de entre aqueles que constituem o dicionário.Todas as coordenadas dos vectores que se tornaram significantes devem ser removidas da IVL e colocadas na SGL.

9. O valor de referência l_n é multiplicado pelo factor de escala α.

10. O processo é repetido desde o passo 6, e é interrompido em qualquer ponto quando o tamanho do *bitstream* exceder um limite pré-estabelecido.

O *bitstream* possui um cabeçalho de 9 bytes com informação extra necessária ao algoritmo de descodificação. Ele contém:

- valor do factor de escala (1 byte);
- o número de estágios (1 byte);
- as dimensões da imagem (4 bytes);
- a média da imagem (1 byte);
- maior plano de bits necessário para representar os coeficientes (1 byte);
- tamanho mínimo da região a codificar (1 byte).

A listas são inicializadas da mesma forma para o codificador e para o descodificador. Ambos vão ter um espelho da árvore quaternária que vai sendo gerada, podendo assim o descodificador seguir exactamente os mesmos passos do codificador, descodificando correctamente o *bitstream* que tinha sido gerado.

4.5.2.4 Análise do algoritmo

O algoritmo descrito acima possui algumas características importante que o tornam bastante eficiente. A saber:

- A decomposição em *quadtree* permite uma rápida detecção dos coeficientes significantes na imagem;
- O facto de os vectores coeficientes serem codificados de acordo com o módulo da sua amplitude, dando sempre prioridade aqueles que são maiores. Esta característica faz com que o nível de distorção seja sempre o menor possível pois, quando o processo é interrompido é garantido que os vectores não codificados são sempre aqueles que apresentam menor módulo de amplitude.
- É possível interromper o processo em qualquer altura, dependendo da taxa de bits que se pretende para a codificação da imagem.

4.5.3 Resumo das principais características do MGE e do MGEVQ

Na tabela 4.5 é apresentado um resumo das principais características dos algoritmos MGE e MGEVQ.

Sobre as diferenças apresentadas é importante ressaltar o número de símbolos que é necessário usar para a representação da imagem para cada um dos algoritmos. Enquanto, que o MGE usa 2 símbolos, o MGEVQ usa bastantes mais, sendo que este número vai depender do número de vectores de orientação que constituem o dicionário, m. Assim sendo, existe uma perda de eficiência clara na parte de codificação por entropia que é consequência do facto do aumento significativo do número de símbolos. No entanto, uma vez que a quantização é efectuada de forma vectorial, ou seja, de cada vez que é codificado o mapa de significâncias de um vector na realidade estão a ser codificados MN coeficientes da transformada, existe uma economia muito grande nos símbolos que é necessário enviar para codificar esse mapa de significâncias (MN vezes menos símbolos).

	MGE	**MGEVQ**
Transformada da imagem	Transformada *wavelet*	Transformada *wavelet*
Método de quantização	Aproximação sucessiva de escalares	Aproximação sucessiva de vectores
Unidade de codificação	Coeficiente	Vector de coeficientes
Codificação mapa de significâncias	Decomposição em *quadtree*	Decomposição em *quadtree*
Alfabeto	2 símbolos	2 ou $m+1$ símbolos
Codificação dos símbolos	Codificador aritmético adaptativo	Codificador aritmético adaptativo
Valor de referência	Dividido por 2 a cada iteração	Multiplicado por α a cada iteração, $0,5 \leq \alpha < 1$

Tabela 4.5: Principais características da implemetação dos algoritmos MGE e MGEVQ.

4.6 Algoritmo SWEET e SWEETVQ

Nesta secção serão descritos o algoritmo SWEET e a sua extensão para planos de bits vectoriais, aqui denominado por SWEETVQ.

Este algoritmo tem uma filosofia de codificação bastante diferente daquela utilizada por todos os algoritmos descritos anteriormente. Neste algoritmo de codificação de imagens, são codificados todos os bits significativos de um dado coeficiente antes de se passar a codificar o próximo, o que é bastante diferente dos anteriores onde, a cada iteração, é codificado um plano de bits de todos os coeficientes da transformada da imagem antes de se passar para o próximo.

Devido a este novo método de codificação, a taxa de bits desejada deixa de ser um parâmetro de entrada pois este tipo de codificação não permite que seja efectuado um controlo preciso da mesma.

Tal como o MGE, este algoritmo localiza os coeficientes significativos da imagem através de uma decomposição em *quadtree*. Assim, o SWEET usa uma

"grelha" que se vai dividindo sobre o domínio da transformada da imagem, e vai encontrando, de uma forma muito eficiente, os coeficientes que são significantes para a codificação.

Para melhor entender como é esta decomposição em *quadtree*, um exemplo dela é apresentado a figura 4.6. Neste exemplo considera-se uma imagem com apenas um coeficiente significante e é apresentada a forma como a imagem é dividida até esse coeficiente ser encontrado.

4.6.1 Algoritmo SWEET [2]

4.6.1.1 Estrutura do algoritmo

A estrutura do algoritmo SWEET é a seguinte:

- **Transformada da imagem**: transformada *wavelet*;
- **Método de quantização**: quantização escalar;
- **Unidade de codificação**: coeficiente;
- **Codificação do mapa de significâncias**: decomposição em *quadtree*;
- **Alfabeto**: 2 símbolos;
- **Codificação dos símbolos**: não usa.

4.6.1.2 Linhas gerais do algoritmo

Os parâmetros de entrada deste algoritmo são os seguintes:

- Nome da imagem a codificar;
- Número de níveis de decomposição para a *wavelet* utilizada;
- Qual o plano de bits máximo que se pretende codificar;
- Qual o plano de bits mínimo que se pretende codificar;
- Nível máximo da decomposição em *quadtrees*:

  - bloco 1×1;
  - bloco 2×2;
  - bloco 4×4;
  - bloco 8×8.

- Qual o tipo de arredondamento que se pretende:
  - nenhum;
  - codificador;
  - descodificador.

Antes da descrição do algoritmo é importante definir algumas das variáveis que são utilizadas. Assim:

- *BitMax* - Esta variável corresponde ao índice do maior plano de bits com que se pretende que os coeficientes da transformada sejam codificados. Este índice é tanto maior quanto maior for o valor associado ao bit correspondente. Esta variável é um parâmetro de entrada do programa;
- *BitMin* - Esta variável corresponde ao índice do menor plano de bits com que se pretende que os coeficientes da transformada sejam codificados. Este índice é tanto menor quanto menor for o valor associado ao bit correspondente. Esta variável é um parâmetro de entrada do programa;
- *nMax* - Esta variável é o valor de *BitMax* que corresponderia a uma imagem sem pré-quantização ($q = 1$). Esta pré-quantização é apresentada na equação 4.7.
- *nMaxReg* - A definição desta variável é muito equivalente à do *nMax*. Ou seja, *nMax* está para a imagem assim como *nMaxReg* está para uma região.

Em termos gerais, a codificação de uma imagem estática usando este algoritmo é implementada da seguinte forma:

É feita a transformada *wavelet* da imagem original. Em seguida, é feita uma pré-quantização dos coeficientes onde todos são escalonados por um factor q. Este factor é calculado levando em conta o bit máximo que é necessário para quantizar o coeficiente máximo da transformada e o número de bits máximo que se deseja usar para codificar toda a imagem. Assim, q é dado pela seguinte expressão:

$$q = \frac{2^{(nMax+1)} - 1}{2^{(BitMax+1)} - 1} \tag{4.7}$$

onde:

- q é o factor de escalonamento;
- $nMax$ é o bit máximo necessário para quantizar o maior coeficiente da transformada da imagem;
- $BitMax$ é o número de bits máximo que se deseja usar na codificação da imagem.

Para a região que está ser codificada é encontrado o bit máximo, $nMaxReg$, necessário para quantizar o maior coeficiente. Caso seja maior que o bit mínimo pré-definido, $BitMin$, a região é dividida em 4 subregiões iguais. Esta divisão pára quando é atingido um nível máximo de decomposição pré-definido, ou seja, quando é atingido um determinado tamanho para a região. Quando é atingido este tamanho, todos os coeficientes da região são completamente codificados, começando desde o $nMaxReg$ e terminando no $BitMin$. Depois de codificar cada coeficiente é enviado um bit a dizer qual é o seu sinal.

A imagem transformada continua a ser dividida até que todos os seus coeficientes tenham sido codificados para as condicões pré-definidas.

4.6.1.3 Implementação do algoritmo

O algoritmo de codificação SWEET pode ser descrito mais detalhadamente da seguinte forma:

1. A média da imagem é calculada e subtraída.
2. Aplica-se uma transformada *wavelet* com R estágios à imagem com média zero.

3. É calculado o maior plano de bits que é necessário para representar o maior coeficiente da transformada da imagem, ou seja:

$$n = \lfloor \log_2(\max_{(i,j)} \|C_{i,j}\|) \rfloor \tag{4.8}$$

onde:

- n representa o maior plano de bits;
- (i, j) as coordenadas do coeficiente;
- $\|C_{i,j}\|$ a amplitude do coeficiente para as coordenadas (i, j).

4. É feita uma pré-quantização dos coeficientes onde todos são escalonados por um factor q. Este factor é calculado levando em conta o bit máximo que é necessário para quantizar o coeficiente máximo da transformada e o bit máximo que se deseja usar para codificar toda a imagem. O cálculo do factor q é dado pela equação 4.7.

5. Inicialmente, a região a codificar G tem o tamanho da imagem original.

6. Início da codificação da região:

   (a) Encontrar o coeficiente da transformada de maior valor absoluto na região G que vai ser codificada.

   (b) Encontrar $nMaxReg$ que corresponde ao bit máximo necessário para codificar o maior coeficiente da região G.

   (c) Pode-se ter duas situações diferentes:

      i. Caso o bit máximo da região seja maior ou igual que o bit mínimo pré-definido, ou seja, $nMaxReg \geq BitMin$ faz-se:

         A. É enviado para a *string* um conjunto de símbolos de forma a indicar o bit máximo da região. Ou seja, são enviados para a *string* tantos símbolos '0' quanto a diferença entre o bit máximo que se pretende utilizar para a codificação da imagem, $BitMax$, e o bit máximo da região em questão, $nMaxReg$ e em seguida

é enviado o símbolo '1'. Desta forma são enviados para a *string* $BitMax - nMaxReg$ simbolos '0' e em seguida o símbolo '1'.

B. Pode-se, de novo, ter duas situações diferentes:

- Caso a região G tenha atingido o tamanho mínimo pré-definido, é codificado cada coeficiente da seguinte forma:
  - Caso tenha sido pré-definido que deveria existir arredondamento no codificador, é somado o valor $2^{(BitMin-1)}$ ao coeficiente. O objectivo deste arredondamento é diminuir o erro médio de reconstrução.
  - São enviados para a *string* todos os símbolos correspondendentes a cada plano de bit, desde $nMaxReg$ até $bitMin$. Estes símbolos podem ser '0' ou '1'.
  - É enviado para a *string* o símbolo correspondente ao sinal do coeficiente. Estes símbolos podem ser '0' ou '1'.
- Caso a região G ainda não tenha atingido o tamanho mínimo pré-definido, esta é dividida em 4 subregiões e, para cada uma delas é repetido tudo desde o ponto 6.

ii. Caso o bit máximo da região seja menor que o bit mínimo pré-definido, ou seja, $nMaxReg < BitMin$ faz-se:

A. É enviado para a *string* um conjunto de símbolos que vai indicar que a região não será codificada. Ou seja é enviado para a *string* tantos '0' quanto o número de planos de bits que foi pré-definido para serem codificados. Desta forma são enviados para a *string* $BitMax - BitMin + 1$ símbolos '0'.

(d) Sai deste passo quando todas as regiões da transformda da imagem tiverem sido codificadas para as condições pré-definidas.

Como pode ser visto, para a codificação de uma imagem usando o algoritmo SWEET é suficiente o uso de apenas 2 símbolos, o '0' e o '1'.

O *bitstream* possui um cabeçalho de 11 bytes com informação extra necessária ao algoritmo de descodificação. Ele contém:

- o número de estágios (1 byte);
- as dimensões da imagem (4 bytes);
- a média da imagem (1 byte);
- plano de bits máximo que se pretende codificar (1 byte);
- plano de bits mínimo que se pretende codificar (1 byte);
- tamanho mínimo da região a codificar (1 byte);
- factor de escalonamento dos coeficientes (1 byte);
- qual o tipo de arredondamento (1 byte);

O descodificador é implementado da mesma forma que o codificador permitindo que estejam sincronizados a qualquer momento de forma a que o *bitstream* seja lido de forma correcta.

Caso tenha sido pré-definido que deveria existir arredondamento no descodificador, é somado o valor $2^{(BitMin-1)}$ a todos os coeficientes que foram quantizados. O objectivo deste arredondamento é diminuir o erro médio de reconstrução.

4.6.1.4 Análise do algoritmo

O algoritmo descrito acima possui algumas características especiais, entre elas pode-se citar:

- A decomposição em *quadtree* permite uma rápida detecção dos coeficientes significantes na imagem;
- É difícil fazer um controlo preciso para a taxa de bits. Para isto, o valor do passo de pré-quantização q tem que se variado até a taxa de bits ser atingida.

4.6.2 Algoritmo SWEETVQ

4.6.2.1 Estrutura do algoritmo

Este algoritmo é uma adaptação do SWEET e foi desenvolvido no âmbito deste trabalho.

Ao contrário do que acontece para todos os outros algoritmos implementados, a adaptação do algoritmo SWEET a planos de bits vectoriais não é trivial, pois obriga a uma série de modificações que são apresentadas em seguida.

A estrutura do algoritmo SWEETVQ é a seguinte:

- **Transformada da imagem**: transformada *wavelet*;
- **Método de quantização**: aproximação de vectores;
- **Unidade de codificação**: vector de coeficientes;
- **Codificação do mapa de significâncias**: decomposição em *quadtree*;
- **Alfabeto**: 2 símbolos ou $m + 1$ símbolos, onde m corresponde ao número de vectores de orientação que constituem o dicionário
- **Codificação dos símbolos**: não usa.

4.6.2.2 Linhas gerais do algoritmo

Os parâmetros de entrada deste algoritmo são os seguintes:

- Nome da imagem a codificar;
- Número de níveis de decomposição para a *wavelet* utilizada;
- Número máximo de planos de bits que se pretende usar na codificação;
- Nível máximo da decomposição em *quadtree*:
  - bloco 1×1;
  - bloco 2×2;

  - bloco 4×4;
  - bloco 8×8.

- Valor do factor de escala α.

A taxa de bits é controlada variando o número máximo de planos de bits que se pretende usar para a codificação, $PlanMax$.

Em termos gerais, a codificação de uma imagem estática usando este algoritmo é implementada da seguinte forma:

É feita a transformada *wavelet* da imagem original. É, em seguida, calculado o valor de referência para inicializar a codificação.

Para a região que está ser codificada é encontrado o número máximo de planos de bits, $PlanMaxReg$, necessário para quantizar o maior vector de coeficientes. É então encontrado o valor de referência inicial para codificar esta região. Caso este valor seja maior do que o valor de referência mínimo pré-definido para a codificação dos vectores de coeficientes da transformada da imagem, a região é dividida em 4 subregiões iguais. Esta divisão pára quando é atingido um nível máximo de decomposição pré-definido, ou seja, quando é atingido um determinado tamanho para a região. Quando é atingido este tamanho, todos os vectores de coeficientes da região são completamente codificados, começando no valor de referência máximo para a região, que vai sendo multiplicado pelo factor de escala α, e parando quando este valor de referência tiver atingido o valor mínimo pré-definido.

A imagem transformada continua a ser dividida até que todos os seus coeficientes tenham sido codificados para as condições pré-definidas.

4.6.2.3 Implementação do algoritmo

O algoritmo de codificação SWEETVQ pode ser descrito mais detalhadamente da seguinte forma:

1. A média da imagem é calculada e subtraída.

2. Aplica-se uma transformada *wavelet* com R estágios à imagem com média zero.

3. É encontrado o vector de coeficientes da transformada que tem maior amplitude $\|\vec{V}\|_{max}$.

4. O valor de referência inicial, l_0, é dado por $\alpha\|\vec{V}\|_{max}$, onde α depende do dicionário de vectores de orientação que será utilizado.

5. Inicialmente, a região a codificar G tem o tamanho da imagem original.

6. Início da codificação da região:

   (a) É encontrado, na região G que está a ser codificada, o vector de coeficientes da transformada que tem maior amplitude $\|\vec{V}\|_{\max\in G}$.

   (b) Calcular o número de planos de bits necessários para codificar o maior vector da região, $PlanMaxReg$. Este número de planos de bits irá depender do factor de escala α que foi escolhido.

   (c) É encontrado qual o valor de referência inicial para a codificação desta região.

   (d) Pode-se ter duas situações diferentes:

      i. Caso o valor de referência inicial para codificar esta região seja maior que o valor de referência mínimo pré-definido para codificar a imagem, faz-se:

         A. É enviado para a *string* um conjunto de símbolos que indicam quantas vezes é que o valor de referência inicial para a imagem tem que ser multiplicado pelo factor de escala α para se poder representar o vector de coeficiente máximo da região. Ou seja, é enviado para a *string* tantos símbolos '0' quanto o número de vezes que esta multiplicação necessitou ser feita. Em seguida é enviado o símbolo '1'.

         B. Pode-se, de novo, ter agora duas situações diferentes:

            - Caso a região G tenha atingido o tamanho mínimo pré-definido, é codificado cada vector de coeficientes da seguinte forma:

- É enviado para a *string* um símbolo que indica se o vector é significante ou não para o plano de bits actual.
- Caso o vector seja significante é enviado para a *string* um símbolo 'C_j' correspondente ao vector de orientação mais próximo, de entre aqueles que constituem o dicionário.
- O valor de referência da região é multiplicado pelo factor de escala α.
- O vector continua a ser codificado até que o valor de referência fique menor que o mínimo pré-definido.

• Caso a região G ainda não tenha atingido o tamanho mínimo pré-definido, esta é dividida em 4 subregiões e, para cada uma delas é repetido tudo desde o ponto 6.

ii. Caso o valor de referência inicial para codificar esta região seja menor que o valor de referência mínimo pré-definido para codificar a imagem, faz-se:

A. É enviado para a *string* um conjunto de símbolos que vai indicar que a região não será codificada. Ou seja é enviado para a *string* tantos '0' quanto o número de planos de bits que foram pré-definidos para serem codificados.

(e) Sai deste passo quando todas as regiões da transformada da imagem tiverem sido codificadas para as condições pré-definidas.

Como pode ser visto, para a codificação de uma imagem usando o algoritmo SWEETVQ são necessários tantos símbolos quanto o número de vectores que constituem o dicionário.

O *bitstream* possui um cabeçalho de 13 bytes com informação extra necessária ao algoritmo de descodificação. Ele contém:

- o número de estágios (1 byte);
- as dimensões da imagem (4 bytes);

- a média da imagem (1 byte);
- plano de bits máximo que se pretende codificar (1 byte);
- número máximo de planos de bits que se pretende codificar (1 byte);
- o valor de referência inicial (4 bytes);
- factor de escala α (1 byte).

O descodificador é implementado da mesma forma que o codificador permitindo que estejam sincronizados a qualquer momento de forma a que o *bitstream* seja lido de forma correcta.

Para optimizar o desempenho deste algoritmo, foi criada uma árvore da decomposição em *quadtree* com o objectivo de evitar enviar para a *string* informação desnecessária. Assim, quando uma região vai ser codificada e tem que ser enviado o número de vezes que o valor de referência dessa região tem que ser multiplicado pelo factor de escala α, o que é enviado para a *string* passa a ser tantos zeros quanto a diferença entre este valor e aquele que foi enviado para a sua região pai. Os resultados que são apresentados no capítulo 5 dizem respeito à implementação desta versão.

4.6.2.4 Análise do algoritmo

O algoritmo descrito acima possui algumas características especiais, entre elas pode-se citar:

- A decomposição em *quadtree* permite uma rápida detecção dos coeficientes significantes na imagem;
- É difícil fazer um controlo preciso para a taxa de bits. Para isto, é necessário variar o valor de referência inicial até que seja atingida a taxa de bits necessária.

4.6.3 Resumo das principais características do SWEET e o SWEETVQ

Na tabela 4.6 é apresentado um resumo das principais características dos algoritmos SWEET e SWEETVQ.

	SWEET	**SWEETVQ**
Transformada da imagem	Transformada *wavelet*	Transformada *wavelet*
Método de quantização	Quantização escalar	Aproximação de vectores
Unidade de codificação	Coeficiente	Vector de coeficientes
Codificação mapa de significâncias	Decomposição em *quadtree*	Decomposição em *quadtree*
Alfabeto	2 símbolos	2 ou $m+1$ símbolos
Codificação dos símbolos	Não usa	Não usa
Valor de referência	Dividido por 2 a cada iteração	Multiplicado por α a cada iteração, $0,5 \leq \alpha < 1$

Tabela 4.6: Principais características da implemetação dos algoritmos SWEET e SWEETVQ.

Sobre as diferenças apresentadas é importante ressaltar o número de símbolos que é necessário usar para a representação da imagem para cada um dos algoritmos. Enquanto que o SWEET usa 2 símbolos, o SWEETVQ usa bastante mais, sendo que este número vai depender do número de vectores de orientação que constituem o dicionário m. Assim sendo, existe uma perda de eficiência clara na parte de codificação por entropia, que é consequência do facto do aumento significativo do número de símbolos. No entanto, uma vez que a quantização é efectuada de forma vectorial, ou seja, de cada vez que é codificado o mapa de significâncias de um vector na realidade estão a ser codificados MN coeficientes da transformada, existe uma economia muito grande no número de símbolos que é necessário enviar para codificar

esse mapa de significâncias (MN vezes menos símbolos).

Capítulo 5

Análise dos algoritmos

5.1 Introdução

Neste capítulo é feita uma análise de desempenho dos algoritmos apresentados no capítulo 4. Espera-se, com esta análise, obter um entendimento mais profundo acerca do comportamento de cada um deles quando implementados tanto usando planos de bits escalares quanto vectoriais. É comparado o número de bits que são gastos no envio da informação de significância com aqueles que são gastos com o envio dos coeficientes propriamente ditos, incluindo os bits de sinal. É igualmente analisada a relação sinal-ruído (PSNR) obtida para os diferentes tipos de algoritmos.

Para comparar o desempenho dos diversos algoritmos foram utilizadas, como referências, as imagens Lena, Barbara, Boats, Girl, Gold e Zelda, todas com dimensão 512x512. No apêndice C são apresentadas as imagens de referência utilizadas neste trabalho. Todas elas foram reconstruídas com 3 diferentes taxas: 0,2 , 0,5 e 1 bit/pixel. A transformada *wavelet* foi baseada no banco de filtros biortogonal F definido em [5], com 6 níveis de decomposição.

Para os algoritmos SPIHT e MGE é feita a análise dos seus desempenhos com e sem o uso do codificador aritmético adaptativo [17], tanto no caso escalar quanto no vectorial.

Na implementação usando planos de bits vectoriais foram empregues dicionários com diferentes dimensões na codificação dos vectores: em dimensão 4,

foi usada a 1^a camada do reticulado D4; em dimensão 8, a 1^a camada do reticulado e8 e em dimensão 16, a 1^a camada do reticulado $\lambda 16$ tal como é definido no apêndice B.

Neste capítulo é mostrado o critério usado para a escolha do factor de escala α para cada um dos reticulados, de seguida é apresentada a variação da relação sinal-ruído em função da taxa de bits e por fim é abordada a forma como são alocados os bits na codificação da imagem.

5.2 Critério de escolha do factor de escala α

A forma utilizada para escolher o factor de escala α foi a seguinte: para cada imagem e para cada reticulado, o valor de α foi variado entre $0,5 \leq \alpha < 1$ e foi observado o que apresentou melhor relação sinal-ruído. Na figura 5.1 observa-se a variação do PSNR em função de α para as imagens Lena 512×512 e Boats para cada um dos três reticulados usados neste trabalho. O algoritmo utilizado para a obtenção destes gráficos foi o SAWVQ/C.

Desta forma, e seguindo este critério, os factores de escala α que foram utilizados para cada imagem e para cada um dos dicionários são os que se encontram apresentados na Tabela 5.1:

	Lena 512x512	Barbara 512x512	Boats 512x512	Girl 512x512	Gold 512x512	Zelda 512x512
D4	$\alpha = 0,58$	$\alpha = 0,60$	$\alpha = 0,58$	$\alpha = 0,60$	$\alpha = 0,60$	$\alpha = 0,60$
E8	$\alpha = 0,61$	$\alpha = 0,61$	$\alpha = 0,59$	$\alpha = 0,60$	$\alpha = 0,60$	$\alpha = 0,61$
$\lambda 16$	$\alpha = 0,63$	$\alpha = 0,62$	$\alpha = 0,64$	$\alpha = 0,64$	$\alpha = 0,63$	$\alpha = 0,64$

Tabela 5.1: Factores de escala por dicionário e por imagem.

Como o objectivo deste trabalho como a quantização vectorial por aproximações sucessivas se comporta quando aplicada nos algoritmos de codificação por planos de bits, optou-se por utilizar o α óptimo para cada imagem e reticulado. Num caso prático deve-se utilizar um α médio para cada reticulado com uma pequena

penalidade no desempenho. Pode ser usada uma variação no algoritmo de quantização vectorial por aproximações sucessivas proposta em [4], que proporciona uma menor sensibilidade da relação sinal-ruído em relação a α, que não foi considerada no âmbito deste trabalho.

5.3 Desempenho dos algoritmos segundo um critério de taxa×distorção

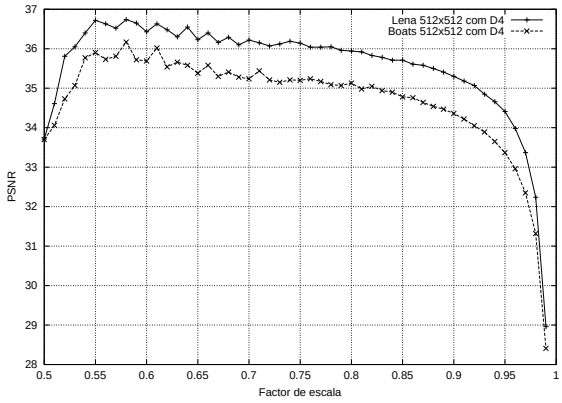
Nesta secção é feita uma comparação de desempenho em termos de relação sinal-ruído para as versões escalares e as suas adaptações vectoriais para cada um dos algoritmos descritos no capítulo 4. Nas figuras 5.2, 5.3, 5.4, 5.5 e 5.6 é apresentado o desempenho para as imagens Lena 512×512, Barbara e Boats.

Para uma avaliação mais precisa do desempenho dos diversos algoritmos implementados são também apresentadas duas tabelas com os valores da PSNR para todas as imagens para as taxas de 0,2, 0,5 e 1 bit/pixel. Na tabela 5.2 são apresentados os valores obtidos quando está a ser utilizada a quantização por aproximações sucessivas de grandezas escalares e na tabela 5.3 são apresentados os valores obtidos quando essas grandezas são vectoriais e os algoritmos são codificados usando os três reticulados.

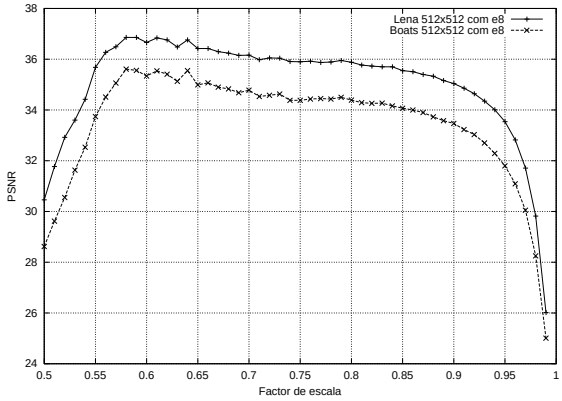
Através da observação destas figuras é possível ver que, qualquer que seja o algotimo empregue na codificação da imagem, os resultados obtidos com a sua versão vectorial são, em geral, melhores quando é usada a 1^a camada do reticulado $\lambda 16$.

Observando agora as tabelas 5.2 e 5.3 pode-se igualmente notar que para os restantes reticulados esta melhoria de desempenho também é observada, embora de uma forma não tão evidente.

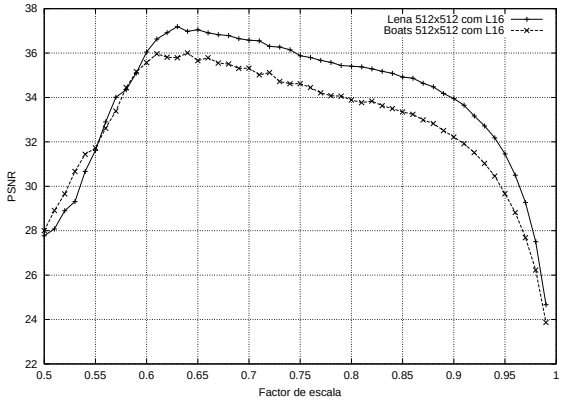
É possível igualmente detectar que, para o caso de utilização de uma quantização usando grandezas escalares, o algoritmo SWEET apresenta o melhor desempenho. Contudo é importante mais uma vez referir que, o controlo preciso da taxa de bits é muito difícil de efectuar para este algoritmo, uma vez que é necessário



(a)



(b)



(c)

Figura 5.1: Desempenho do SAWVQ/C em função de α para as imagens Lena 512×512 e Boats para 0,5 bits/pixel e para os diferentes dicionários: (a) 1^a camada do reticulado D4; (b) 1^a camada do reticulado e8; (c) 1^a camada do reticulado λ16.

ajustar os parâmetros (q, $BitMin$ ou $threshold$) de tal forma que seja possível obter a taxa de bits pretendida.

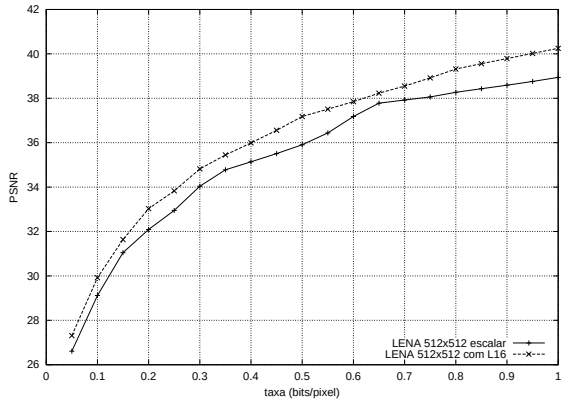
5.4 Análise comparativa do número de bits gastos nas diferentes etapas dos algoritmos

Agora é avaliado o comportamento dos algoritmos no que diz respeito à forma como os bits são gastos durante o processo de codificação. Isto é, qual a quantidade de bits que são gastos para enviar a informação de significância e aquela que é gasta para o envio dos planos de bits propriamente ditos.

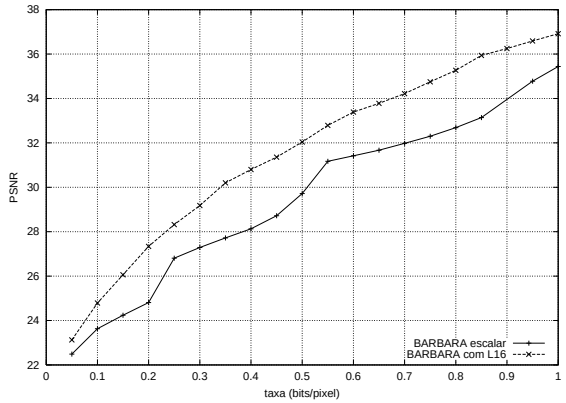
De forma a facilitar esta análise são apresentadas as seguintes figuras:

- C.1, C.2 e C.3, para a imagem Lena 512×512 para as taxas 0,2, 0,5 e 1 respectivamente;
- C.4, C.5 e C.6, para a imagem Barbara para as taxas 0,2, 0,5 e 1 respectivamente;
- C.7, C.8 e C.9, para a imagem Boats para as taxas 0,2, 0,5 e 1 respectivamente;
- C.10, C.11 e C.12, para a imagem Girl para as taxas 0,2, 0,5 e 1 respectivamente;
- C.13, C.14 e C.15, para a imagem Gold para as taxas 0,2, 0,5 e 1 respectivamente;
- C.16, C.17 e C.18, para a imagem Zelda para as taxas 0,2, 0,5 e 1 respectivamente;

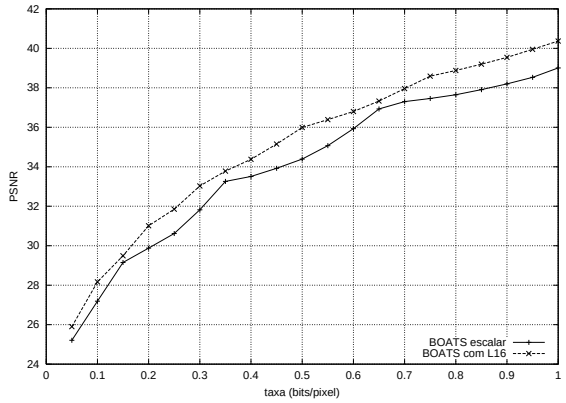
Na alínea (a) das figuras C.1-C.18, encontra-se apresentada a forma como os bits são distribuidos pelas várias etapas da codificação. Para o caso dos algoritmos SPIHT e MGE são apresentados os resultados obtidos quando implementados com e sem codificador aritmético. Uma diferença que se pode tirar dos vários algoritmos é o facto de, no EZW não ser possível diferenciar os bits que são gastos para o envio do



(a)

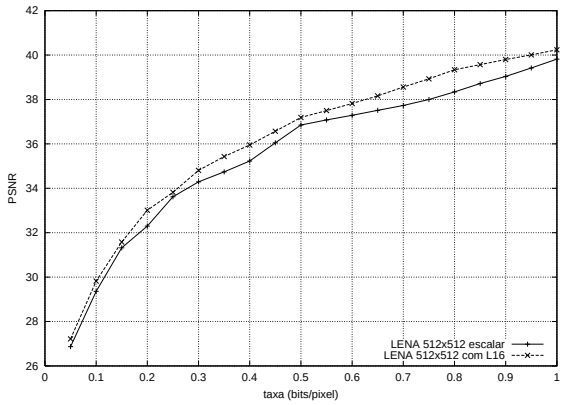


(b)

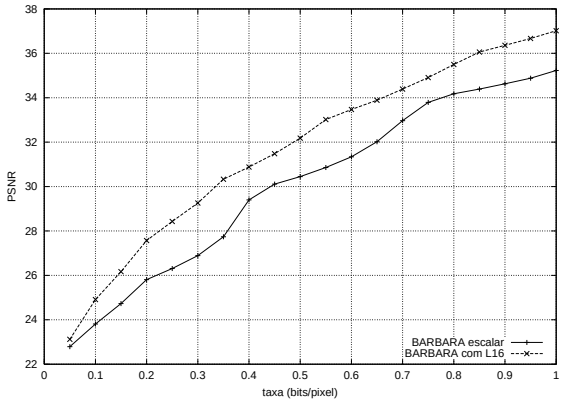


(c)

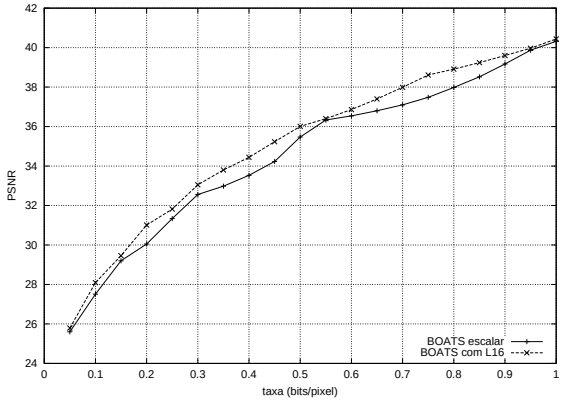
Figura 5.2: Desempenho dos algoritmos EZW e SAWVQ com dicionário $\lambda 16$ em função da taxa: (a) Imagem Lena 512×512; (b) Imagem Barbara; (c) Imagem Boats.



(a)

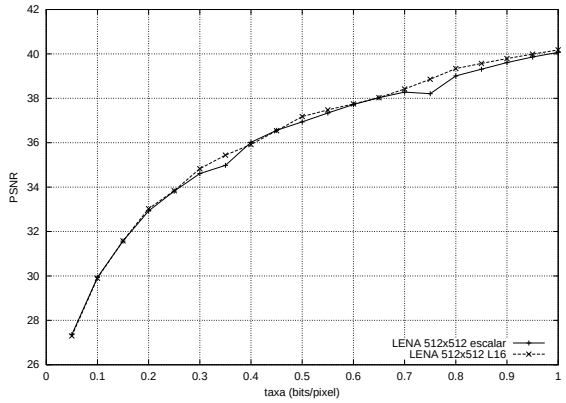


(b)

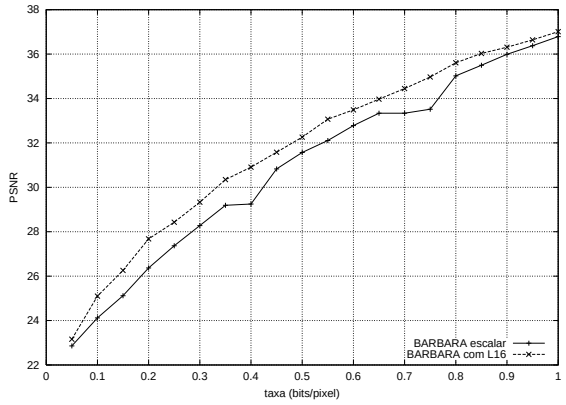


(c)

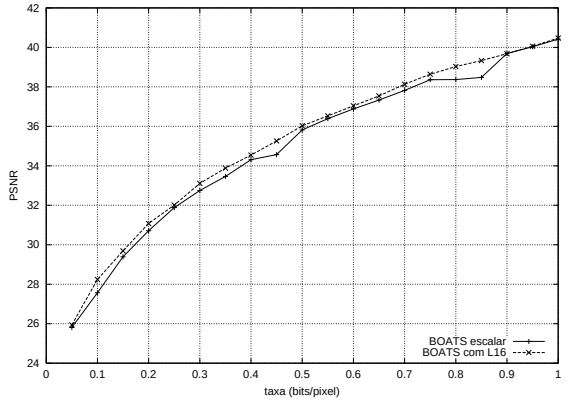
Figura 5.3: Desempenho dos algoritmos EZW/C e SAWVQ/C com dicionário $\lambda 16$ em função da taxa: (a) Imagem Lena 512×512; (b) Imagem Barbara; (c) Imagem Boats.



(a)

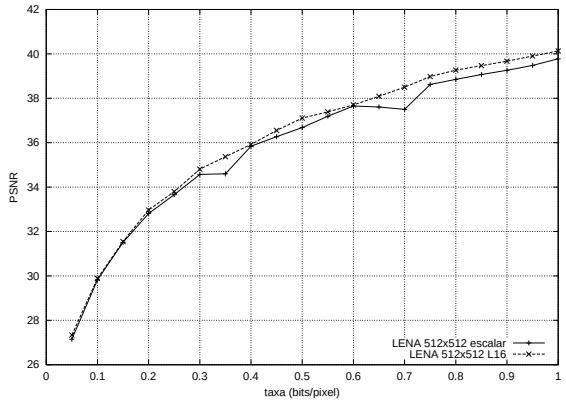


(b)

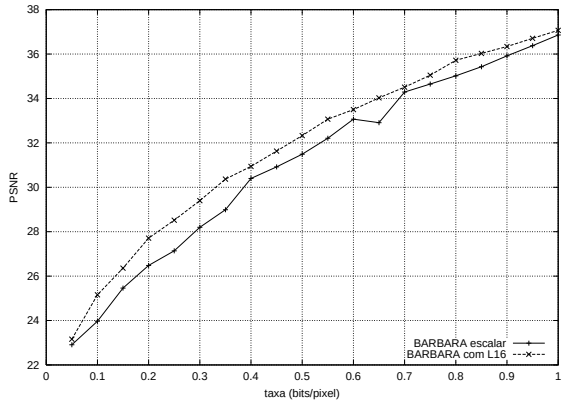


(c)

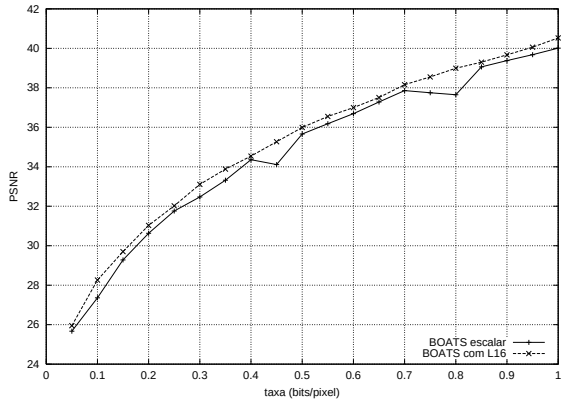
Figura 5.4: Desempenho dos algoritmos SPIHT e SPIHTVQ com dicionário $\lambda 16$ em função da taxa: (a) Imagem Lena 512×512; (b) Imagem Barbara; (c) Imagem Boats.



(a)

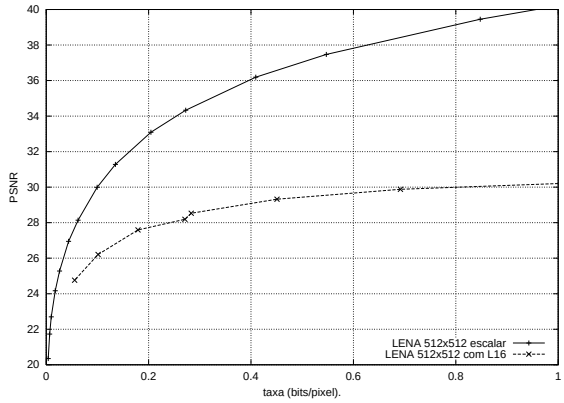


(b)

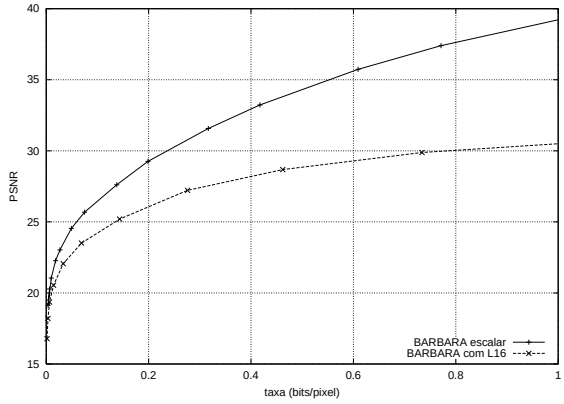


(c)

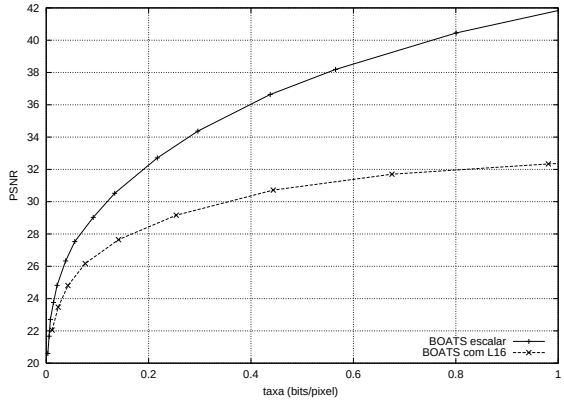
Figura 5.5: Desempenho dos algoritmos MGE e MGEVQ com dicionário $\lambda 16$ em função da taxa: (a) Imagem Lena 512×512; (b) Imagem Barbara; (c) Imagem Boats.



(a)



(b)



(c)

Figura 5.6: Desempenho dos algoritmos SWEET e SWEETVQ com dicionário $\lambda 16$ em função da taxa: (a) Imagem Lena 512×512; (b) Imagem Barbara; (c) Imagem Boats.

	Lena 512x512			Barbara			Boats			Girl			Gold			Zelda		
	0,2	0,5	1	0,2	0,5	1	0,2	0,5	1	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	31,90	36,07	40,02	29,41	32,57	35,47	35,65	39,02	40,95
EZW/C	32,30	36,85	39,82	25,81	30,45	35,23	30,05	35,47	40,32	32,50	37,18	41,29	29,53	32,56	35,91	36,32	39,13	41,79
SPIHT S/ COD	32,73	36,80	39,94	26,29	31,37	36,62	30,52	35,61	40,18	36,61	37,03	40,89	29,59	32,77	36,09	36,31	38,86	41,48
MGE S/ COD	32,69	36,58	39,60	26,45	31,37	36,68	30,45	35,51	39,75	32,72	37,09	40,93	29,63	32,86	36,17	35,93	39,28	41,07
SPIHT C/ COD	32,93	36,94	40,07	26,37	31,57	36,79	30,71	35,82	40,42	32,80	37,19	41,07	29,70	32,91	36,24	36,37	39,24	41,38
MGE C/ COD	32,81	36,68	39,78	26,48	31,49	36,88	30,63	35,66	40,01	32,81	37,23	41,15	29,71	32,98	36,34	35,89	39,34	41,06
SWEET	32,96	37,07	40,04	29,30	34,31	39,14	32,30	37,36	41,70	32,62	37,26	41,06	29,74	32,98	36,29	37,14	39,86	42,35

Tabela 5.2: Comparação entre as PSNR (dB) para os diferentes algoritmos usando planos de bits escalares.

		Lena 512x512			Barbara			Boats			Girl			Gold			Zelda		
		0,2	0,5	1	0,2	0,5	1	0,2	0,5	1	0,2	0,5	1	0,2	0,5	1	0,2	0,5	1
SAWVQ	D4	32,52	36,60	39,96	26,44	30,77	35,85	30,73	35,04	40,06	32,47	36,74	40,43	29,45	32,71	36,17	35,91	39,18	41,81
	e8	32,79	36,74	39,92	26,67	31,75	36,16	30,40	35,49	39,89	32,72	37,00	40,80	29,69	33,12	36,62	36,48	39,31	41,95
	λ16	33,03	37,18	40,25	27,34	32,04	36,92	31,01	35,99	40,37	32,88	37,42	41,10	29,91	33,49	36,71	36,47	39,49	41,94
SAWVQ/C	D4	32,62	36,74	40,08	26,64	31,24	36,14	30,77	35,21	40,13	32,59	37,03	40,61	29,55	32,91	36,34	35,99	39,25	41,85
	e8	32,79	36,84	40,00	26,95	31,90	36,57	30,42	35,56	40,00	32,79	37,17	40,94	29,72	33,26	36,66	36,48	39,34	42,06
	λ16	33,01	37,19	40,24	27,44	32,38	37,32	31,01	36,01	40,44	32,85	37,34	41,40	29,92	33,52	36,76	36,62	39,74	42,41
SPIHTVQ S/ COD	D4	32,46	36,23	39,36	26,58	30,90	35,93	30,32	34,98	39,27	32,44	36,52	40,46	29,55	32,57	35,64	36,04	39,15	41,37
	e8	32,76	36,63	40,09	27,07	31,88	36,43	30,58	35,58	39,87	32,84	37,00	41,07	29,82	33,17	36,66	36,49	39,33	41,95
	λ16	32,66	36,76	39,92	27,21	31,71	36,45	30,79	35,51	39,89	32,67	37,09	41,11	29,76	33,16	36,47	36,47	39,38	41,94
SPIHTVQ C/ COD	D4	32,72	36,55	40,02	26,83	31,25	36,48	30,78	35,33	40,15	32,71	36,85	40,90	29,72	32,82	36,27	36,26	39,38	41,86
	e8	32,84	36,77	40,17	27,22	32,05	36,62	30,69	35,70	40,00	32,88	37,13	41,15	29,87	33,24	36,71	36,53	39,35	42,00
	λ16	33,03	37,18	40,18	27,68	32,26	37,01	31,08	36,03	40,47	32,97	37,46	41,44	29,96	33,52	36,78	36,68	39,74	42,43
MGEVQ S/ COD	D4	32,44	36,19	39,51	26,64	30,95	35,89	30,36	34,96	39,33	32,47	36,52	40,40	29,52	32,54	35,69	35,96	39,10	41,50
	λ16	32,65	36,78	39,84	27,27	31,77	36,52	30,82	35,54	39,88	32,71	37,10	41,04	29,77	33,20	36,39	36,43	39,51	42,07
MGEVQ C/ COD	D4	32,62	36,56	39,93	26,82	31,30	36,41	30,73	35,24	39,93	32,71	36,90	40,83	29,68	32,82	36,26	36,16	39,27	41,75
	λ16	32,97	37,11	40,13	27,71	32,33	37,07	31,03	35,99	40,53	32,99	37,45	41,38	29,97	33,48	36,72	36,59	39,69	42,33
SWEETVQ	D4	30,38	34,00	36,65	26,90	31,34	34,73	26,90	33,55	36,98	30,24	33,92	37,03	27,03	30,14	33,47	34,62	37,31	38,76
	λ16	27,73	29,00	30,22	26,24	29,38	30,17	28,02	30,82	32,19	29,55	31,30	32,95	25,27	26,53	27,61	33,20	34,88	35,73

Tabela 5.3: Comparação entre as PSNR (dB) para os diferentes algoritmos usando planos de bits vectoriais.

sinal dos coeficientes. Neste caso, estes bits encontram-se embutidos nos bits gastos com o envio da significância. Em qualquer um dos outros algoritmos estes bits podem-se diferenciar dos restantes. Pode-se também observar nestes gráficos que o comportamento dos diversos algoritmos implementados com planos de bits escalares é muito similar. O que mais pesa para este tipo de quantização é o envio dos bits correspondentes à informação de significância. Apenas para o caso do algoritmo SWEET é que esta característica se inverte totalmente, ou seja, são gastos muito mais bits com a informação referente aos planos de bits propriamente ditos do que na informação referente à significância. Esta caraterística deve-se ao facto de, neste algoritmo, quando é atingida uma região com um tamanho pré-definido (1×1, 2×2, 4×4 ou 8×8) que contém pelo menos um coeficiente significante, este e todos os restantes coeficientes que a constituem são completamente codificados, quer sejam significantes quer não. Este algoritmo apresenta um óptimo desempenho, como pode ser observado na tabela 5.2, mas, como referido, tem o problema de não permitir que seja feito facilmente um controlo de taxa.

Nas alíneas (b) das figuras C.1-C.18 são comparados os diferentes algoritmos usando quantização vectorial e codificados usando o reticulado $\lambda 16$. Como no caso da comparação dos escalares, aqui também são apresentados os resultados obtidos para os algoritmos SPIHTVQ e MGEVQ implementados com e sem codificador aritmético. Pode-se então notar que o comportamento apresentado pelos algoritmos é muito parecido, ou seja, todos gastam poucos bits com a codificação da informação de significância e muitos bits com a informação referente aos planos de bits. A similaridade existente entre os diferentes algoritmos vectoriais pode, igualmente, ser vista através da tabela 5.3 onde se mostra que não existe uma variação muito grande da PSNR obtida na reconstrução das várias imagens e diferentes taxas. No caso do SWEETVQ, como mais adiante é explicado, esta similaridade com os restantes algoritmos não se verifica.

Em seguida, são comparados, individualmente, cada um dos codificadores implementados usando planos de bits escalares com a versão que usa planos de bits vectoriais. Esta versão é apresentada usando dicionários de diferentes dimensões na

codificação do vectores. Aqui, os resultados apresentados para o caso dos algoritmos SPIHT e MGE dizem respeito às suas implementações usando codificador aritmético.

Das alíneas (c)-(g) das figuras C.1-C.18 pode-se observar que, no caso escalar a maior parte dos bits é gasta com a informação de significância. No caso vectorial verifica-se o inverso, sendo muito maior o número de bits gastos com a informação referente aos planos de bits de vectores de coeficientes. Este comportamento é justificado pelos seguintes factos:

- No caso vectorial, no lugar de codificar um coeficiente individualmente passa-se a codificar um grupo de coeficientes que, para os exemplos apresentados, podem ser constituidos por 4, 8 ou 16 coeficientes. Assim, os bits que necessitaram ser gastos com a informação de significância são na ordem de 4, 8 ou 16 vezes menor;

- Ao contrário do caso escalar onde o factor de escala tem sempre o valor de 0,5, no caso vectorial este valor é maior o que obriga a que sejam codificados mais planos de bits para enviar a informação;

- No caso vectorial, o envio de um plano de bits gasta muito mais bits do que no caso escalar, uma vez que são necessários muitos mais bits para codificar um vector do que um escalar.

Pode também ser observado que, à medida que aumenta a dimensão do dicionário usado para a codificação dos vectores, maior é a quantidade de informação referente aos planos de bits e menor aquela que diz respeito à significância. Esta diminuição do número de bits gastos com a informação de significância é justificada pelo facto de apenas se ter uma significância vectorial, ou seja, se um símbolo indica que um vector é significante, este mesmo símbolo vai indicar a significância de um número de coeficientes igual à dimensão do vector. Assim sendo, no caso vectorial tem-se uma diminuição do número de bits gastos com a informação de significância da mesma ordem da dimensão do vector utilizado para a codificação. Por outro lado, o aumento do número de bits gastos com a codificação dos planos de bits tem duas causas: A primeira está relacionada com o facto de num vector significante

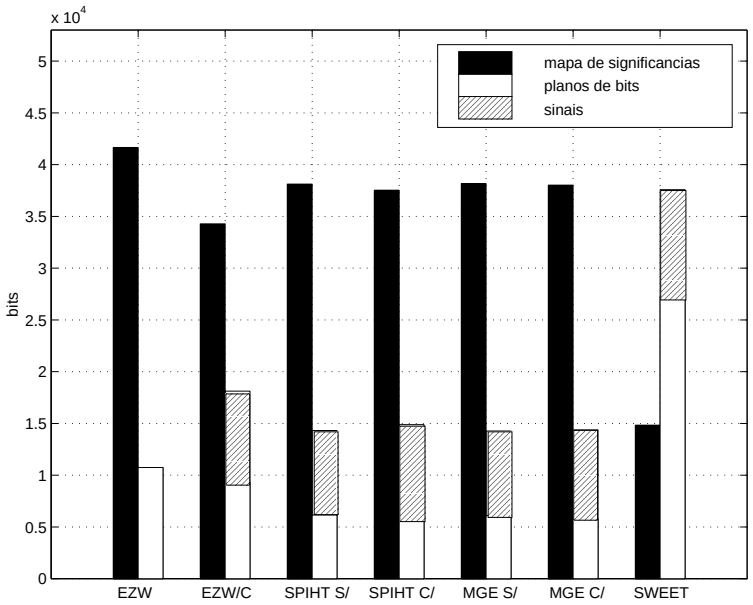
poder haver diversas coordenadas insignificantes, fazendo com que os planos de bits vectoriais também carreguem informação que, no caso escalar, é puramente de significância. Uma segunda causa para este comportamento, é a diferente eficiência intrínseca da representação usando planos de bits vectoriais em relação àquela que usa planos de bits escalares, conforme mencionado acima.

É importante notar, que a principal diferença entre os codificadores usando planos de bits, encontra-se na forma como a informação de significância é transmitida. No caso escalar, isto é confirmado pelas alíneas (a) das figuras C.1-C.18, onde se vê que são gastos muito mais bits com o envio da informação de significância do que com o envio dos planos de bits. Assim, neste tipo de codificador, quanto maior a eficiência na codificação da informação referente à significância, melhor o desempenho obtido. Contudo, as alíneas (b) das figuras C.1-C.18, mostram que, para o caso do uso de codificadores vectoriais, a quantidade de bits gastos com a significância é muito baixa quando comparada com aquela gasta com os planos de bits. Desta forma, é de se esperar que os desempenhos dos diversos codificadores vectoriais sejam equivalentes, como pode ser confirmado por inspecção da tabela 5.2.

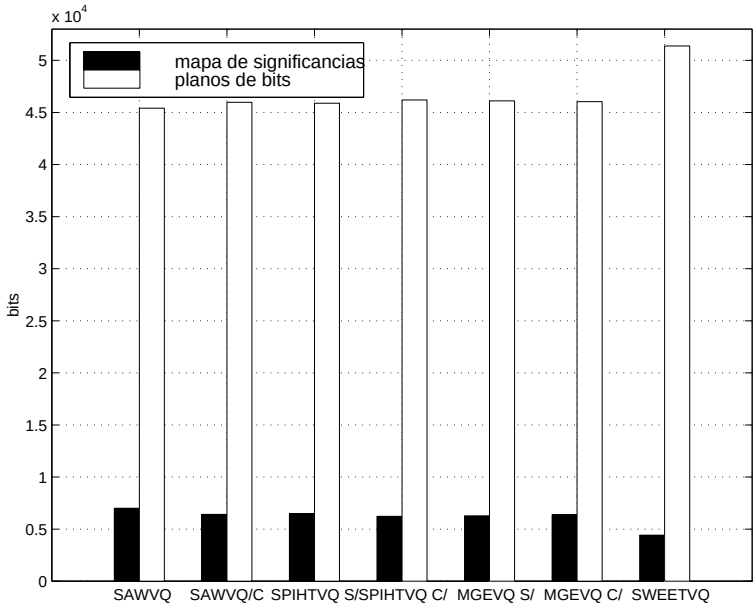
Conforme mencionado anteriormente, o SWEET gasta muitos mais bits com o envio dos planos de bits do que com a informação de significância, uma vez que para cada região, com um tamanho pré-definido (1×1, 2×2, 4×4 ou 8×8), todos os restantes coeficientes que a constituem são completamente codificados, quer sejam significantes quer não. No caso vectorial, SWEETVQ, esta característica é particularmente desvantajosa pois quanto maior a dimensão do vector maior o α necessário para a codificação e, consequentemente, maior o número de planos de bits que é necessário codificar para que se atinja um determinado nível de distorção. A conjunção destes factores faz com que a eficiência de codificação em planos de bits vectoriais diminua quando este algoritmo é utilizado. Este comportamento pode ser confirmado pela tabela 5.3.

Finalmente, uma comparação entra as tabelas 5.2 e 5.3 mostra que, em geral, para os algoritmos SAWVQ, SAWVQ/C, SPIHTVQ e MGEVQ, usando como di-

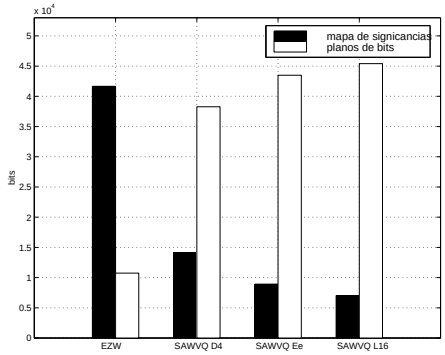
cionário o reticulado $\lambda 16$, os codificadores usando planos de bits vectoriais têm um desempenho superior. Isto é uma indicação de que o uso deste tipo de codificadores pode levar a um aumento do desempenho.


x 10^4
mapa de significancias
planos de bits
sinais
bits
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
EZW
EZW/C
SPIHT S/
SPIHT C/
MGE S/
MGE C/
SWEET

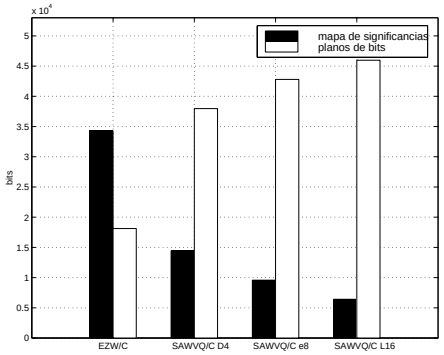
(a)


x 10^4
mapa de significancias
planos de bits
bits
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
SAWVQ
SAWVQ/C
SPIHTVQ S/
SPIHTVQ C/
MGEVQ S/
MGEVQ C/
SWEETVQ

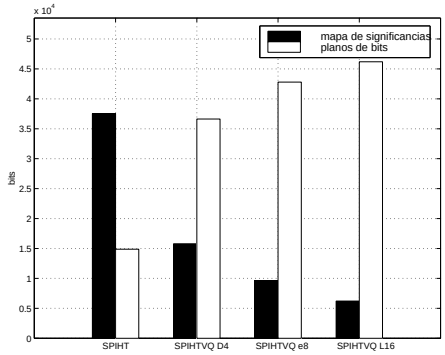
(b)



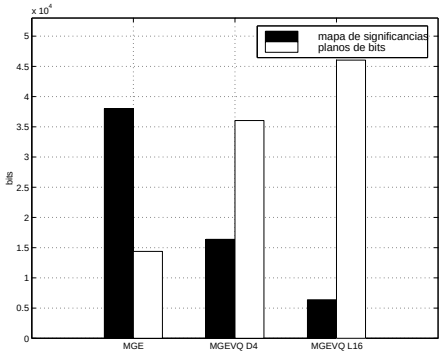
(c)



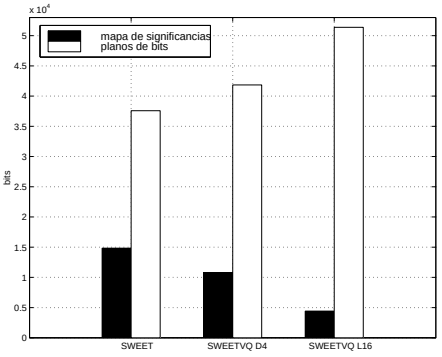
(d)



(e)

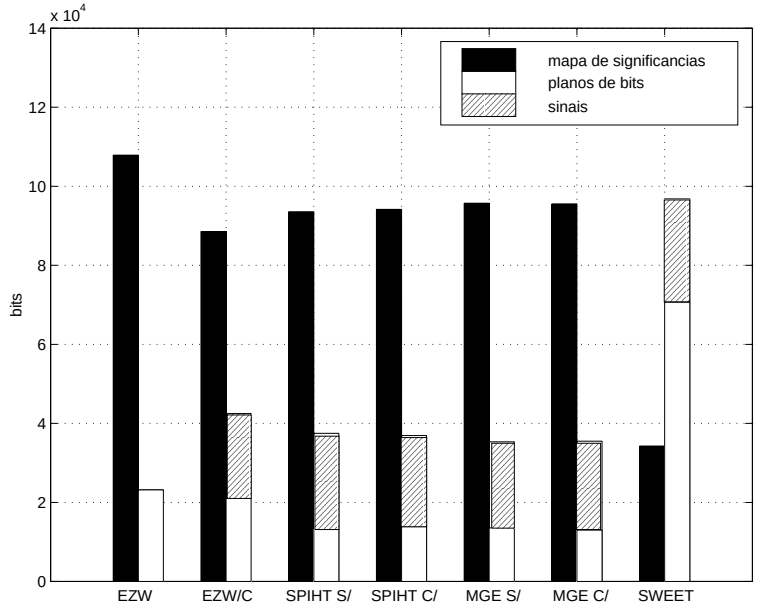


(f)

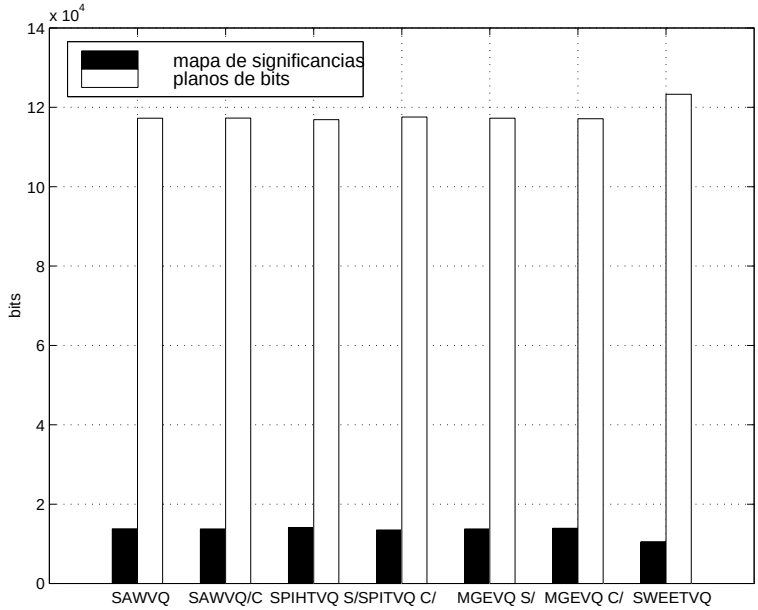


(d)

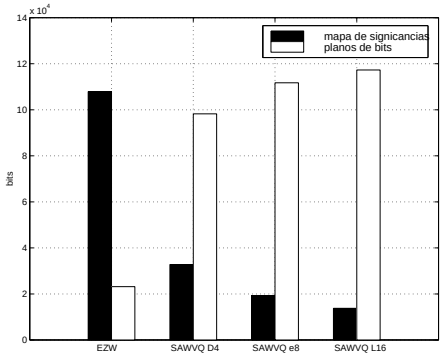
Figura 5.7: Para a imagem Lena 512×512 a 0,2 bits/pixel, comparação dos: (a) codificadores escalares; (b) codificadores vectoriais; (c) codificadores escalares e vectoriais EZW; (d) codificadores escalares e vectoriais EZW/C; (e) codificadores escalares e vectoriais SPIHT; (f) codificadores escalares e vectoriais MGE; (g) codificadores escalares e vectoriais SWEET


x 10^4
14
12
10
8
6
4
2
0
bits
mapa de significancias
planos de bits
sinais
EZW
EZW/C
SPIHT S/
SPIHT C/
MGE S/
MGE C/
SWEET

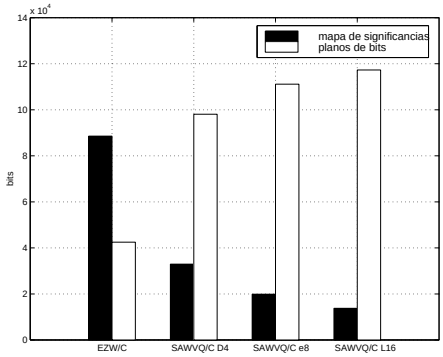
(a)


x 10^4
14
12
10
8
6
4
2
0
bits
mapa de significancias
planos de bits
SAWVQ
SAWVQ/C
SPIHTVQ S/
SPITVQ C/
MGEVQ S/
MGEVQ C/
SWEETVQ

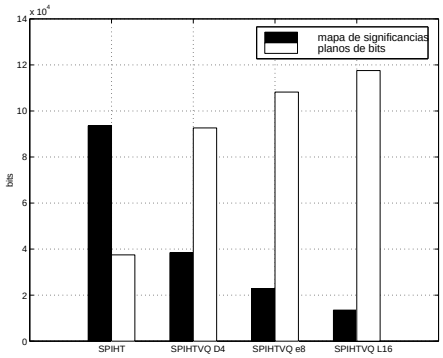
(b)



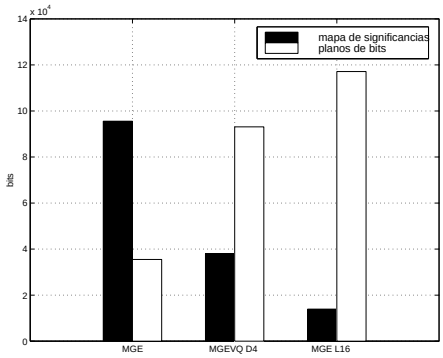
(c)



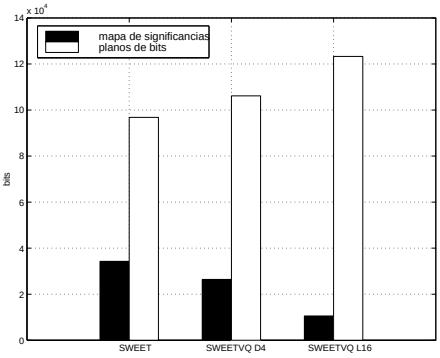
(d)



(e)

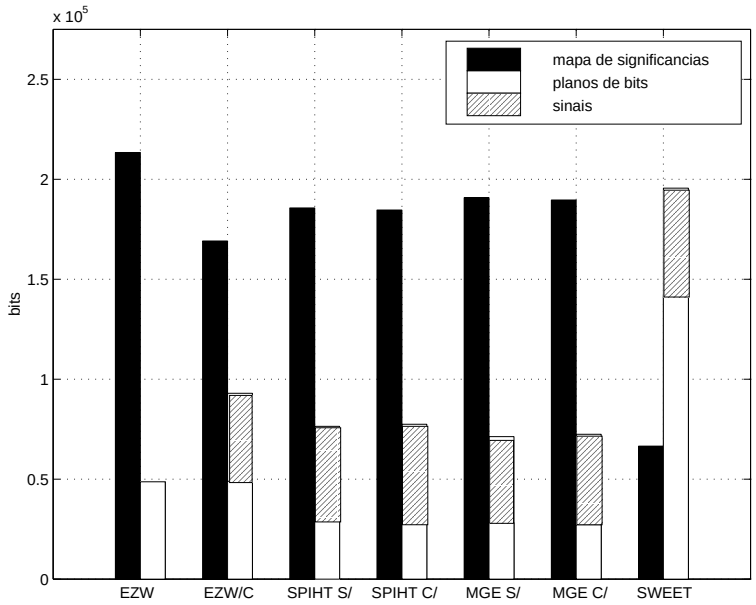


(f)

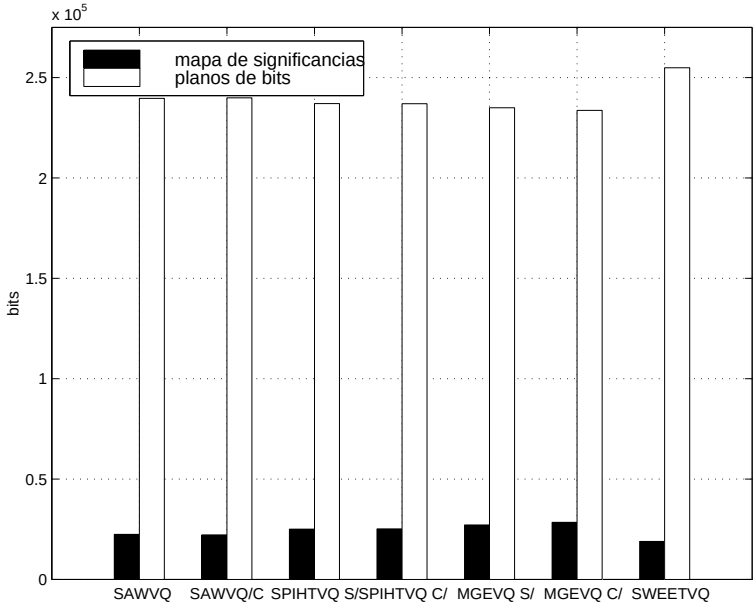


(d)

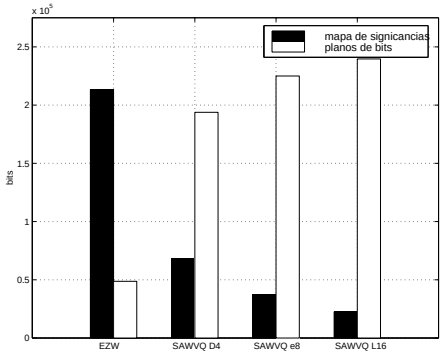
Figura 5.8: Para a imagem Lena 512×512 a 0,5 bits/pixel, comparação dos: (a) codificadores escalares; (b) codificadores vectoriais; (c) codificadores escalares e vectoriais EZW; (d) codificadores escalares e vectoriais EZW/C; (e) codificadores escalares e vectoriais SPIHT; (f) codificadores escalares e vectoriais MGE; (g) codificadores escalares e vectoriais SWEET


x 10^5
mapa de significancias
planos de bits
sinais
bits
2.5
2
1.5
1
0.5
0
EZW
EZW/C
SPIHT S/
SPIHT C/
MGE S/
MGE C/
SWEET

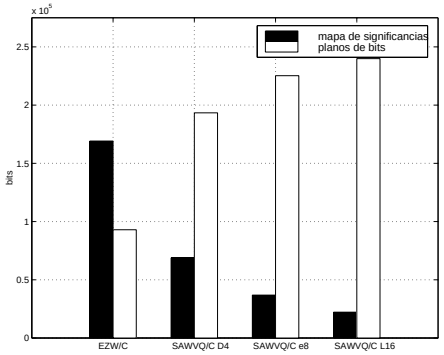
(a)


x 10^5
mapa de significancias
planos de bits
bits
2.5
2
1.5
1
0.5
0
SAWVQ
SAWVQ/C
SPIHTVQ S/
SPIHTVQ C/
MGEVQ S/
MGEVQ C/
SWEETVQ

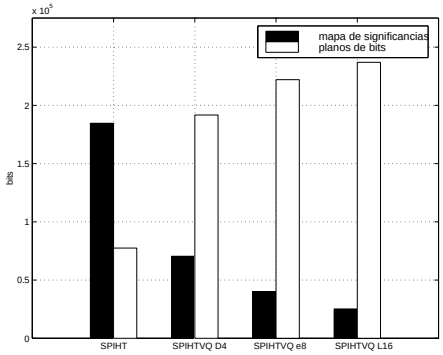
(b)



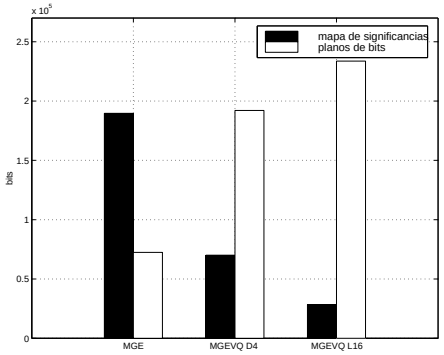
(c)



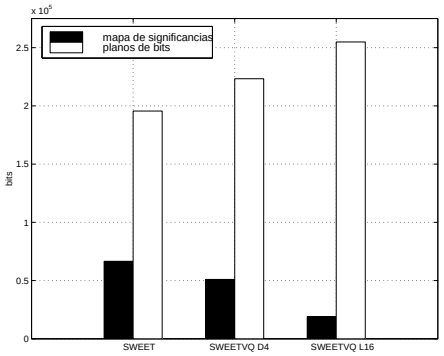
(d)



(e)

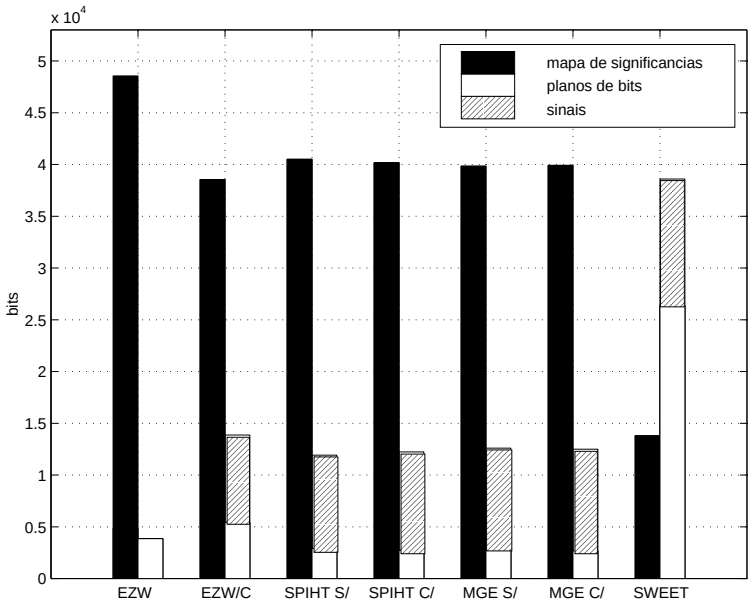


(f)

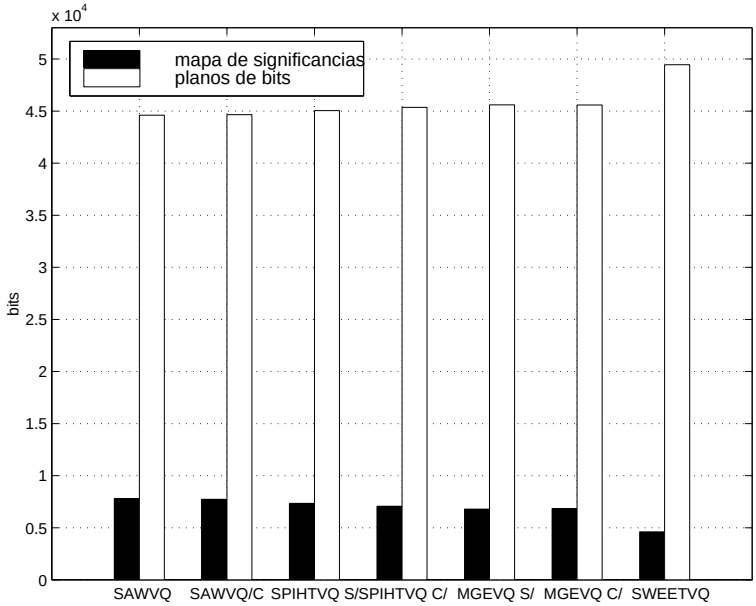


(d)

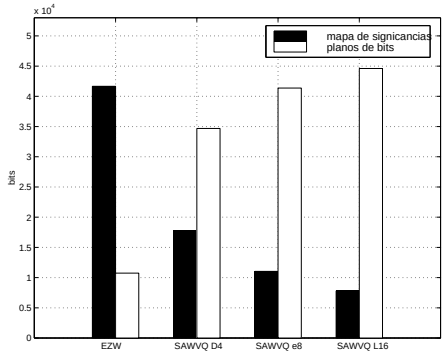
Figura 5.9: Para a imagem Lena 512×512 a 1 bits/pixel, comparação dos: (a) codificadores escalares; (b) codificadores vectoriais; (c) codificadores escalares e vectoriais EZW; (d) codificadores escalares e vectoriais EZW/C; (e) codificadores escalares e vectoriais SPIHT; (f) codificadores escalares e vectoriais MGE; (g) codificadores escalares e vectoriais SWEET


x 10^4
mapa de significancias
planos de bits
sinais
bits
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
EZW
EZW/C
SPIHT S/
SPIHT C/
MGE S/
MGE C/
SWEET

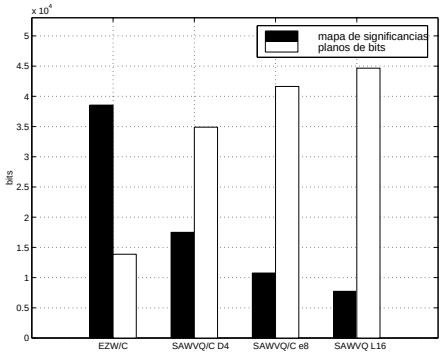
(a)


x 10^4
mapa de significancias
planos de bits
bits
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
SAWVQ
SAWVQ/C
SPIHTVQ S/
SPIHTVQ C/
MGEVQ S/
MGEVQ C/
SWEETVQ

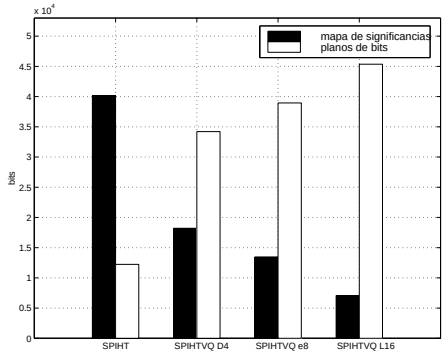
(b)



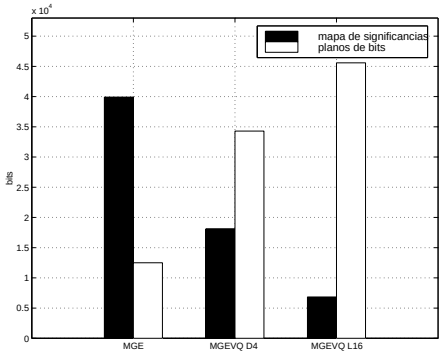
(c)



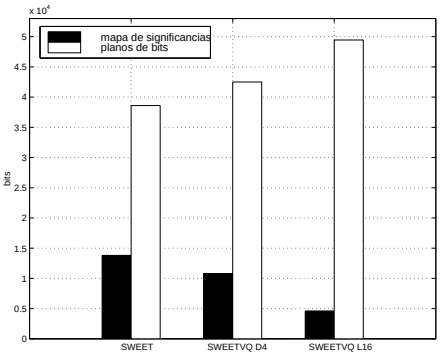
(d)



(e)

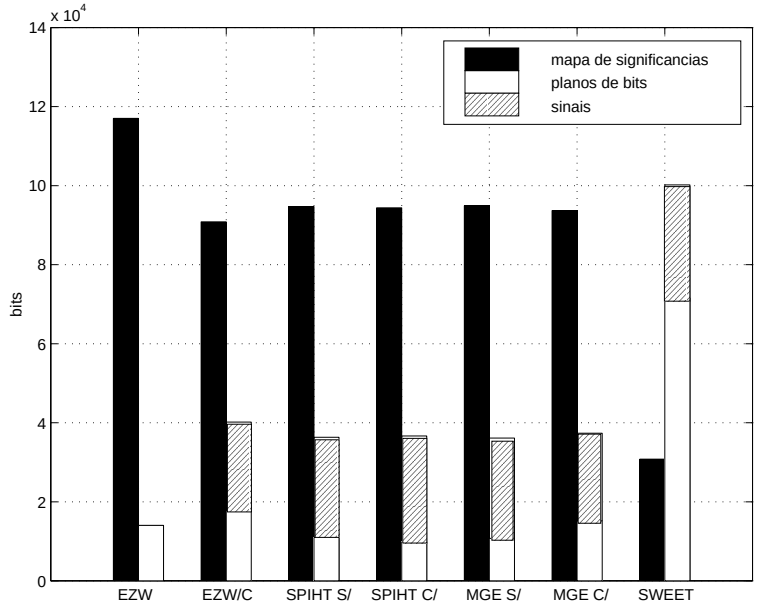


(f)

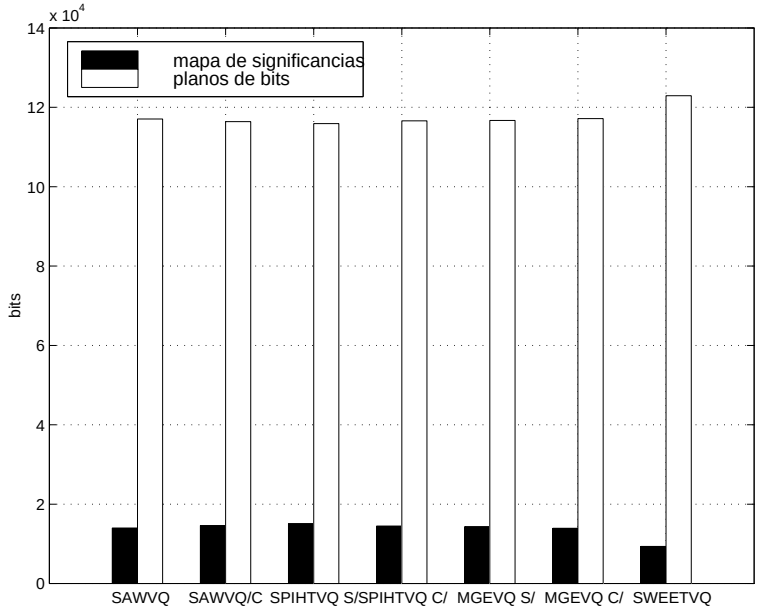


(d)

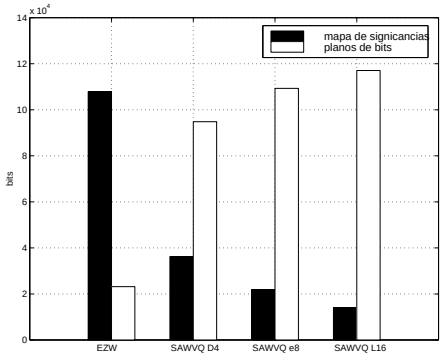
Figura 5.10: Para a imagem Barbara a 0,2 bits/pixel, comparação dos: (a) codificadores escalares; (b) codificadores vectoriais; (c) codificadores escalares e vectoriais EZW; (d) codificadores escalares e vectoriais EZW/C; (e) codificadores escalares e vectoriais SPIHT; (f) codificadores escalares e vectoriais MGE; (g) codificadores escalares e vectoriais SWEET


x 10^4
14
12
10
8
6
4
2
0
bits
mapa de significancias
planos de bits
sinais
EZW
EZW/C
SPIHT S/
SPIHT C/
MGE S/
MGE C/
SWEET

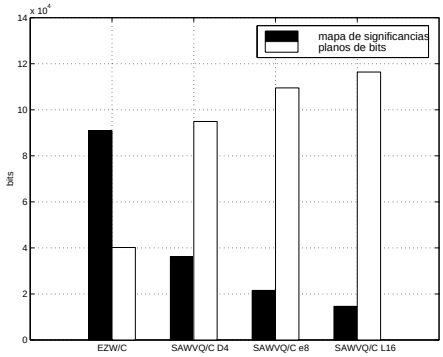
(a)


x 10^4
14
12
10
8
6
4
2
0
bits
mapa de significancias
planos de bits
SAWVQ
SAWVQ/C
SPIHTVQ S/
SPIHTVQ C/
MGEVQ S/
MGEVQ C/
SWEETVQ

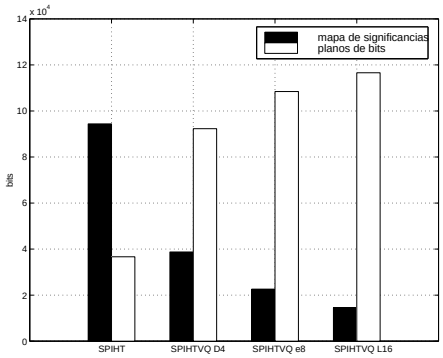
(b)



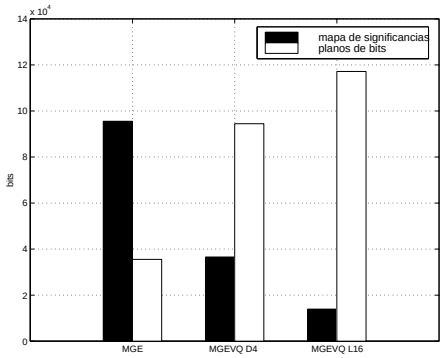
(c)



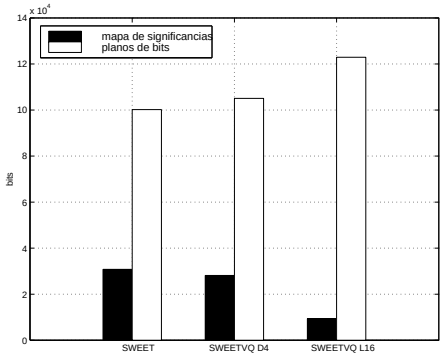
(d)



(e)

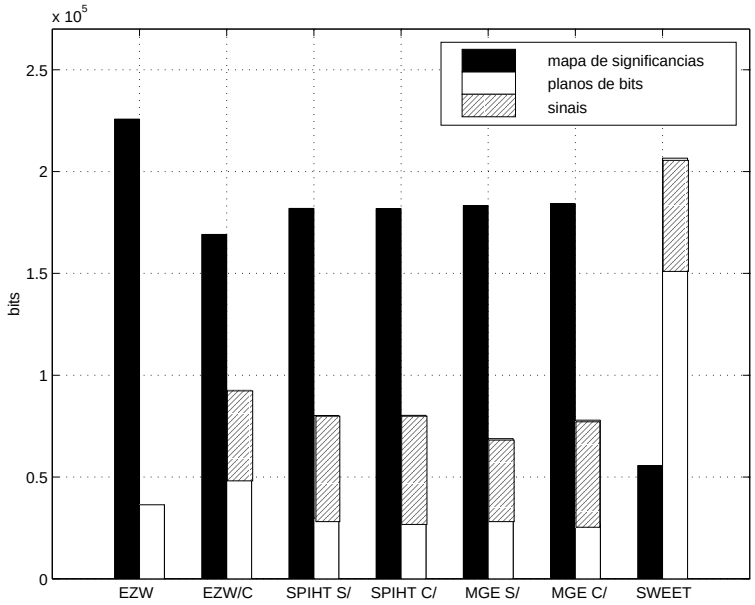


(f)

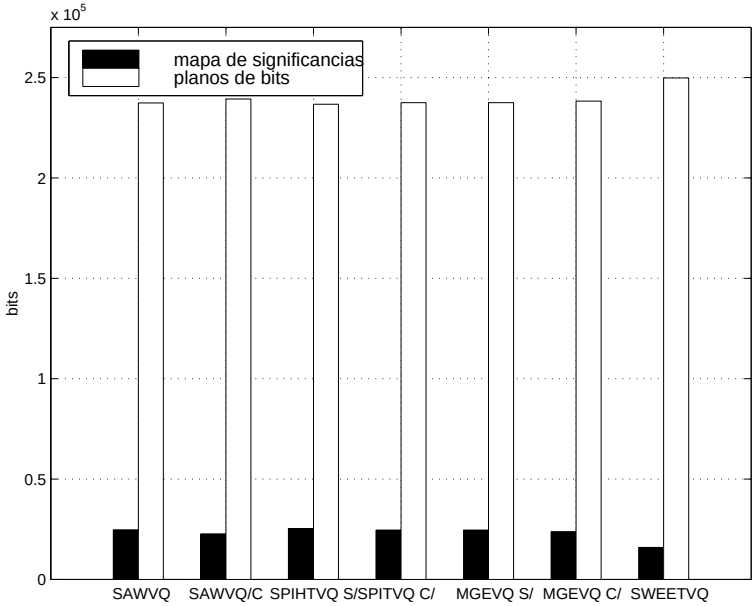


(d)

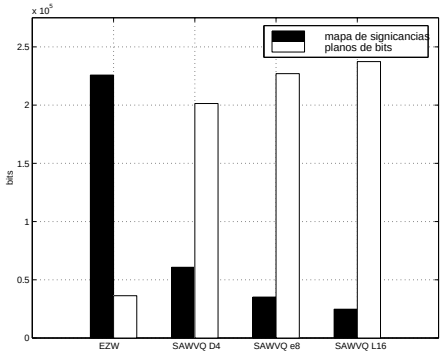
Figura 5.11: Para a imagem Barbara a 0,5 bits/pixel, comparação dos: (a) codificadores escalares; (b) codificadores vectoriais; (c) codificadores escalares e vectoriais EZW; (d) codificadores escalares e vectoriais EZW/C; (e) codificadores escalares e vectoriais SPIHT; (f) codificadores escalares e vectoriais MGE; (g) codificadores escalares e vectoriais SWEET


x 10^5
mapa de significancias
planos de bits
sinais
bits
2.5
2
1.5
1
0.5
0
EZW
EZW/C
SPIHT S/
SPIHT C/
MGE S/
MGE C/
SWEET

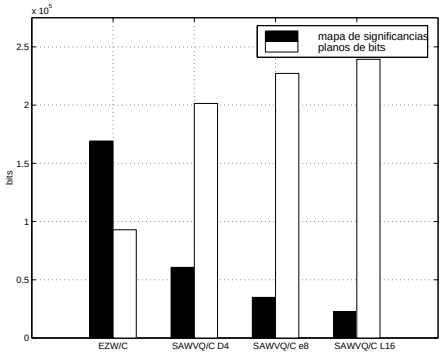
(a)


x 10^5
mapa de significancias
planos de bits
bits
2.5
2
1.5
1
0.5
0
SAWVQ
SAWVQ/C
SPIHTVQ S/
SPITVQ C/
MGEVQ S/
MGEVQ C/
SWEETVQ

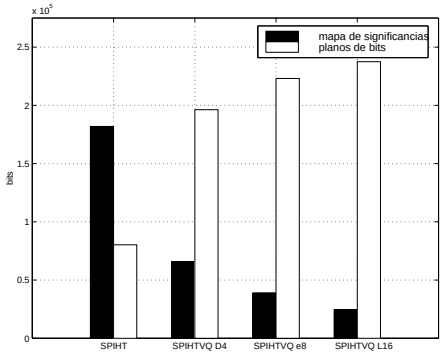
(b)



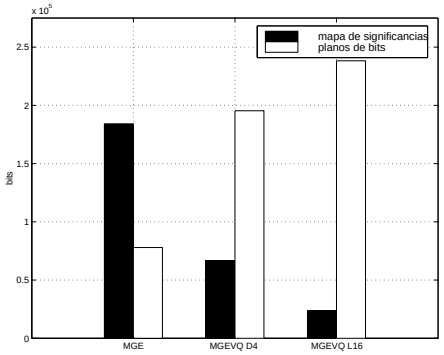
(c)



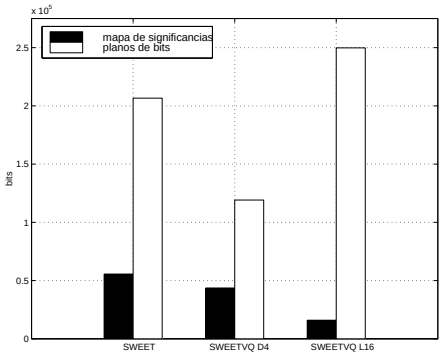
(d)



(e)

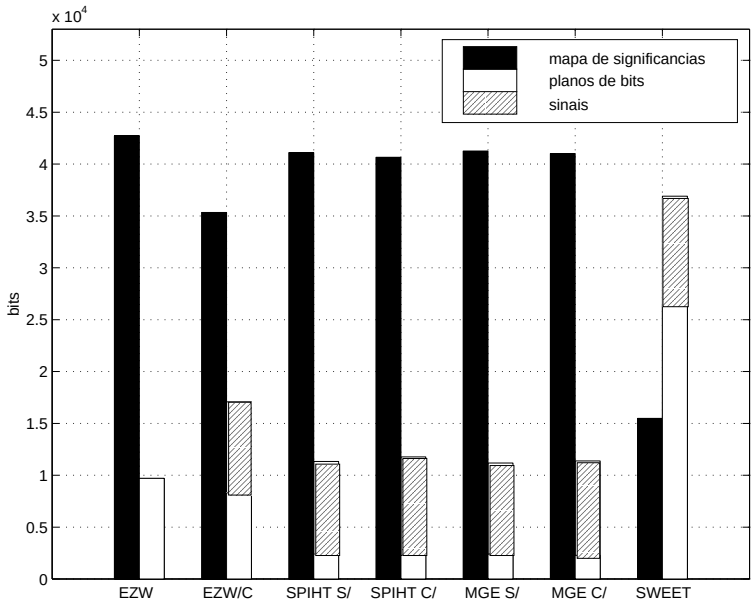


(f)

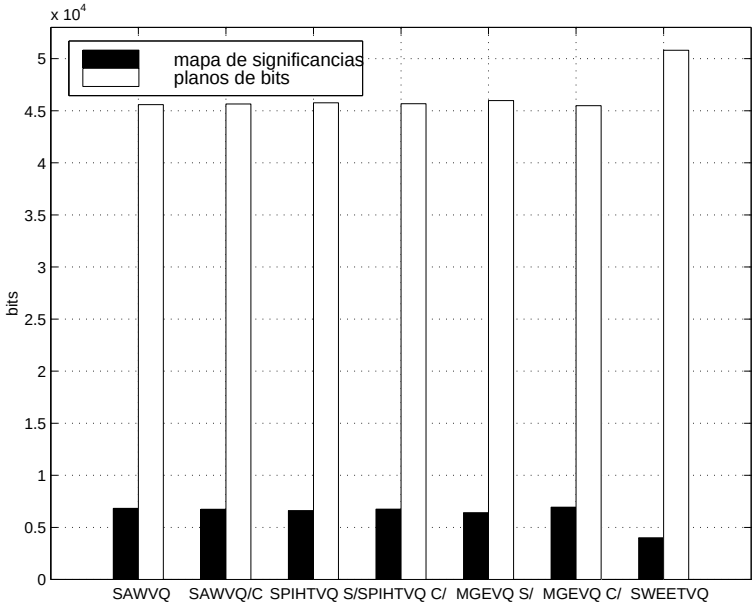


(d)

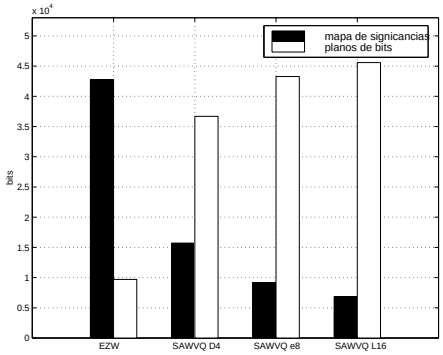
Figura 5.12: Para a imagem Barbara a 1 bits/pixel, comparação dos: (a) codificadores escalares; (b) codificadores vectoriais; (c) codificadores escalares e vectoriais EZW; (d) codificadores escalares e vectoriais EZW/C; (e) codificadores escalares e vectoriais SPIHT; (f) codificadores escalares e vectoriais MGE; (g) codificadores escalares e vectoriais SWEET


x 10^4
mapa de significancias
planos de bits
sinais
bits
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
EZW
EZW/C
SPIHT S/
SPIHT C/
MGE S/
MGE C/
SWEET

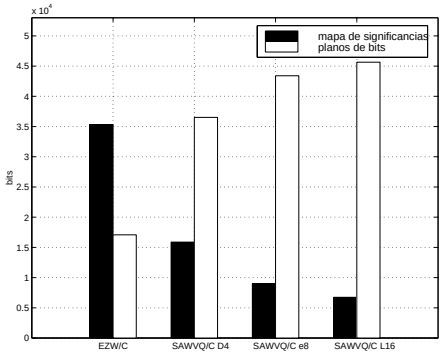
(a)


x 10^4
mapa de significancias
planos de bits
bits
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
SAWVQ
SAWVQ/C
SPIHTVQ S/
SPIHTVQ C/
MGEVQ S/
MGEVQ C/
SWEETVQ

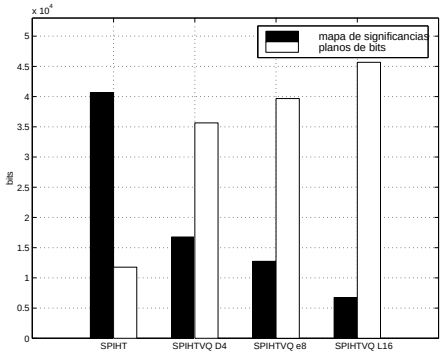
(b)



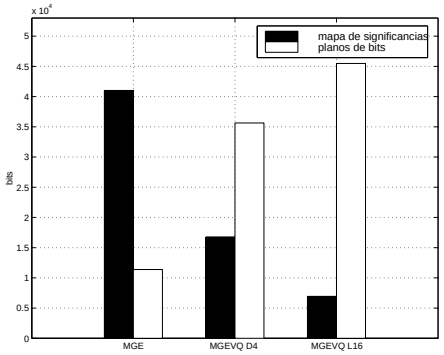
(c)



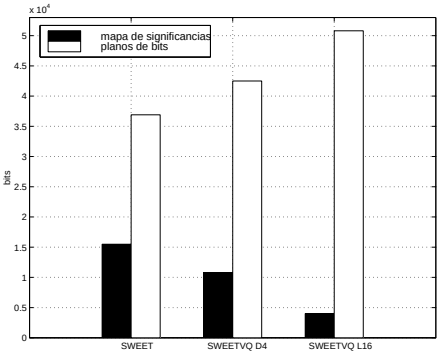
(d)



(e)

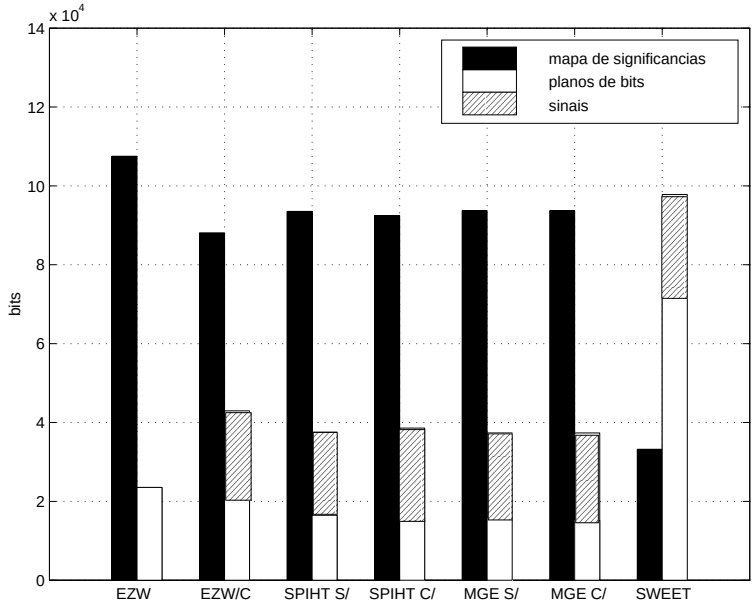


(f)

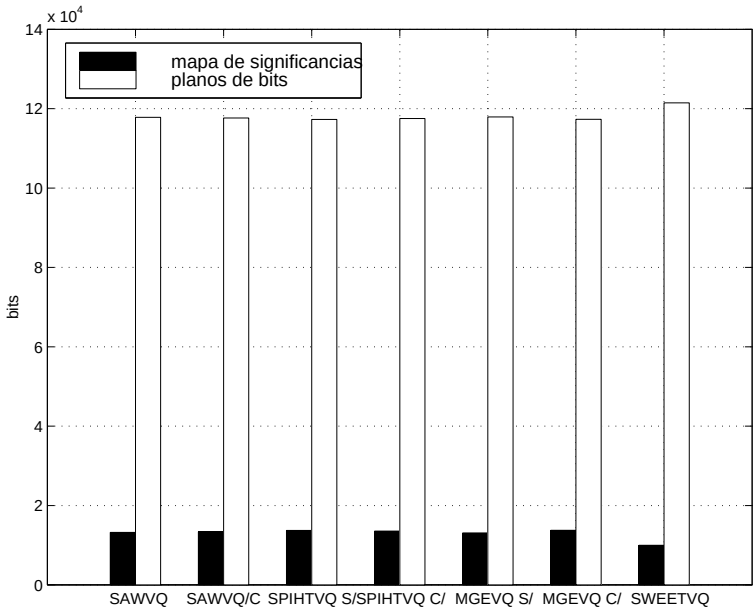


(d)

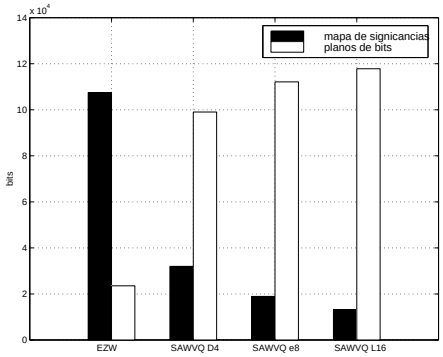
Figura 5.13: Para a imagem Boats a 0,2 bits/pixel, comparação dos: (a) codificadores escalares; (b) codificadores vectoriais; (c) codificadores escalares e vectoriais EZW; (d) codificadores escalares e vectoriais EZW/C; (e) codificadores escalares e vectoriais SPIHT; (f) codificadores escalares e vectoriais MGE; (g) codificadores escalares e vectoriais SWEET


x 10^4
14
12
10
8
6
4
2
0
bits
mapa de significancias
planos de bits
sinais
EZW
EZW/C
SPIHT S/
SPIHT C/
MGE S/
MGE C/
SWEET

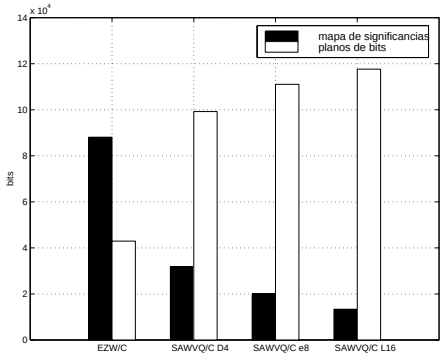
(a)


x 10^4
14
12
10
8
6
4
2
0
bits
mapa de significancias
planos de bits
SAWVQ
SAWVQ/C
SPIHTVQ S/
SPIHTVQ C/
MGEVQ S/
MGEVQ C/
SWEETVQ

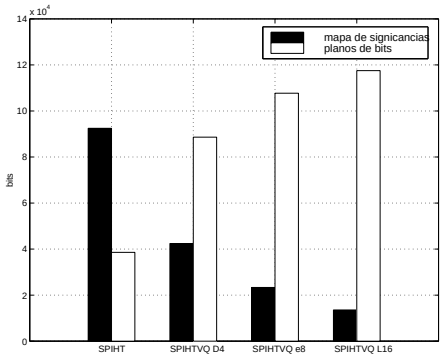
(b)



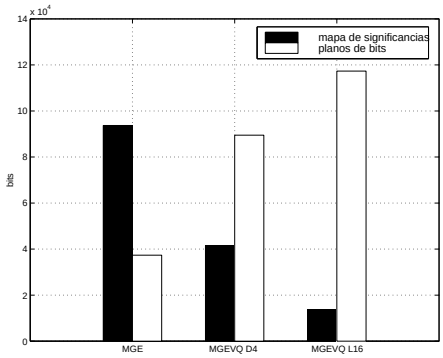
(c)



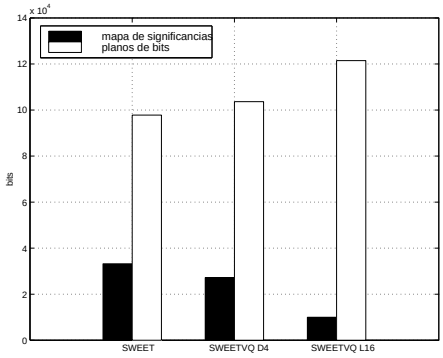
(d)



(e)

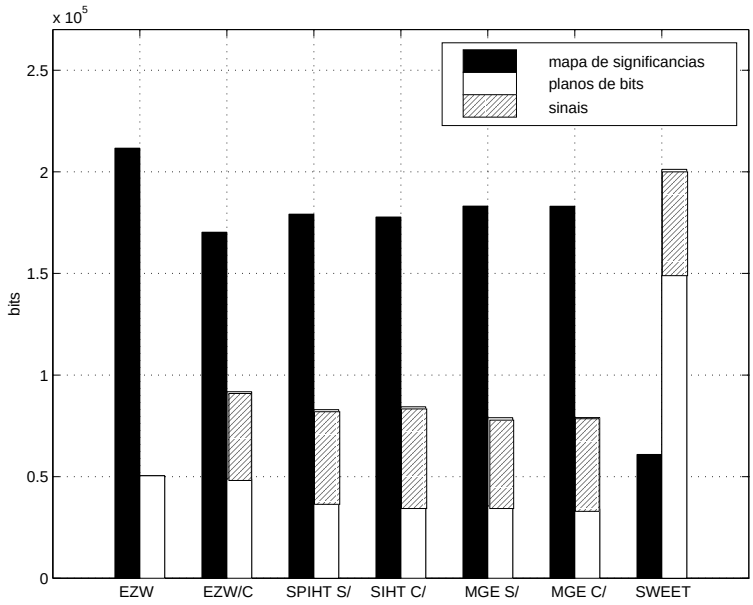


(f)

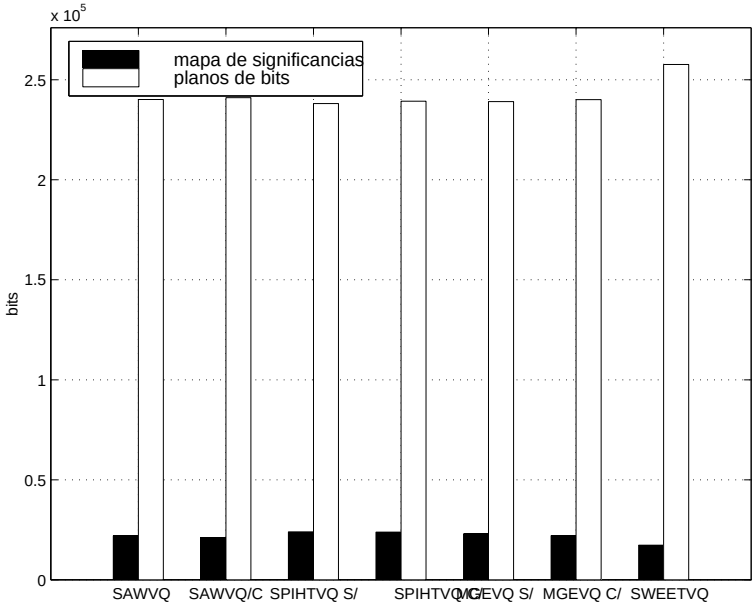


(d)

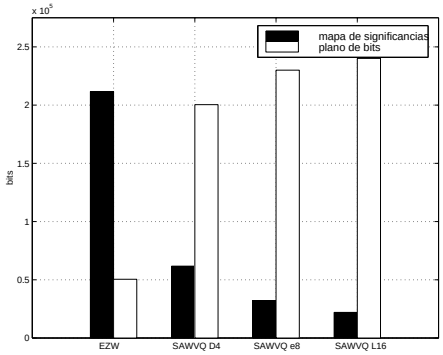
Figura 5.14: Para a imagem Boats a 0,5 bits/pixel, comparação dos: (a) codificadores escalares; (b) codificadores vectoriais; (c) codificadores escalares e vectoriais EZW; (d) codificadores escalares e vectoriais EZW/C; (e) codificadores escalares e vectoriais SPIHT; (f) codificadores escalares e vectoriais MGE; (g) codificadores escalares e vectoriais SWEET


x 10^5
mapa de significancias
planos de bits
sinais
bits
2.5
2
1.5
1
0.5
0
EZW
EZW/C
SPIHT S/
SIHT C/
MGE S/
MGE C/
SWEET

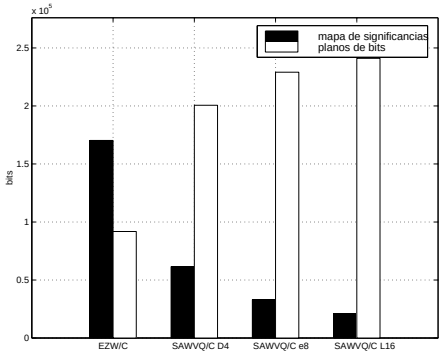
(a)


x 10^5
mapa de significancias
planos de bits
bits
2.5
2
1.5
1
0.5
0
SAWVQ
SAWVQ/C
SPIHTVQ S/
SPIHTVQ C/
MGEVQ S/
MGEVQ C/
SWEETVQ

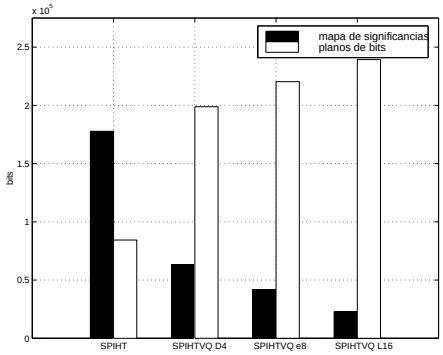
(b)



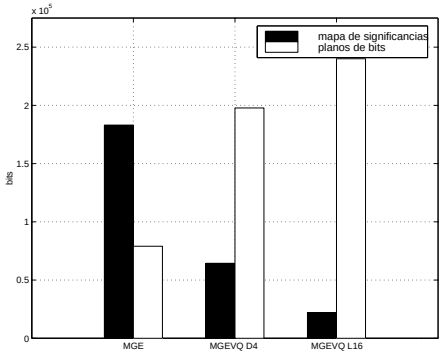
(c)



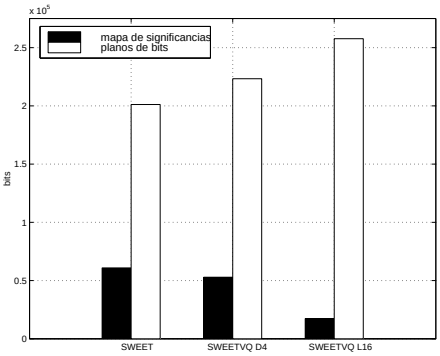
(d)



(e)

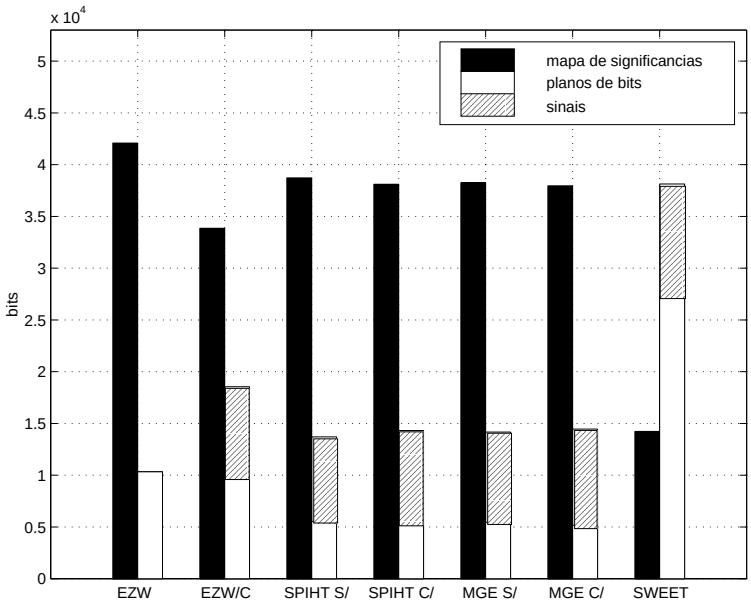


(f)

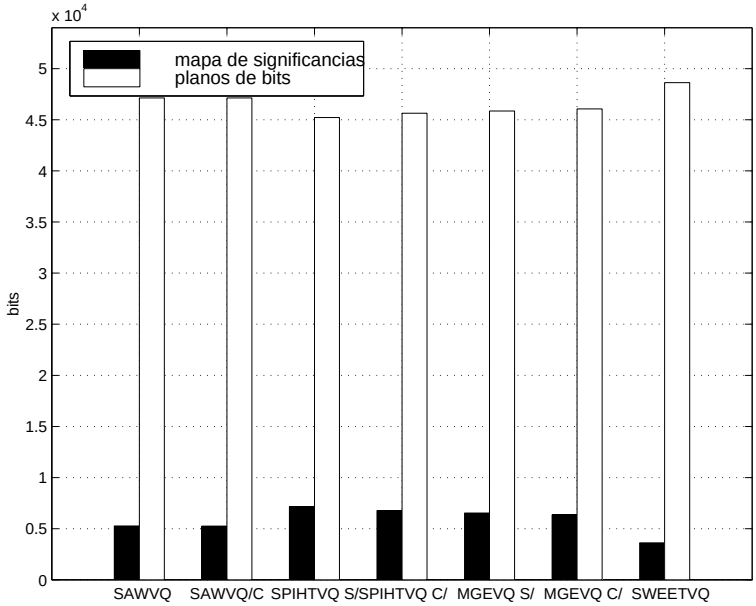


(d)

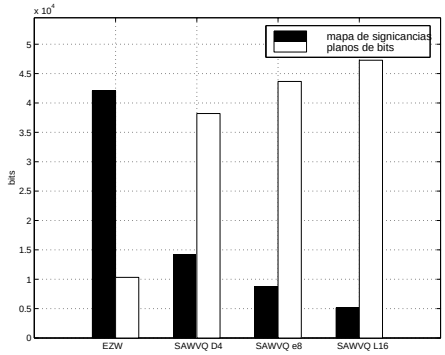
Figura 5.15: Para a imagem Boats a 1 bits/pixel, comparação dos: (a) codificadores escalares; (b) codificadores vectoriais; (c) codificadores escalares e vectoriais EZW; (d) codificadores escalares e vectoriais EZW/C; (e) codificadores escalares e vectoriais SPIHT; (f) codificadores escalares e vectoriais MGE; (g) codificadores escalares e vectoriais SWEET


x 10^4
mapa de significancias
planos de bits
sinais
bits
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
EZW
EZW/C
SPIHT S/
SPIHT C/
MGE S/
MGE C/
SWEET

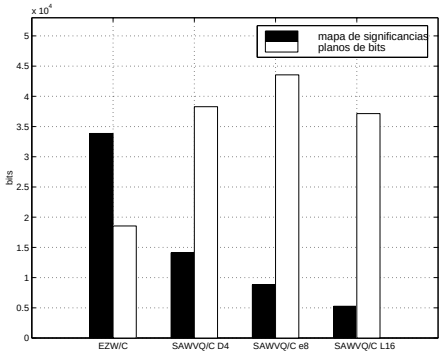
(a)


x 10^4
mapa de significancias
planos de bits
bits
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
SAWVQ
SAWVQ/C
SPIHTVQ S/
SPIHTVQ C/
MGEVQ S/
MGEVQ C/
SWEETVQ

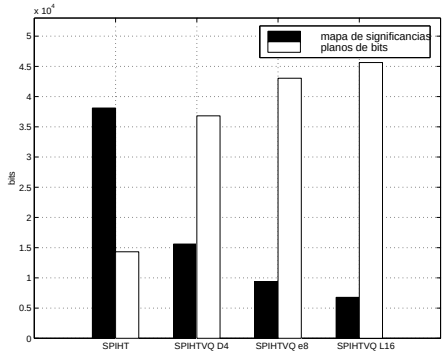
(b)



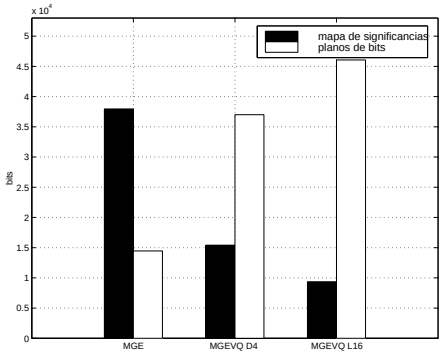
(c)



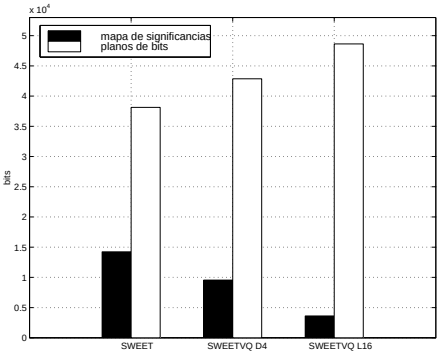
(d)



(e)

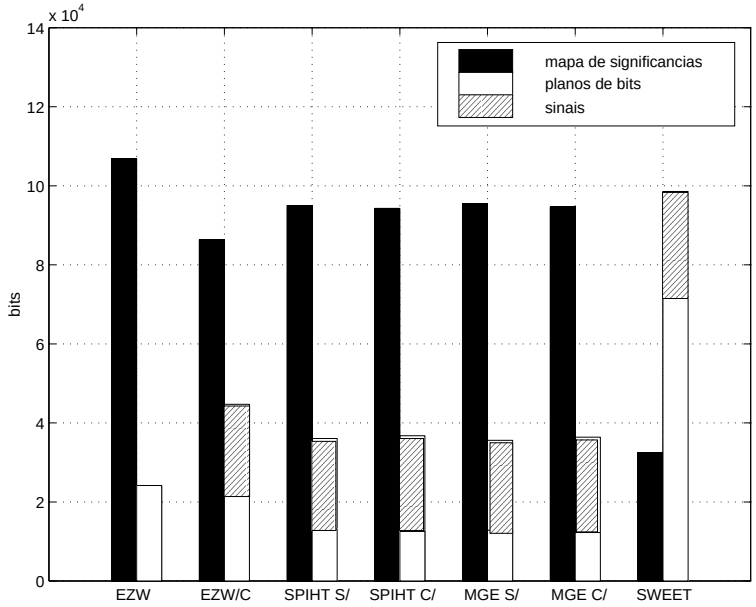


(f)

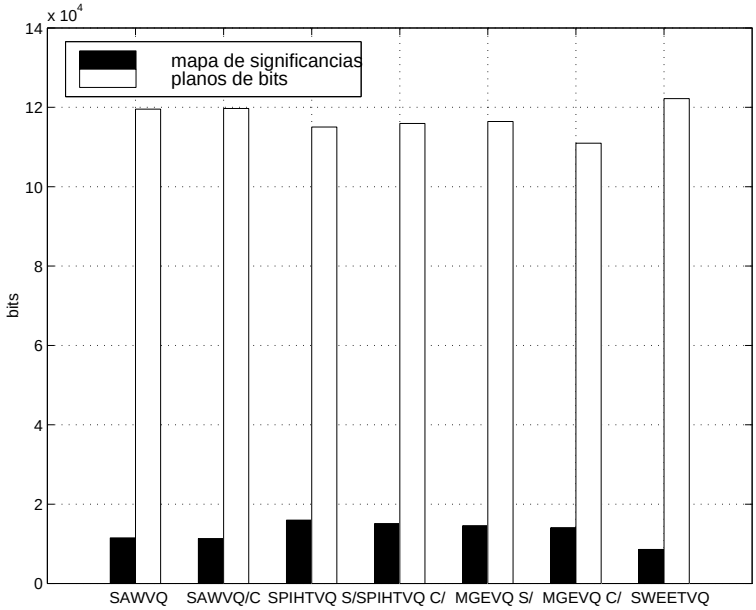


(d)

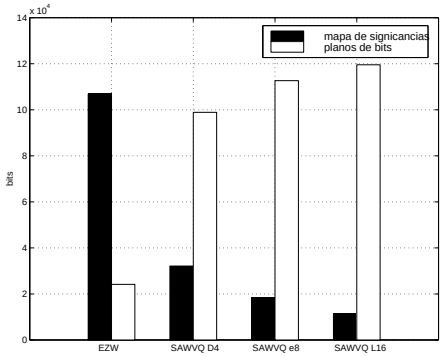
Figura 5.16: Para a imagem Girl a 0,2 bits/pixel, comparação dos: (a) codificadores escalares; (b) codificadores vectoriais; (c) codificadores escalares e vectoriais EZW; (d) codificadores escalares e vectoriais EZW/C; (e) codificadores escalares e vectoriais SPIHT; (f) codificadores escalares e vectoriais MGE; (g) codificadores escalares e vectoriais SWEET


x 10^4
14
12
10
8
6
4
2
0
bits
mapa de significancias
planos de bits
sinais
EZW
EZW/C
SPIHT S/
SPIHT C/
MGE S/
MGE C/
SWEET

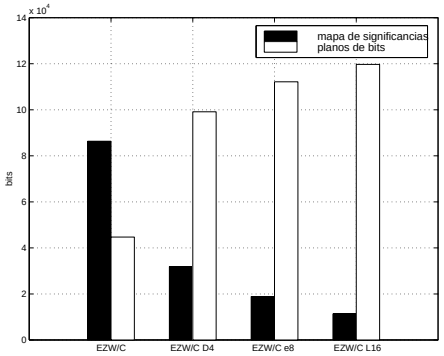
(a)


x 10^4
14
12
10
8
6
4
2
0
bits
mapa de significancias
planos de bits
SAWVQ
SAWVQ/C
SPIHTVQ S/
SPIHTVQ C/
MGEVQ S/
MGEVQ C/
SWEETVQ

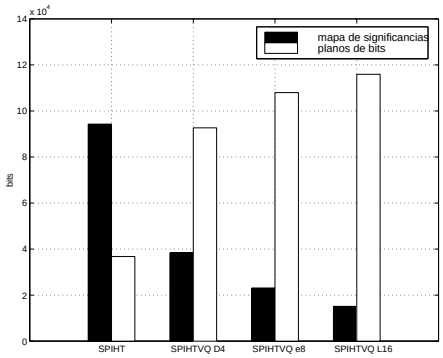
(b)



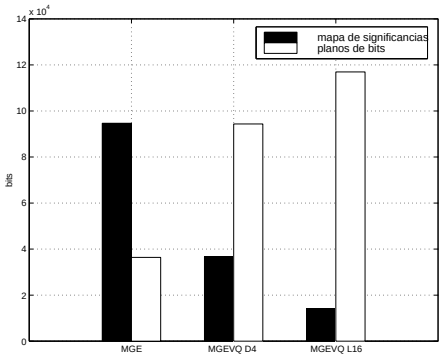
(c)



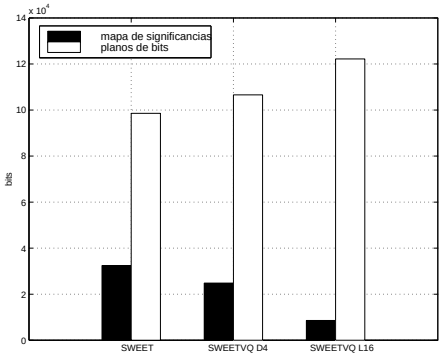
(d)



(e)

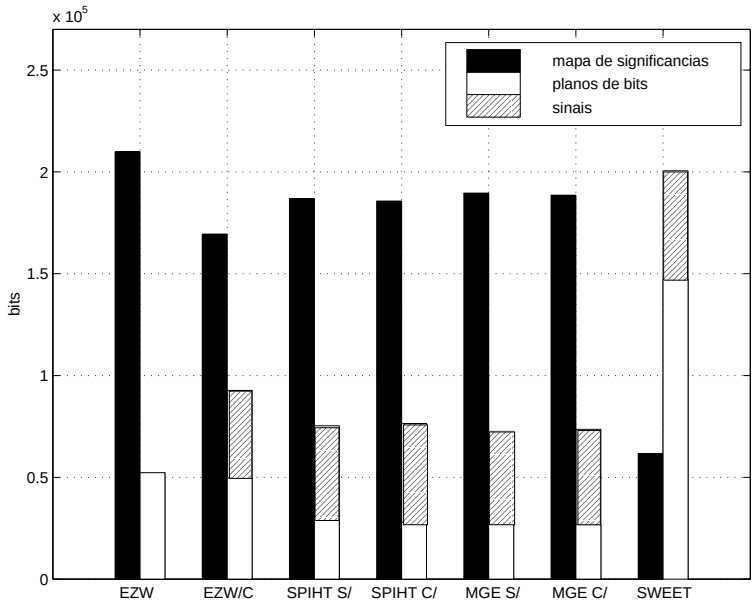


(f)

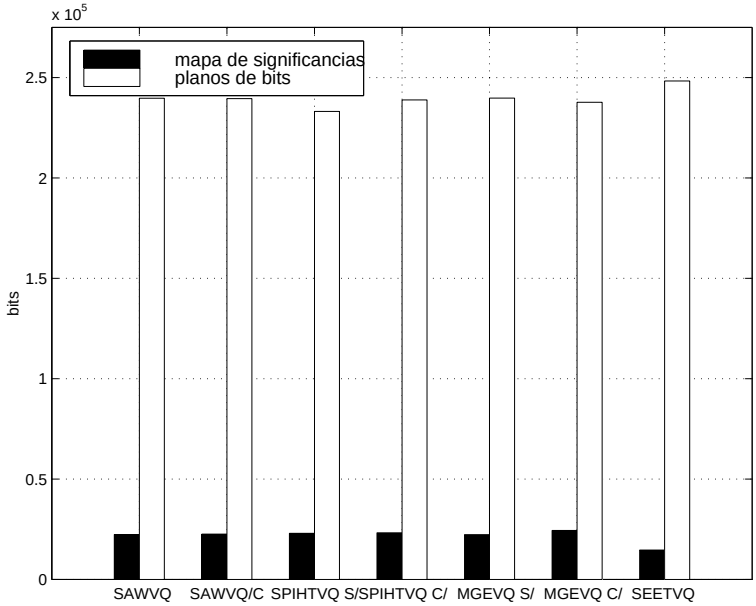


(d)

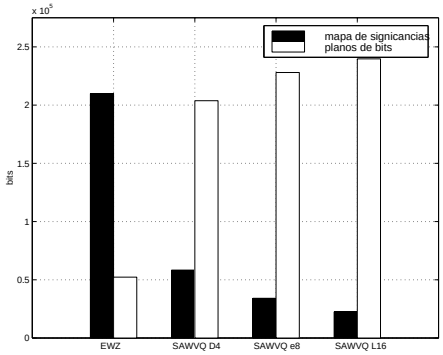
Figura 5.17: Para a imagem Girl a 0,5 bits/pixel, comparação dos: (a) codificadores escalares; (b) codificadores vectoriais; (c) codificadores escalares e vectoriais EZW; (d) codificadores escalares e vectoriais EZW/C; (e) codificadores escalares e vectoriais SPIHT; (f) codificadores escalares e vectoriais MGE; (g) codificadores escalares e vectoriais SWEET


x 10^5
2.5
2
1.5
1
0.5
0
bits
mapa de significancias
planos de bits
sinais
EZW
EZW/C
SPIHT S/
SPIHT C/
MGE S/
MGE C/
SWEET

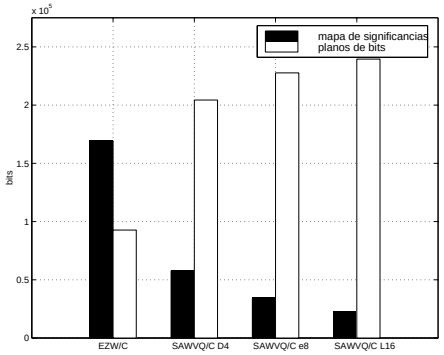
(a)


x 10^5
2.5
2
1.5
1
0.5
0
bits
mapa de significancias
planos de bits
SAWVQ
SAWVQ/C
SPIHTVQ S/
SPIHTVQ C/
MGEVQ S/
MGEVQ C/
SEETVQ

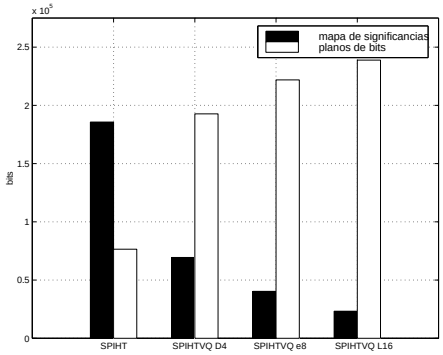
(b)



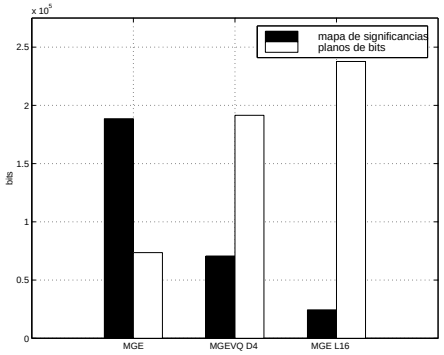
(c)



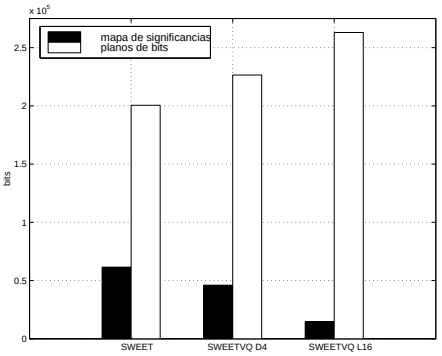
(d)



(e)

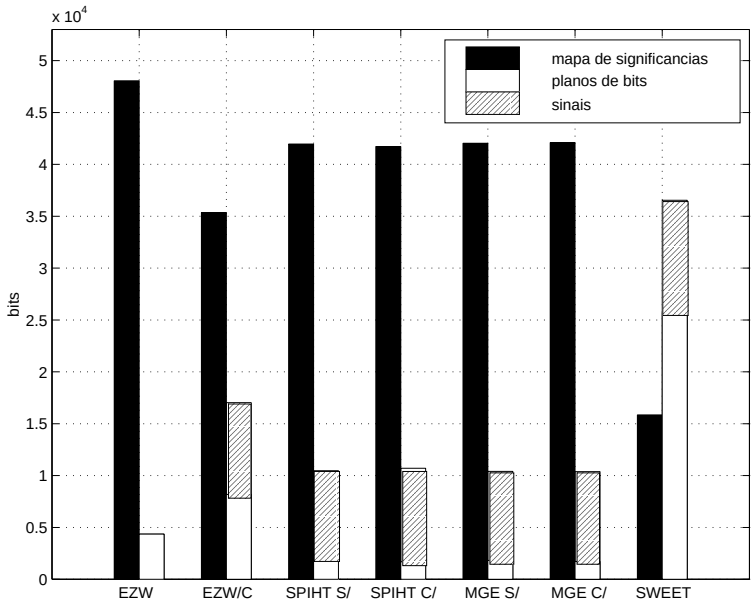


(f)

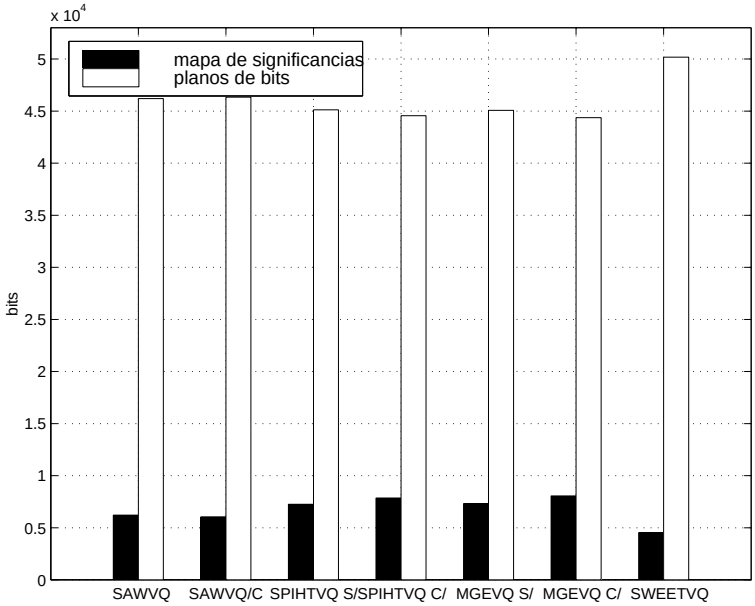


(d)

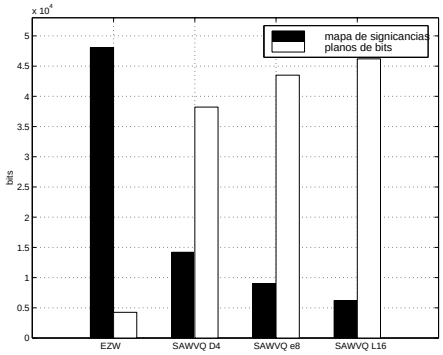
Figura 5.18: Para a imagem Girl a 1 bits/pixel, comparação dos: (a) codificadores escalares; (b) codificadores vectoriais; (c) codificadores escalares e vectoriais EZW; (d) codificadores escalares e vectoriais EZW/C; (e) codificadores escalares e vectoriais SPIHT; (f) codificadores escalares e vectoriais MGE; (g) codificadores escalares e vectoriais SWEET


x 10^4
bits
mapa de significancias
planos de bits
sinais
EZW
EZW/C
SPIHT S/
SPIHT C/
MGE S/
MGE C/
SWEET

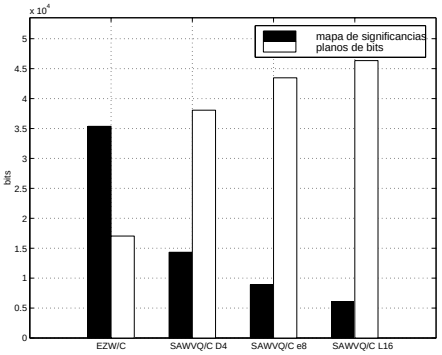
(a)


x 10^4
bits
mapa de significancias
planos de bits
SAWVQ
SAWVQ/C
SPIHTVQ S/
SPIHTVQ C/
MGEVQ S/
MGEVQ C/
SWEETVQ

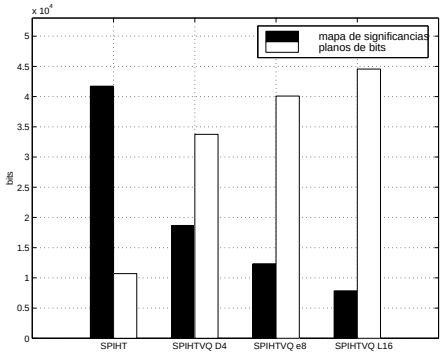
(b)



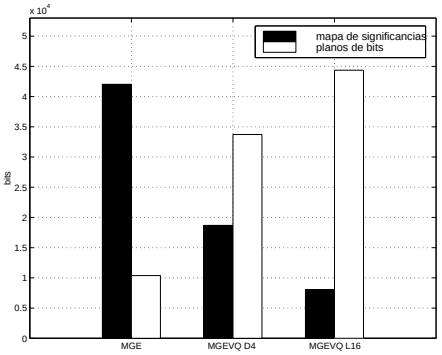
(c)



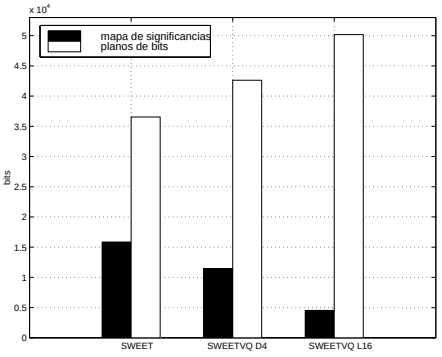
(d)



(e)

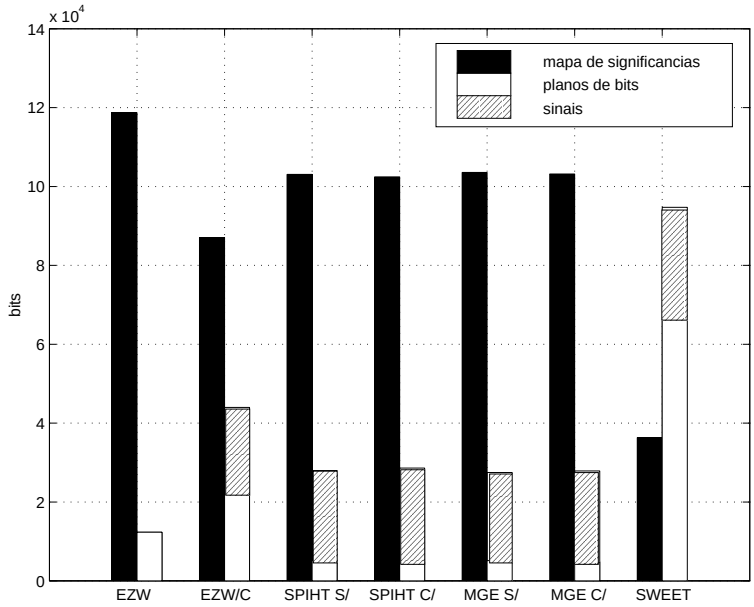


(f)

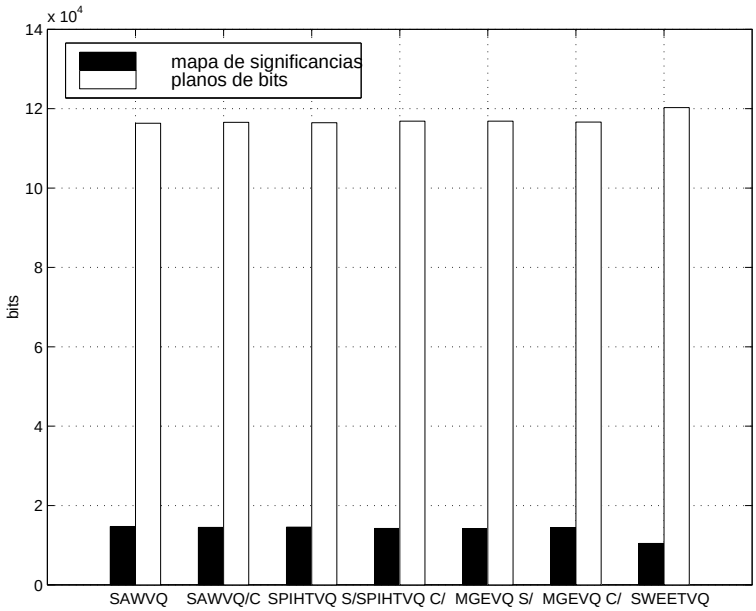


(d)

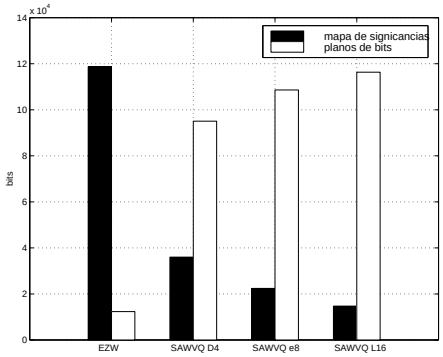
Figura 5.19: Para a imagem Gold a 0,2 bits/pixel, comparação dos: (a) codificadores escalares; (b) codificadores vectoriais; (c) codificadores escalares e vectoriais EZW; (d) codificadores escalares e vectoriais EZW/C; (e) codificadores escalares e vectoriais SPIHT; (f) codificadores escalares e vectoriais MGE; (g) codificadores escalares e vectoriais SWEET


x 10^4
14
12
10
8
6
4
2
0
bits
mapa de significancias
planos de bits
sinais
EZW
EZW/C
SPIHT S/
SPIHT C/
MGE S/
MGE C/
SWEET

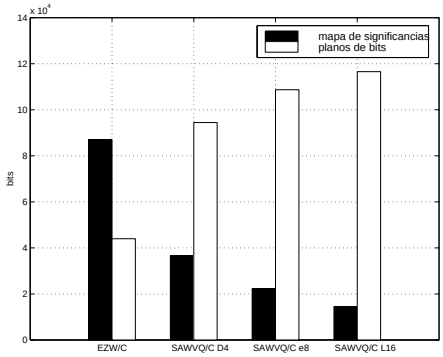
(a)


x 10^4
14
12
10
8
6
4
2
0
bits
mapa de significancias
planos de bits
SAWVQ
SAWVQ/C
SPIHTVQ S/
SPIHTVQ C/
MGEVQ S/
MGEVQ C/
SWEETVQ

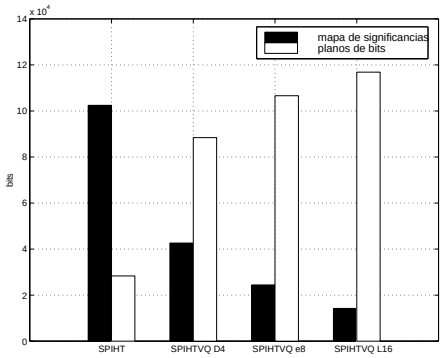
(b)



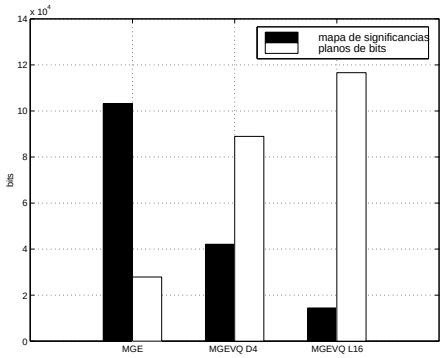
(c)



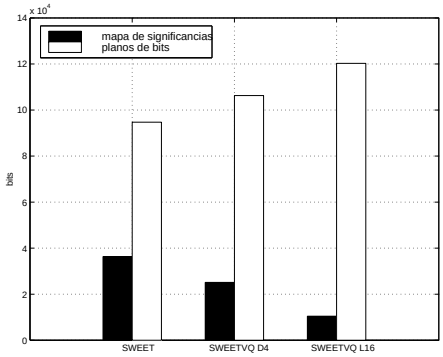
(d)



(e)

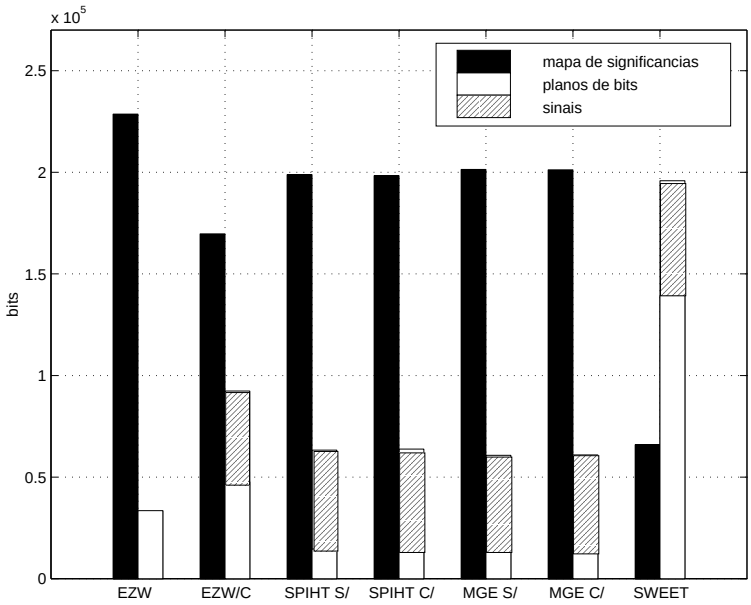


(f)

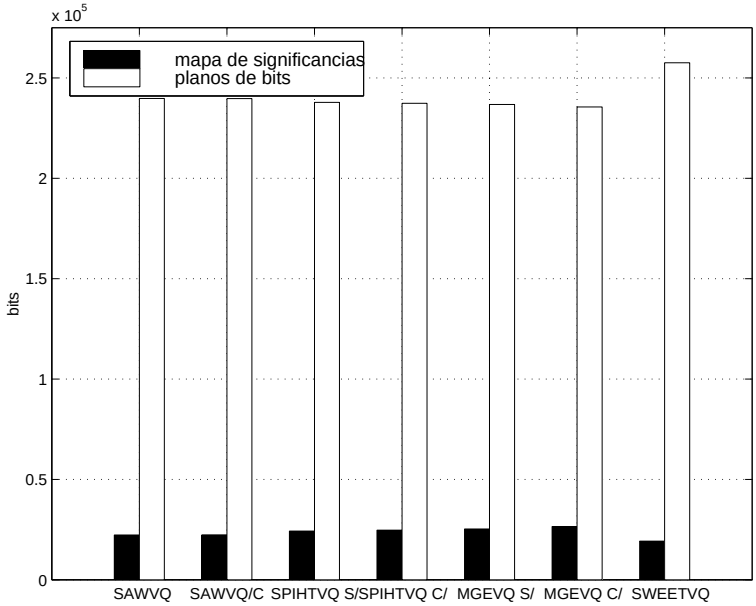


(d)

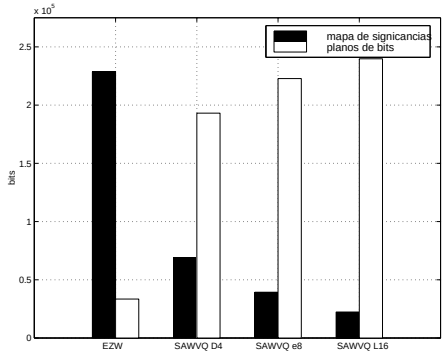
Figura 5.20: Para a imagem Gold a 0,5 bits/pixel, comparação dos: (a) codificadores escalares; (b) codificadores vectoriais; (c) codificadores escalares e vectoriais EZW; (d) codificadores escalares e vectoriais EZW/C; (e) codificadores escalares e vectoriais SPIHT; (f) codificadores escalares e vectoriais MGE; (g) codificadores escalares e vectoriais SWEET


x 10^5
2.5
2
1.5
1
0.5
0
bits
mapa de significancias
planos de bits
sinais
EZW
EZW/C
SPIHT S/
SPIHT C/
MGE S/
MGE C/
SWEET

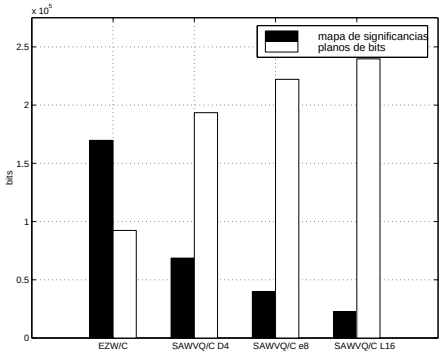
(a)


x 10^5
2.5
2
1.5
1
0.5
0
bits
mapa de significancias
planos de bits
SAWVQ
SAWVQ/C
SPIHTVQ S/
SPIHTVQ C/
MGEVQ S/
MGEVQ C/
SWEETVQ

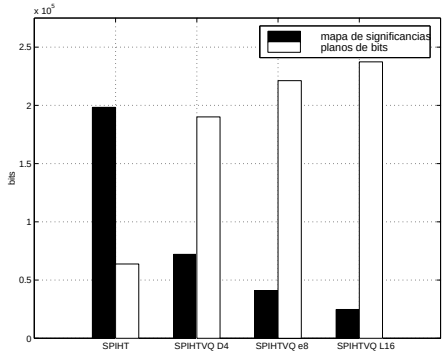
(b)



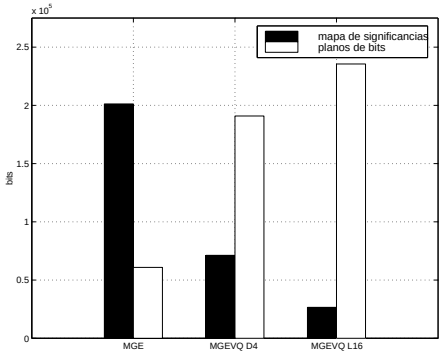
(c)



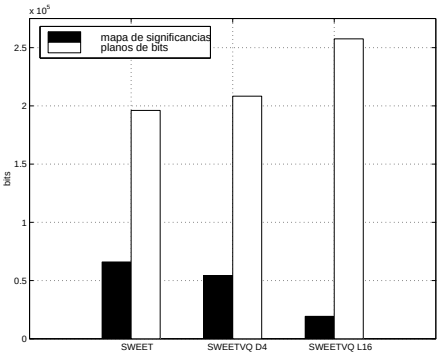
(d)



(e)

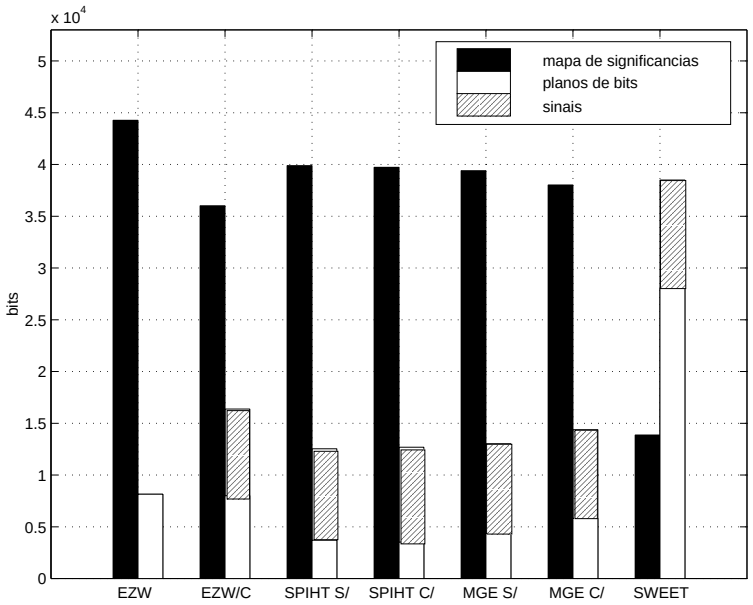


(f)

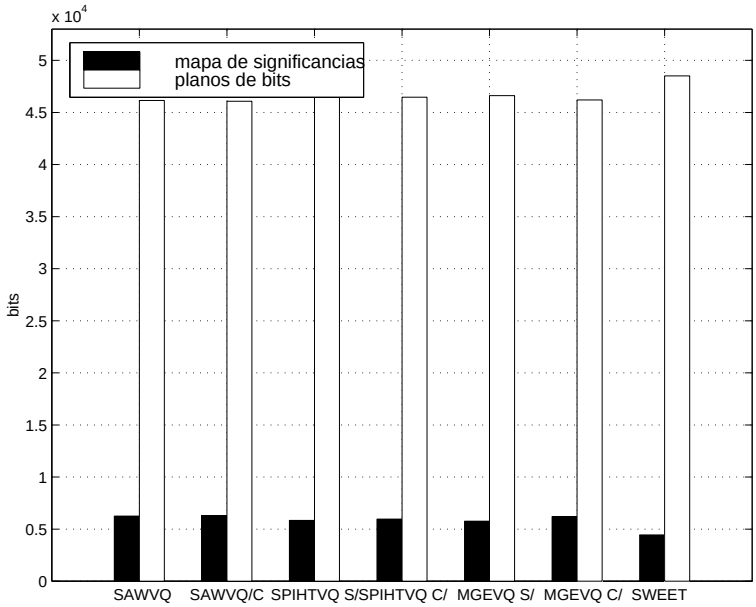


(d)

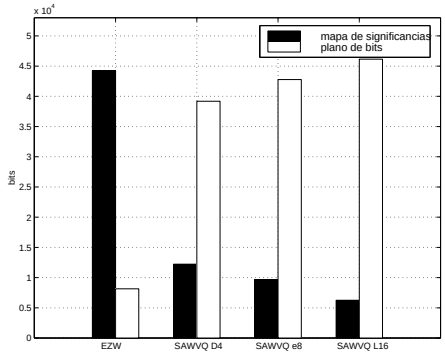
Figura 5.21: Para a imagem Gold a 1 bits/pixel, comparação dos: (a) codificadores escalares; (b) codificadores vectoriais; (c) codificadores escalares e vectoriais EZW; (d) codificadores escalares e vectoriais EZW/C; (e) codificadores escalares e vectoriais SPIHT; (f) codificadores escalares e vectoriais MGE; (g) codificadores escalares e vectoriais SWEET


x 10^4
mapa de significancias
planos de bits
sinais
bits
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
EZW
EZW/C
SPIHT S/
SPIHT C/
MGE S/
MGE C/
SWEET

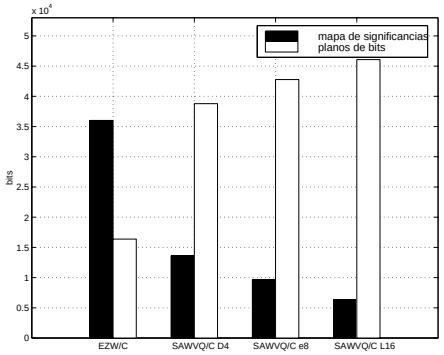
(a)


x 10^4
mapa de significancias
planos de bits
bits
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
SAWVQ
SAWVQ/C
SPIHTVQ S/
SPIHTVQ C/
MGEVQ S/
MGEVQ C/
SWEET

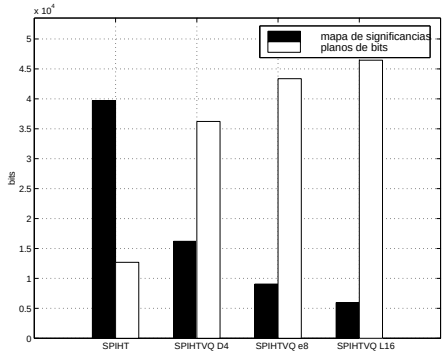
(b)



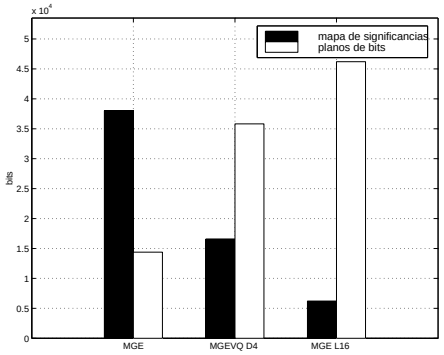
(c)



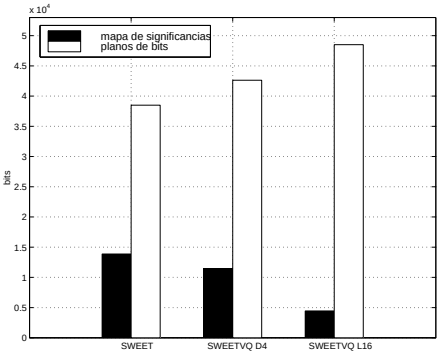
(d)



(e)

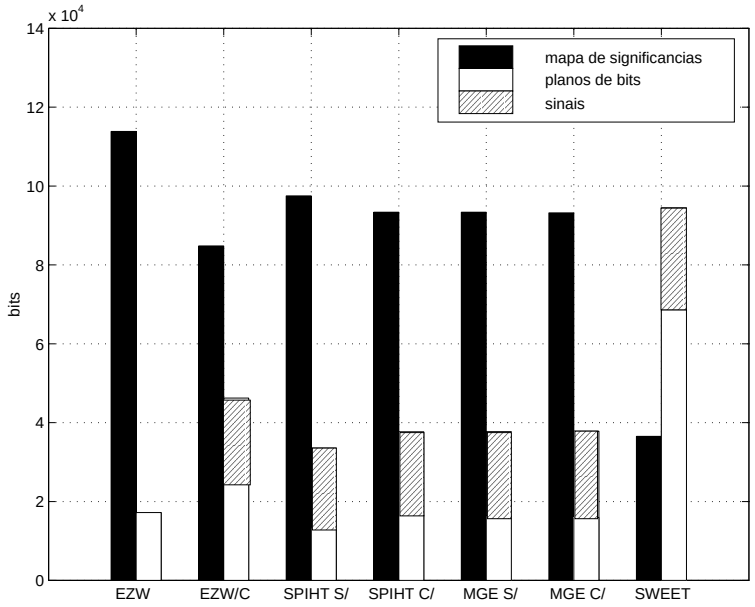


(f)

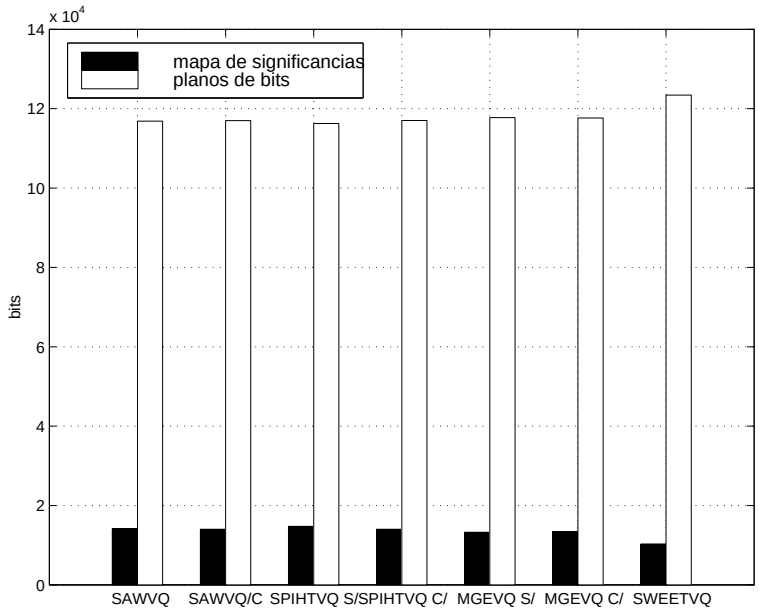


(d)

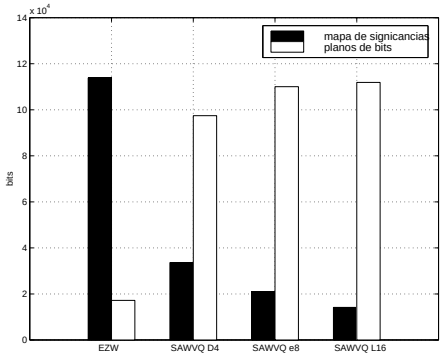
Figura 5.22: Para a imagem Zelda a 0,2 bits/pixel, comparação dos: (a) codificadores escalares; (b) codificadores vectoriais; (c) codificadores escalares e vectoriais EZW; (d) codificadores escalares e vectoriais EZW/C; (e) codificadores escalares e vectoriais SPIHT; (f) codificadores escalares e vectoriais MGE; (g) codificadores escalares e vectoriais SWEET


x 10^4
14
12
10
8
6
4
2
0
bits
mapa de significancias
planos de bits
sinais
EZW
EZW/C
SPIHT S/
SPIHT C/
MGE S/
MGE C/
SWEET

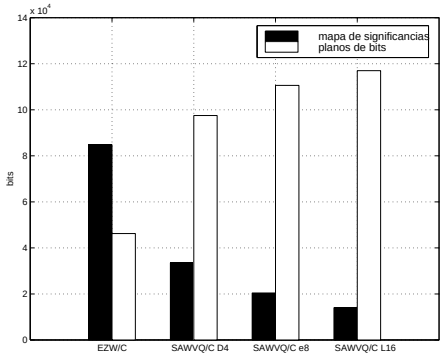
(a)


x 10^4
14
12
10
8
6
4
2
0
bits
mapa de significancias
planos de bits
SAWVQ
SAWVQ/C
SPIHTVQ S/
SPIHTVQ C/
MGEVQ S/
MGEVQ C/
SWEETVQ

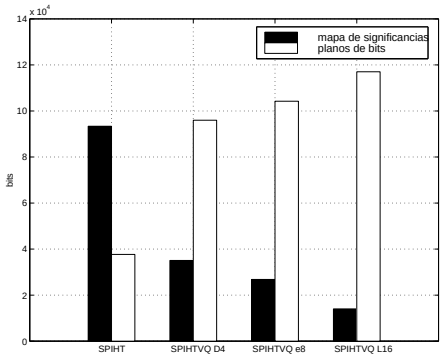
(b)



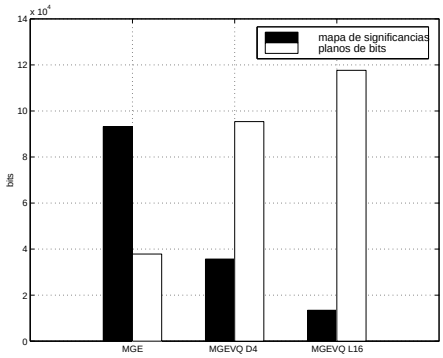
(c)



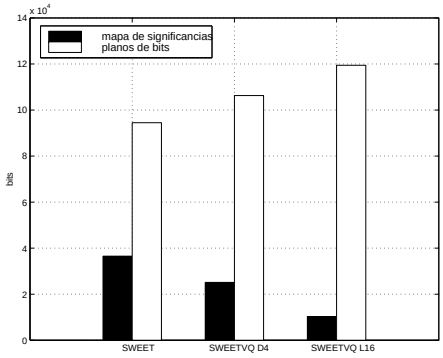
(d)



(e)

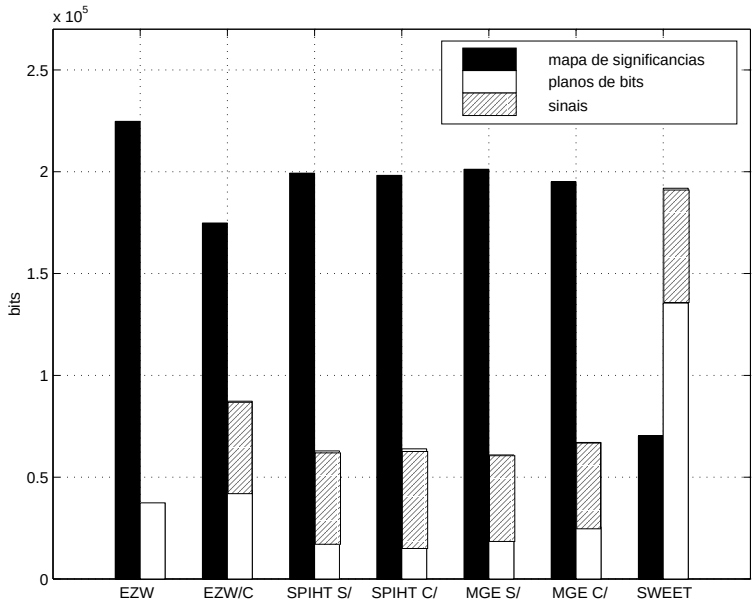


(f)

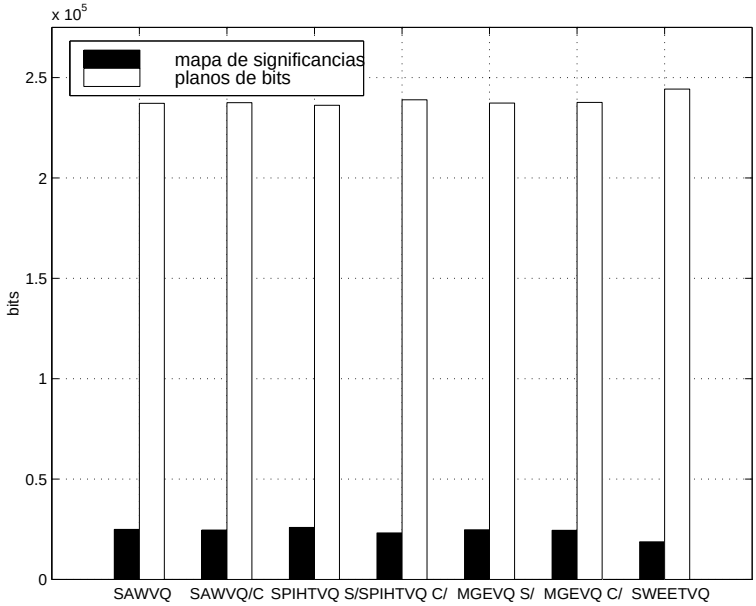


(d)

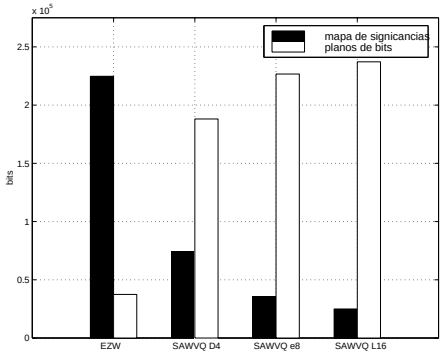
Figura 5.23: Para a imagem Zelda a 0,5 bits/pixel, comparação dos: (a) codificadores escalares; (b) codificadores vectoriais; (c) codificadores escalares e vectoriais EZW; (d) codificadores escalares e vectoriais EZW/C; (e) codificadores escalares e vectoriais SPIHT; (f) codificadores escalares e vectoriais MGE; (g) codificadores escalares e vectoriais SWEET


x 10^5
2.5
2
1.5
1
0.5
0
bits
mapa de significancias
planos de bits
sinais
EZW
EZW/C
SPIHT S/
SPIHT C/
MGE S/
MGE C/
SWEET

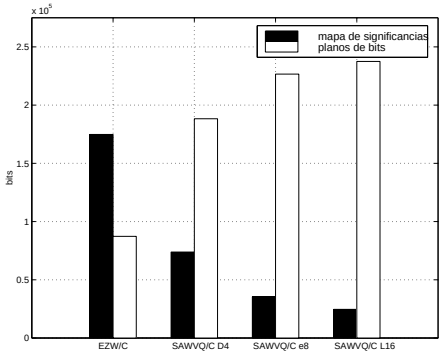
(a)


x 10^5
2.5
2
1.5
1
0.5
0
bits
mapa de significancias
planos de bits
SAWVQ
SAWVQ/C
SPIHTVQ S/
SPIHTVQ C/
MGEVQ S/
MGEVQ C/
SWEETVQ

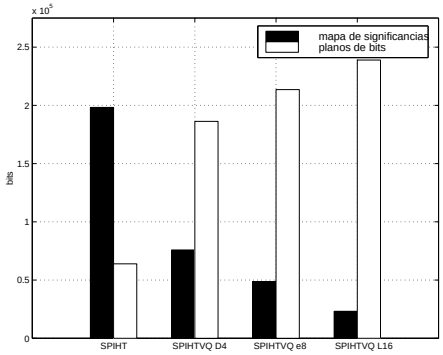
(b)



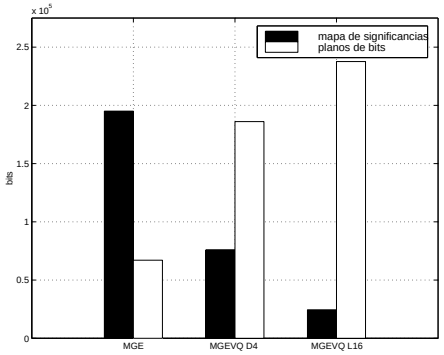
(c)



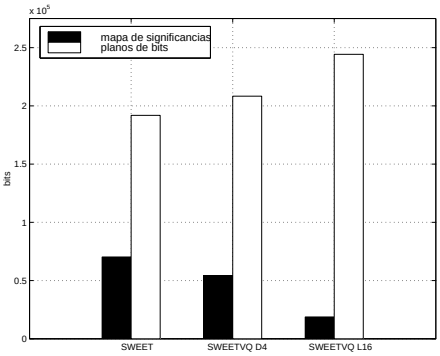
(d)



(e)



(f)



(d)

Figura 5.24: Para a imagem Zelda a 1 bits/pixel, comparação dos: (a) codificadores escalares; (b) codificadores vectoriais; (c) codificadores escalares e vectoriais EZW; (d) codificadores escalares e vectoriais EZW/C; (e) codificadores escalares e vectoriais SPIHT; (f) codificadores escalares e vectoriais MGE; (g) codificadores escalares e vectoriais SWEET

Capítulo 6

Conclusões

Neste trabalho foram desenvolvidos algoritmos baseados na codificação de planos de bits vectoriais. Eles são generalizações de algoritmos já existentes, onde a quantização é feita por planos de bits escalares.

Tanto as versões que fazem quantização através de grandezas escalares assim como as que utilizam grandezas vectoriais foram analisadas e o seu desempenho comparado. Assim, esta análise foi feita segundo um critério de taxa×distorção e em relação ao número de bits gastos em cada uma das etapas da codificação da imagem.

Após a análise dos algoritmos é observado o seguinte:

- A principal diferença entre os diversos algoritmos está na forma como a informação referente à significância é codificada;
- Não existe quase diferença entre os diversos algoritmos que fazem codificação por planos de bits vectoriais, uma vez que a informação de significância praticamente não tem peso no processo de codificação. Contudo, uma excepção é o algoritmo SWEETVQ que na sua versão escalar já apresentava muito pouca informação de significância;

Pode-se então concluir o seguinte:

- Há ganhos em utilizar implementações usando planos de bits vetoriais, e que há indicações de que aumentos significativos de desempenho podem ser obtidos se

for aumentada a eficiência de codificação dos planos de bits. Entre as tentativas neste sentido inclui-se, por exemplo, o uso de outros tipos de dicionários;

- Os algoritmos implementados com quantização vectorial são muito similares, não apresentado praticamente nenhuma diferença nos seus desempenhos;

- Os algoritmos implementados com quantização vectorial levam, por um lado, a uma diminuição na carga computacional pois, na codificação do mapa de significâncias existem muito menos elementos a serem enviados, uma vez que cada vector contém mais do que um coeficiente. Por outro lado, esta carga computacional aumenta com a necessidade de procura do vector mais próximo dentre aqueles que se encontram no dicionário. Contudo, em [3] são apresentados algoritmos rápidos de busca dos vectores nos reticulados. O processo de descodificação é muito rápido quando é usado este tipo de quantização;

- Em termos práticos, apesar da melhoria de desempenho apresentada com os algoritmos que usam quantização vectorial, os algoritmos que utilizam quantização escalar continuam a ser os mais utilizaveis devido à sua simplicidade;

- O uso da codificação aritmética apresentam melhorias bastante significativas no desempenho dos algoritmos.

Este trabalho mostra que é interessante investigar uma forma mais eficiente de implementar o algoritmo SWEETVQ.

Espera-se também que futuros desenvolvimentos na teoria da quantização vectorial por aproximações sucessivas levem a uma melhoria na codificação por planos de bits vectoriais. Este tipo de melhoria pode levar à codificação em planos de bits vectoriais de forma ainda mais eficiente do que aquela que é usada neste trabalho.

Referências Bibliográficas

[1] V. Ralph Algazi and Jr. Robert R. Estes. Analysis based coding of image transform and subband coefficients. In *Proceeding SPIE*, pages 11–21, 1995.

[2] Jim Andrew. A simpler and efficient hierarchical image coder. In *ICIP*, Santa Barbara,CA, 1997.

[3] J. H. Conway and N. J. A. Sloane. *Sphere Packings, Lattices and Groups.* Springer-Verlag, New York, 1988.

[4] Marcos Craizer, Eduardo Antônio Barros da SILVA, and Eloane Gonçalves Ramos. Convergent algorithms for successive approximation vector quantization with applications to wavelet image coding. *IEE Proceedings - Vision, Image and Signal Processing.* To appear.

[5] Eduardo A. B. da Silva and Mohammad Ghanbari. On the performance of linear phase wavelet transforms in low bit rate image coding. *IEEE Transactions on Image Processing*, 5(5):689–704, May 1996.

[6] Eduardo A. Barros da Silva. *Wavelet Transforms for Image Coding.* PhD thesis, University of Essex, England, 1981.

[7] Eduardo Antônio Barros da SILVA and Marcos Craizer. Generalized bit-planes for embedded codes. In *1998 IEEE International Conference on Image Processing*, Chicago, Illinois, October 1998.

[8] Eduardo Antonio Barros da SILVA, Demetrios G. Sampson, and Mohammad Ghanbari. A successive approximation vector quantizer for wavelet transform image coding. *IEEE Transactions on Image Processing, Special Issue on Vector Quantization*, 5(2):299–310, February 1996.

[9] ISO/IEC JTC1/SC29/WG11. MPEG-4 video verification model version 8.0, July 1997.

[10] Anil K. Jain. *Fundamentals of Digital Image Processing*. Prentice Hall, 1989.

[11] N. S. Jayant and P. Noll. *Digital coding of waveforms*. Prentice Hall, New York, 1984.

[12] J.M.Shapiro. Embedded image coding using zerotrees os wavelet coefficients. *IEEE Transactions on Signal Processing*, 41:3445–3462, Dec. 1993.

[13] JPEG2000. Jpeg2000 verification model 5.3 (technical description). JPEG200 Website, JPEG2000 core and VM reflectors. Editor: Charilaos Christopoulos.

[14] Tse-Hua Lan and Ahmed H. Tewfik. Multigrid embedding (mge) image coding. In *ICIP*, Kobe, 1999.

[15] Amir Said and William Pearlman. A new fast and efficient image codec based on set partitioning in hierarchical trees. In *1996 IEEE Transactions on Circuits and Systems for Video Technology*, volume 6, Chicago, Illinois, June 1996.

[16] M. Vetterli and C. Herley. Wavelets and filters banks. *IEEE Transactions on Signal Processing*, 40(9):2207–2232, Sept. 1992.

[17] I.H. Witten, R. Neal, and J. G. Cleary. Arithmetic coding for data compression. *Comm. ACM*, 30:520–540, June 1987.

Apêndice A

Lista de Abreviaturas

- **EZW** - *Embedded Zerotree Wavelet*;
- **SPIHT** - *Set Partitional in Hierarchical Trees*;
- **SWEET** - *Structured Wavelet Embedded Encoding Trees*;
- **MGE** - *Multigrid Embedding*;
- **SAWVQ** - *Successive Approximation Wavelet Vector Quantization*;
- **SPIHTVQ** - *Set Partitional in Hierarchical Trees Vector Quantization*;
- **SWEETVQ** - *Structured Wavelet Embedded Encoding Trees Vector Quantization*;
- **MGEVQ** - *Multigrid Embedding Vector Quantization.*

Apêndice B

Imagens originais

Figura B.1: Lena 512×512, 8 bit/pixel original

Figura B.2: Barbara 512×512, 8 bit/pixel original

Figura B.3: Boats 512×512, 8 bit/pixel original

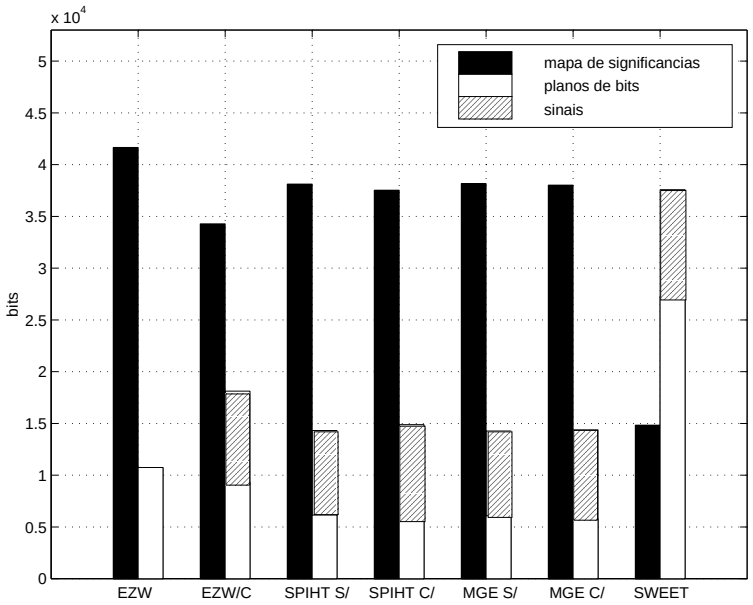
Figura B.4: Girl 512×512, 8 bit/pixel original

Figura B.5: Gold 512×512, 8 bit/pixel original

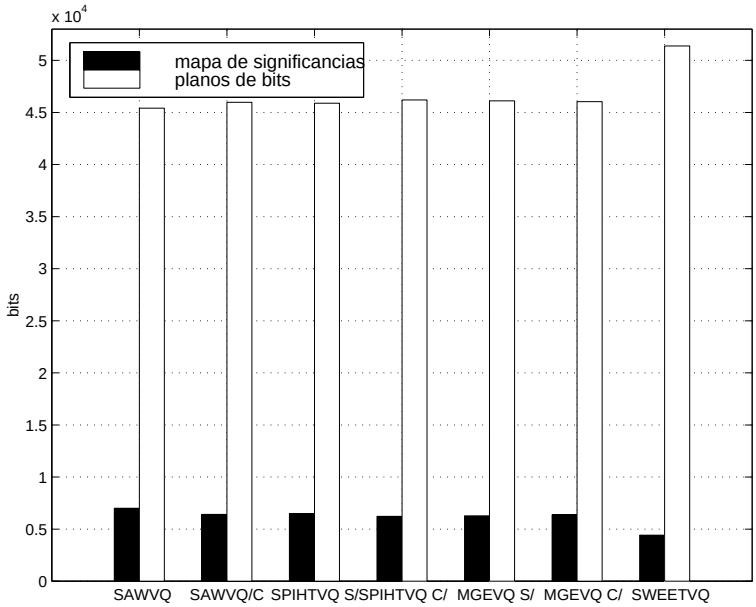
Figura B.6: Zelda 512×512, 8 bit/pixel original

Apêndice C

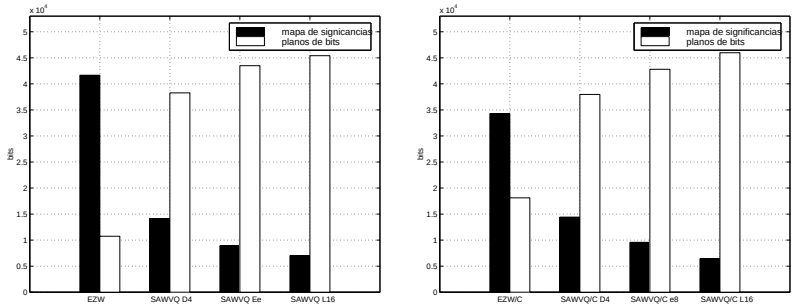
Graficos


x 10^4
mapa de significancias
planos de bits
sinais
bits
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
EZW
EZW/C
SPIHT S/
SPIHT C/
MGE S/
MGE C/
SWEET

(a)

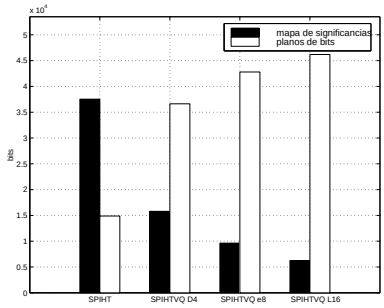

x 10^4
mapa de significancias
planos de bits
bits
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
SAWVQ
SAWVQ/C
SPIHTVQ S/
SPIHTVQ C/
MGEVQ S/
MGEVQ C/
SWEETVQ

(b)

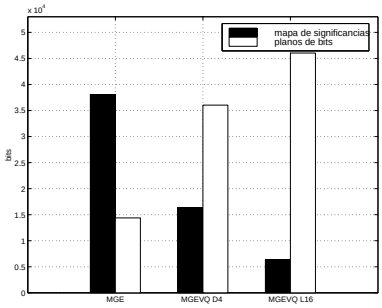


(c)

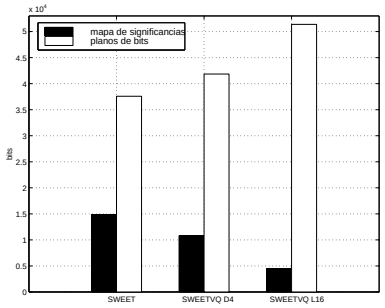
(d)



(e)

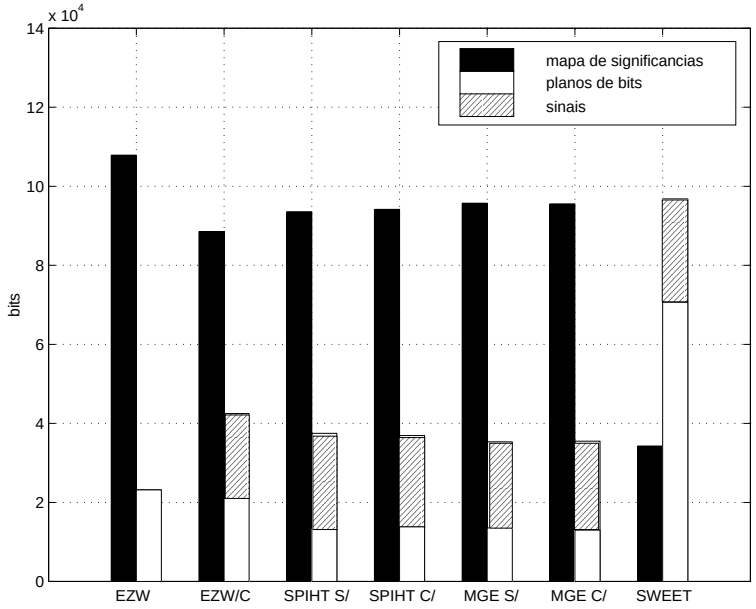


(f)

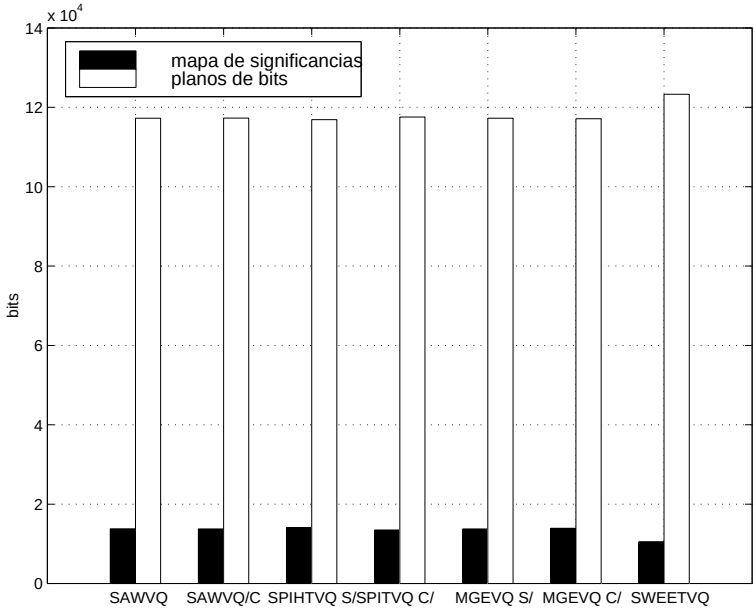


(d)

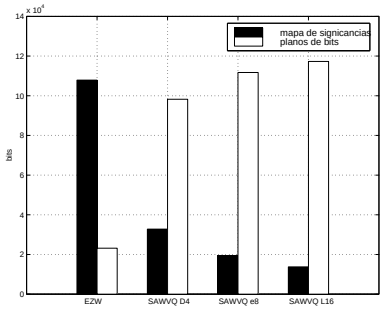
Figura C.1: Para a imagem Lena 512×512 a 0,2 bits/pixel, comparação dos: (a) codificadores escalares; (b) codificadores vetoriais; (c) codificadores escalares e vetoriais EZW; (d) codificadores escalares e vetoriais EZW/C; (e) codificadores escalares e vetoriais SPIHT; (f) codificadores escalares e vetoriais MGE; (g) codificadores escalares e vectoriais SWEET


x 10^4
14
12
10
8
6
4
2
0
bits
mapa de significancias
planos de bits
sinais
EZW
EZW/C
SPIHT S/
SPIHT C/
MGE S/
MGE C/
SWEET

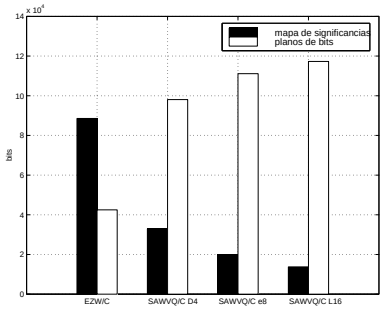
(a)


x 10^4
14
12
10
8
6
4
2
0
bits
mapa de significancias
planos de bits
SAWVQ
SAWVQ/C
SPIHTVQ S/
SPITVQ C/
MGEVQ S/
MGEVQ C/
SWEETVQ

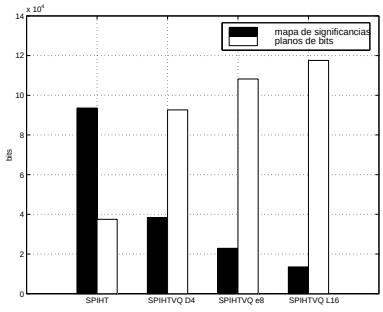
(b)



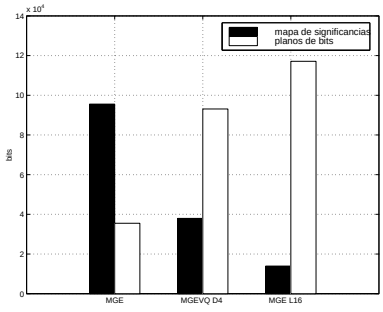
(c)



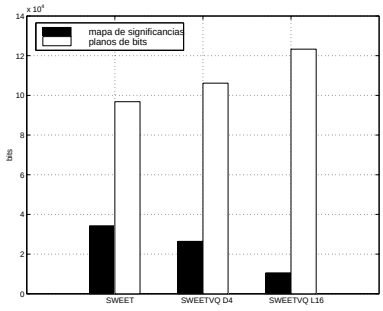
(d)



(e)

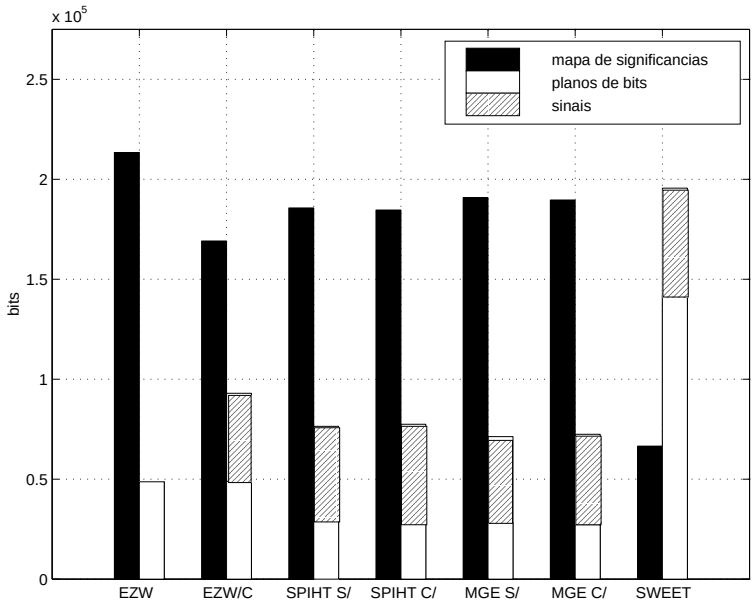


(f)

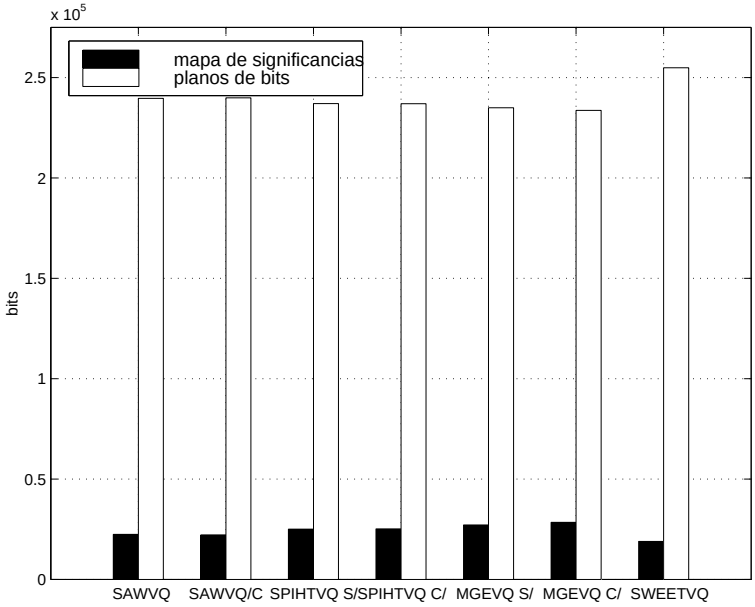


(d)

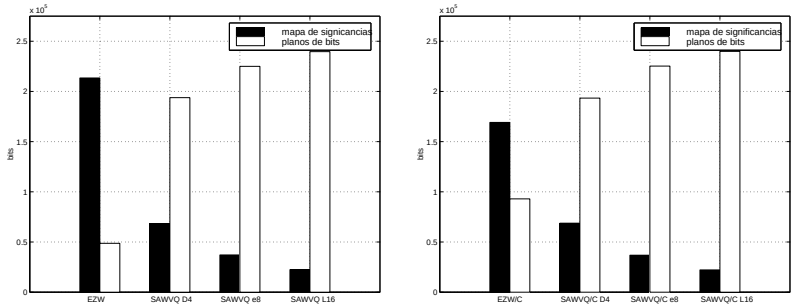
Figura C.2: Para a imagem Lena 512×512 a 0,5 bits/pixel, comparação dos: (a) codificadores escalares; (b) codificadores vetoriais; (c) codificadores escalares e vetoriais EZW; (d) codificadores escalares e vetoriais EZW/C; (e) codificadores escalares e vetoriais SPIHT; (f) codificadores escalares e vetoriais MGE; (g) codificadores escalares e vectoriais SWEET


x 10^5
mapa de significancias
planos de bits
sinais
bits
2.5
2
1.5
1
0.5
0
EZW
EZW/C
SPIHT S/
SPIHT C/
MGE S/
MGE C/
SWEET

(a)

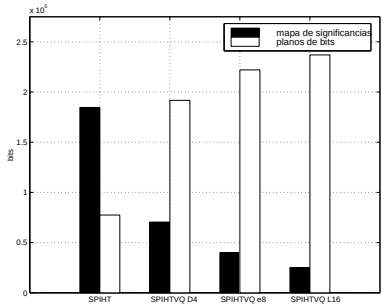

x 10^5
mapa de significancias
planos de bits
bits
2.5
2
1.5
1
0.5
0
SAWVQ
SAWVQ/C
SPIHTVQ S/
SPIHTVQ C/
MGEVQ S/
MGEVQ C/
SWEETVQ

(b)

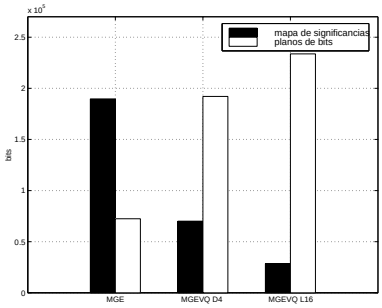


(c)

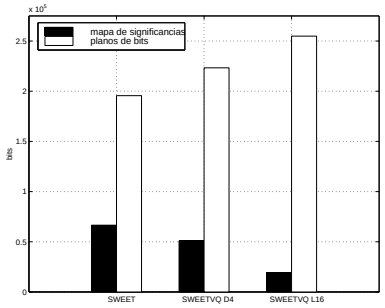
(d)



(e)

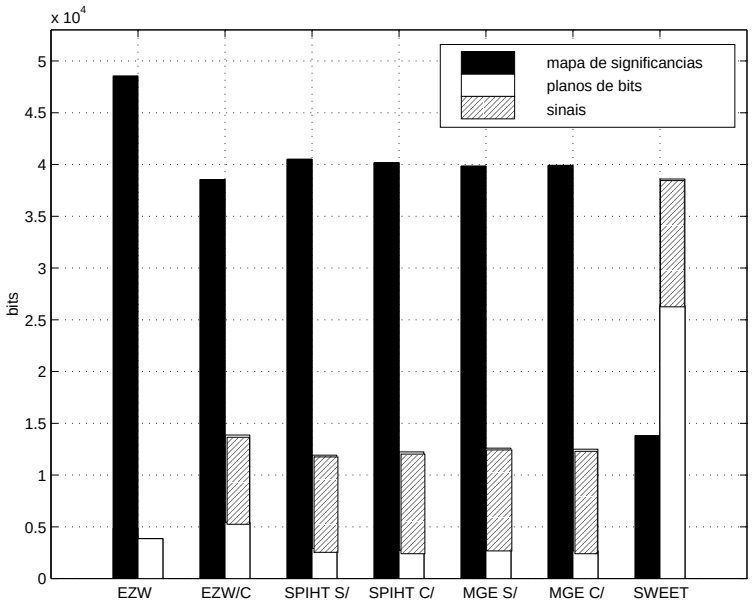


(f)

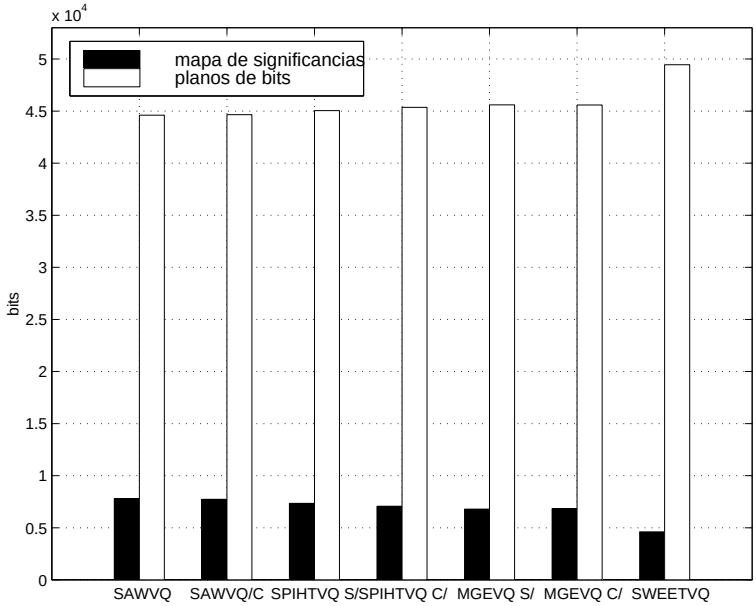


(d)

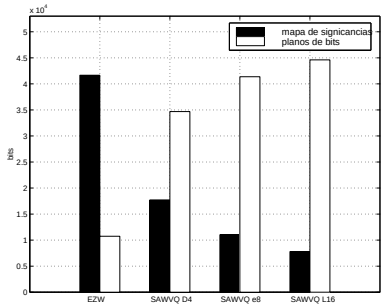
Figura C.3: Para a imagem Lena 512×512 a 1 bits/pixel, comparação dos: (a) codificadores escalares; (b) codificadores vetoriais; (c) codificadores escalares e vetoriais EZW; (d) codificadores escalares e vetoriais EZW/C; (e) codificadores escalares e vetoriais SPIHT; (f) codificadores escalares e vetoriais MGE; (g) codificadores escalares e vectoriais SWEET


x 10^4
mapa de significancias
planos de bits
sinais
bits
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
EZW
EZW/C
SPIHT S/
SPIHT C/
MGE S/
MGE C/
SWEET

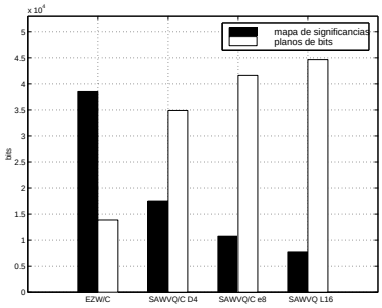
(a)


x 10^4
mapa de significancias
planos de bits
bits
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
SAWVQ
SAWVQ/C
SPIHTVQ S/
SPIHTVQ C/
MGEVQ S/
MGEVQ C/
SWEETVQ

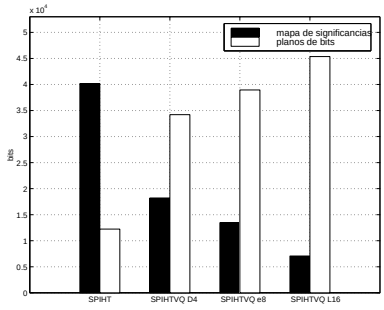
(b)



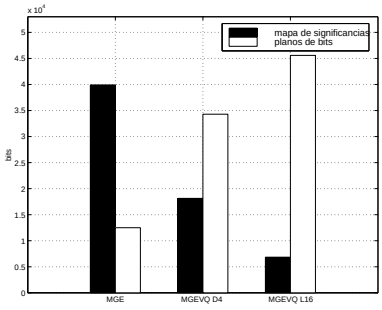
(c)



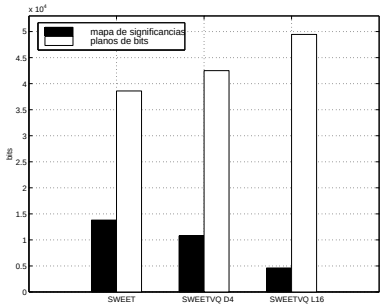
(d)



(e)

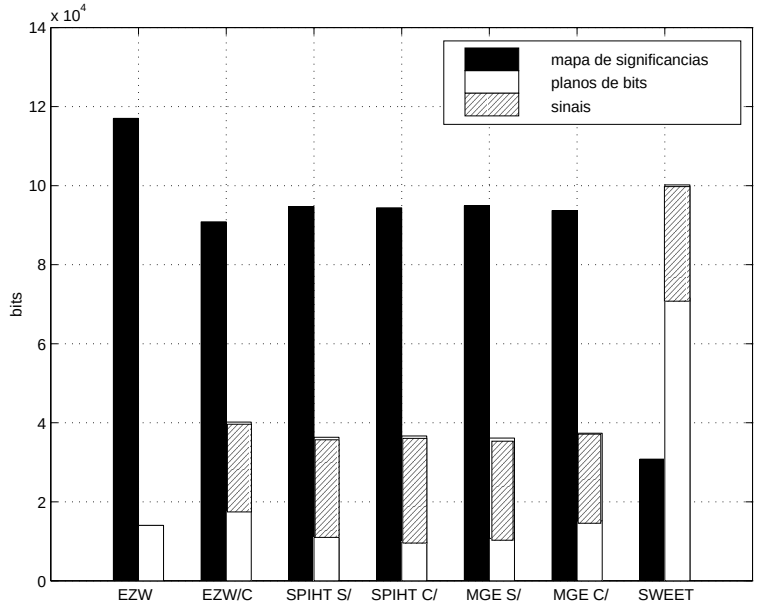


(f)

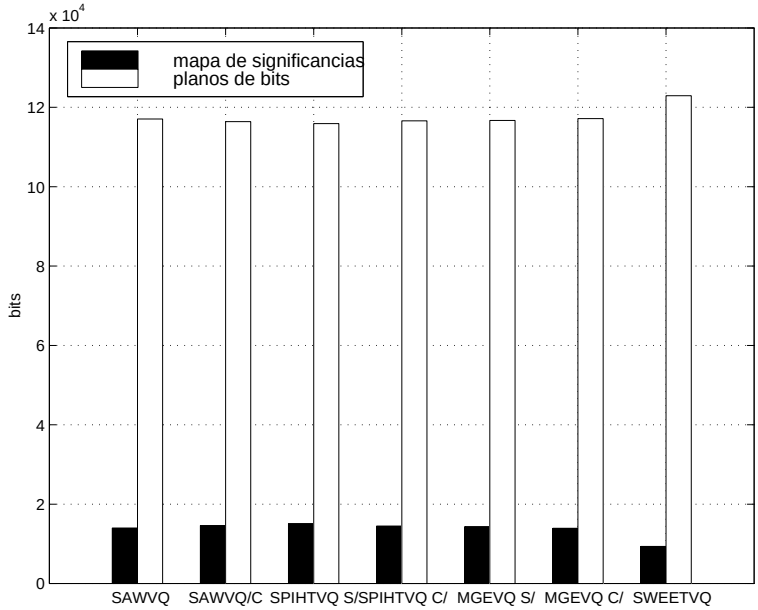


(d)

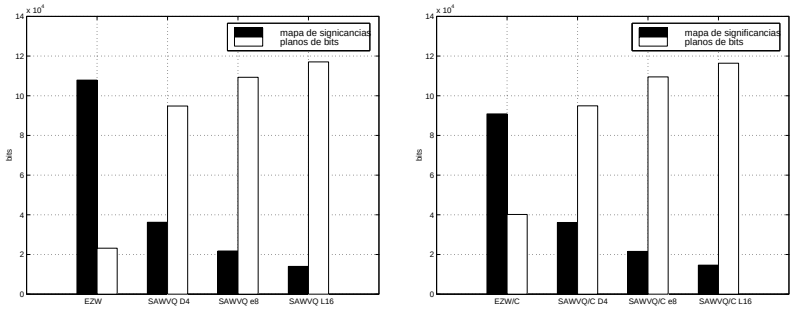
Figura C.4: Para a imagem Barbara a 0,2 bits/pixel, comparação dos: (a) codificadores escalares; (b) codificadores vetoriais; (c) codificadores escalares e vetoriais EZW; (d) codificadores escalares e vetoriais EZW/C; (e) codificadores escalares e vetoriais SPIHT; (f) codificadores escalares e vetoriais MGE; (g) codificadores escalares e vectoriais SWEET


x 10^4
14
12
10
8
6
4
2
0
bits
mapa de significancias
planos de bits
sinais
EZW
EZW/C
SPIHT S/
SPIHT C/
MGE S/
MGE C/
SWEET

(a)

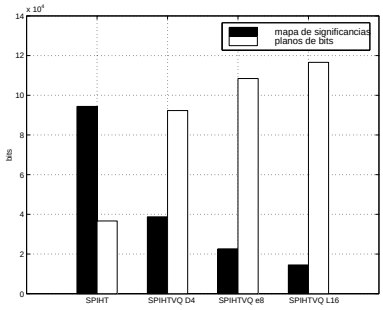

x 10^4
14
12
10
8
6
4
2
0
bits
mapa de significancias
planos de bits
SAWVQ
SAWVQ/C
SPIHTVQ S/
SPIHTVQ C/
MGEVQ S/
MGEVQ C/
SWEETVQ

(b)

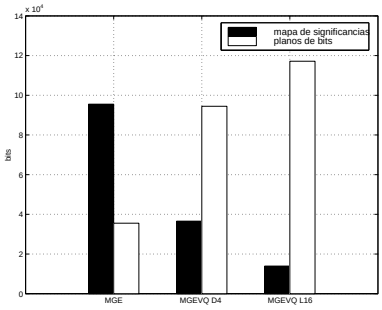


(c)

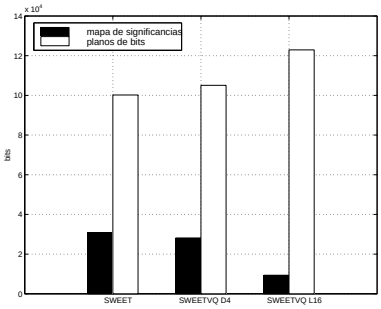
(d)



(e)

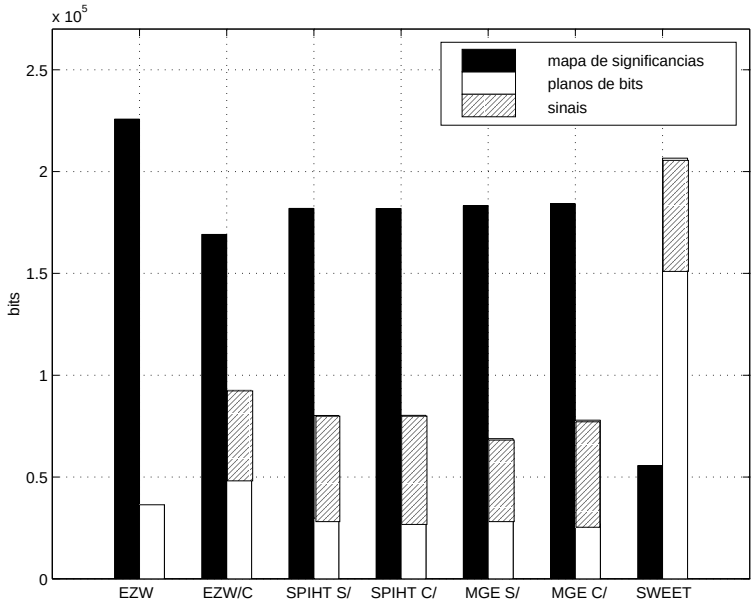


(f)

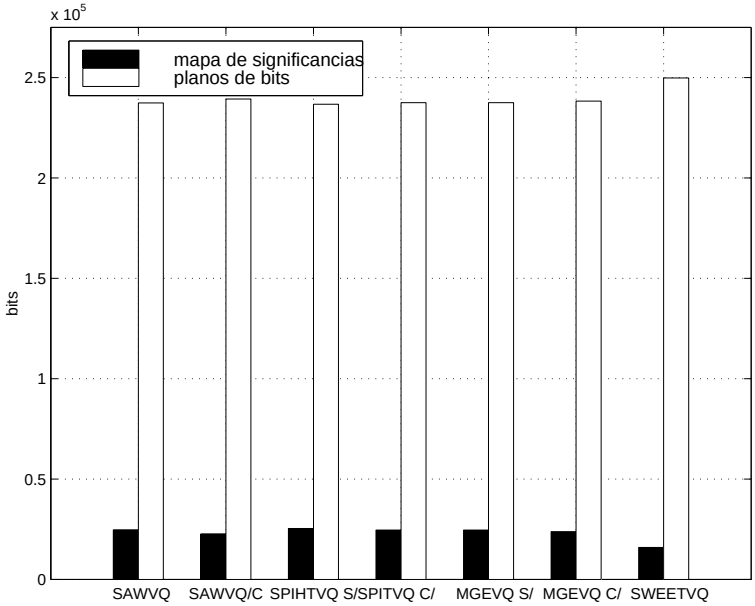


(d)

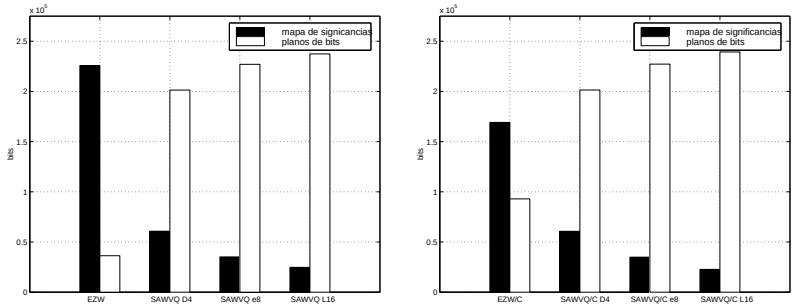
Figura C.5: Para a imagem Barbara a 0,5 bits/pixel, comparação dos: (a) codificadores escalares; (b) codificadores vetoriais; (c) codificadores escalares e vetoriais EZW; (d) codificadores escalares e vetoriais EZW/C; (e) codificadores escalares e vetoriais SPIHT; (f) codificadores escalares e vetoriais MGE; (g) codificadores escalares e vectoriais SWEET


x 10^5
mapa de significancias
planos de bits
sinais
bits
2.5
2
1.5
1
0.5
0
EZW
EZW/C
SPIHT S/
SPIHT C/
MGE S/
MGE C/
SWEET

(a)

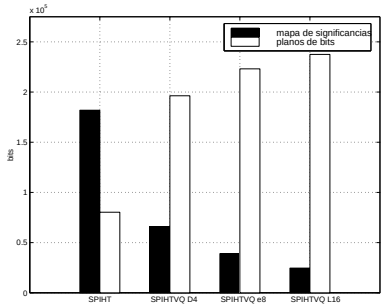

x 10^5
mapa de significancias
planos de bits
bits
2.5
2
1.5
1
0.5
0
SAWVQ
SAWVQ/C
SPIHTVQ S/
SPITVQ C/
MGEVQ S/
MGEVQ C/
SWEETVQ

(b)

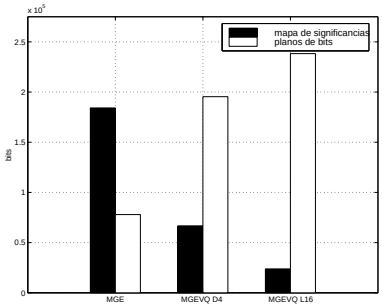


(c)

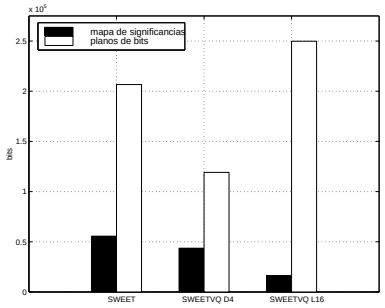
(d)



(e)

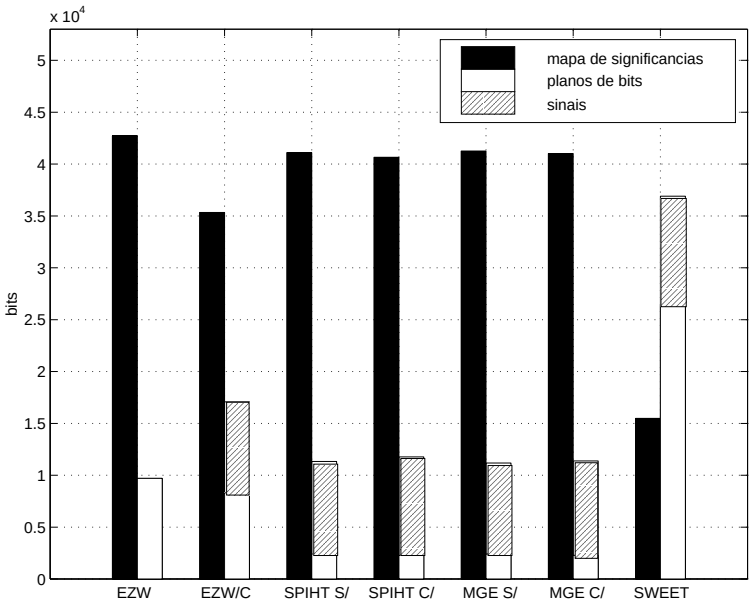


(f)

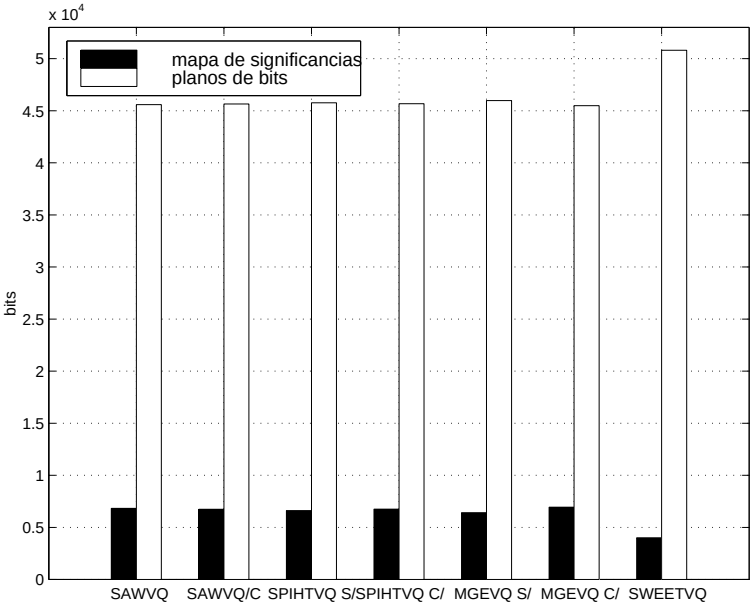


(d)

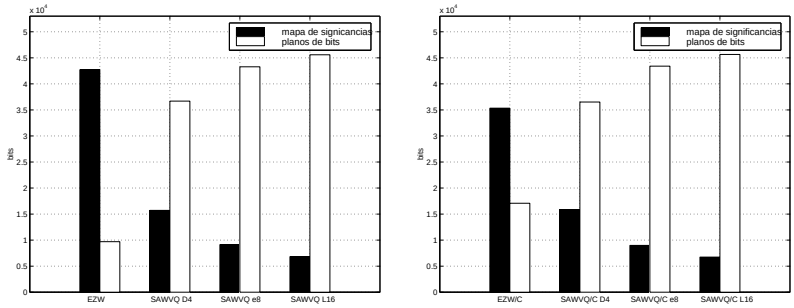
Figura C.6: Para a imagem Barbara a 1 bits/pixel, comparação dos: (a) codificadores escalares; (b) codificadores vetoriais; (c) codificadores escalares e vetoriais EZW; (d) codificadores escalares e vetoriais EZW/C; (e) codificadores escalares e vetoriais SPIHT; (f) codificadores escalares e vetoriais MGE; (g) codificadores escalares e vectoriais SWEET


x 10^4
mapa de significancias
planos de bits
sinais
bits
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
EZW
EZW/C
SPIHT S/
SPIHT C/
MGE S/
MGE C/
SWEET

(a)

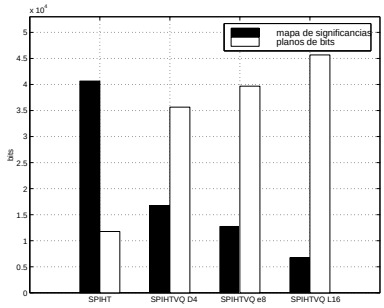

x 10^4
mapa de significancias
planos de bits
bits
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
SAWVQ
SAWVQ/C
SPIHTVQ S/
SPIHTVQ C/
MGEVQ S/
MGEVQ C/
SWEETVQ

(b)

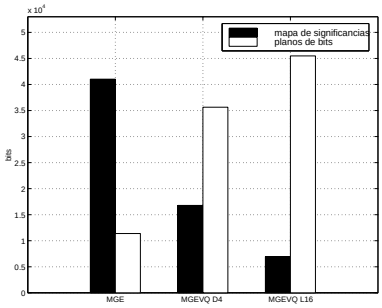


(c)

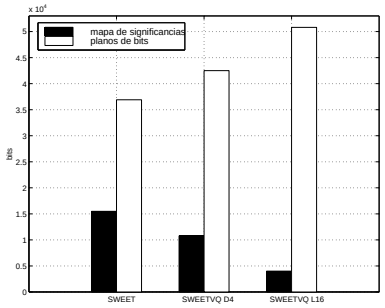
(d)



(e)

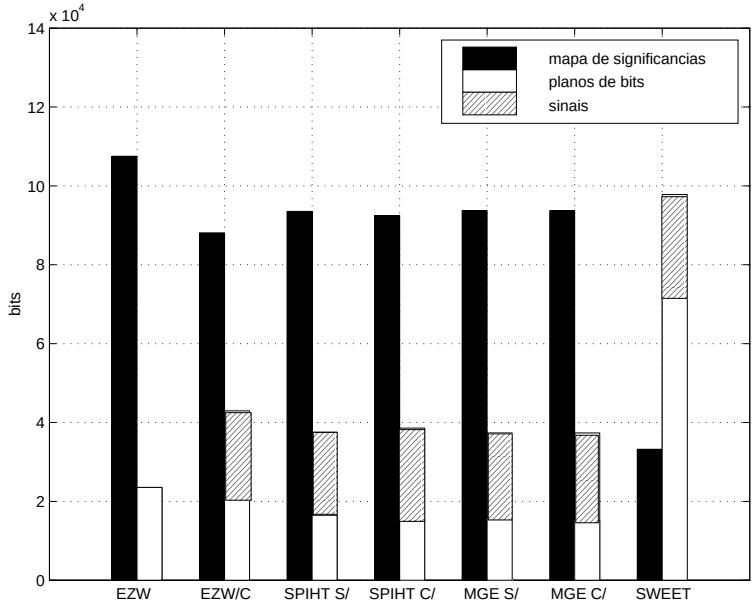


(f)

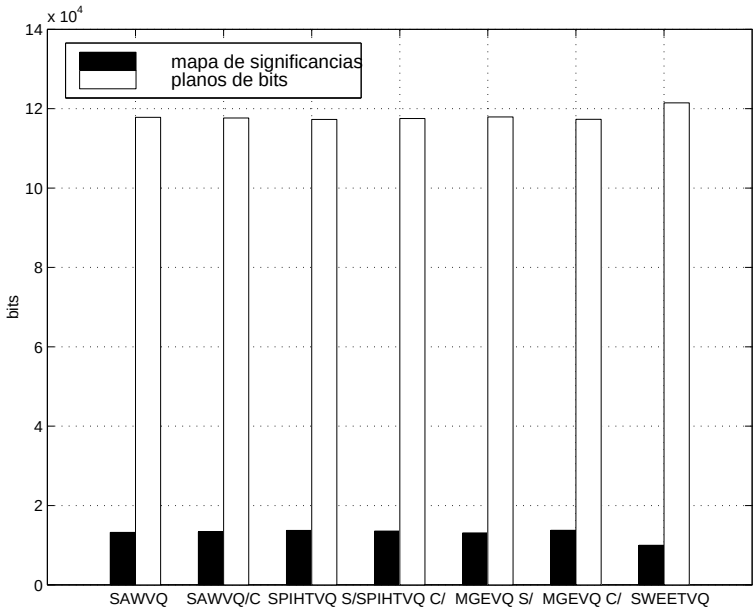


(d)

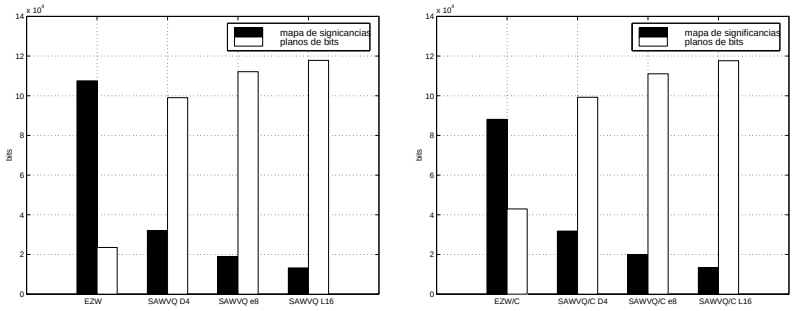
Figura C.7: Para a imagem Boats a 0,2 bits/pixel, comparação dos: (a) codificadores escalares; (b) codificadores vetoriais; (c) codificadores escalares e vetoriais EZW; (d) codificadores escalares e vetoriais EZW/C; (e) codificadores escalares e vetoriais SPIHT; (f) codificadores escalares e vetoriais MGE; (g) codificadores escalares e vectoriais SWEET


x 10^4
14
12
10
8
6
4
2
0
bits
mapa de significancias
planos de bits
sinais
EZW
EZW/C
SPIHT S/
SPIHT C/
MGE S/
MGE C/
SWEET

(a)

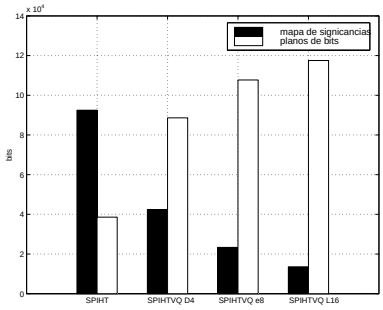

x 10^4
14
12
10
8
6
4
2
0
bits
mapa de significancias
planos de bits
SAWVQ
SAWVQ/C
SPIHTVQ S/
SPIHTVQ C/
MGEVQ S/
MGEVQ C/
SWEETVQ

(b)

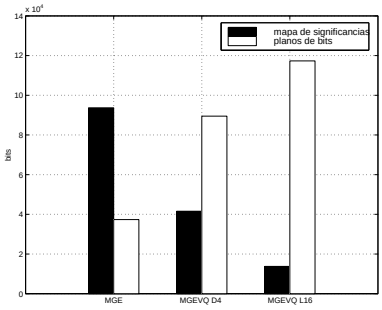


(c)

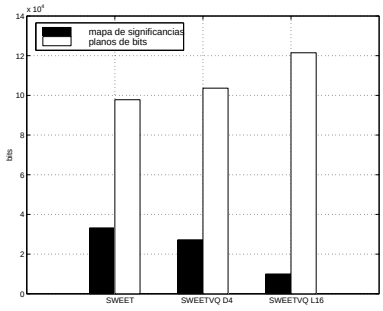
(d)



(e)

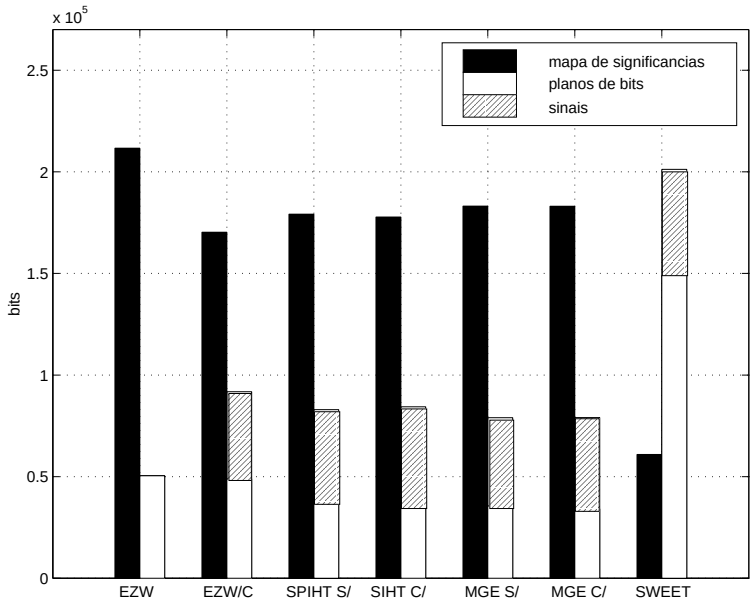


(f)

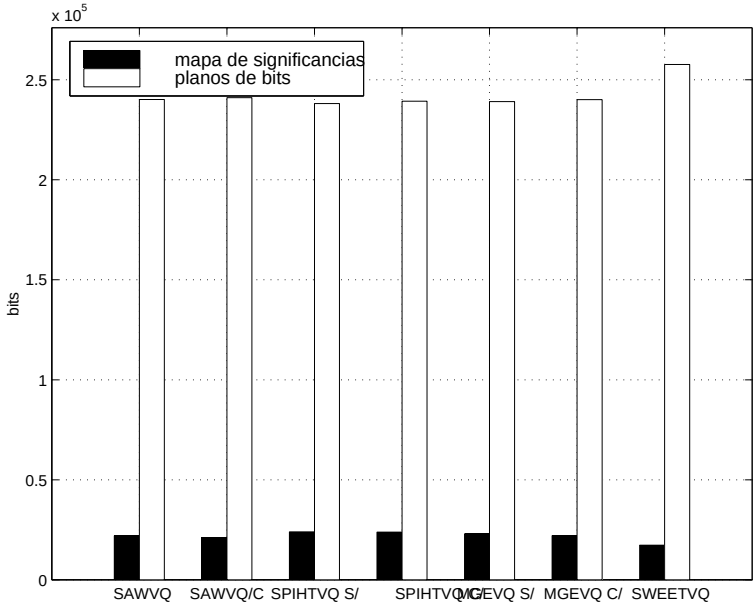


(d)

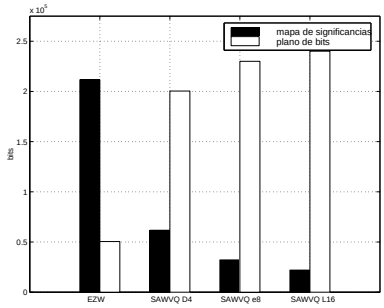
Figura C.8: Para a imagem Boats a 0,5 bits/pixel, comparação dos: (a) codificadores escalares; (b) codificadores vetoriais; (c) codificadores escalares e vetoriais EZW; (d) codificadores escalares e vetoriais EZW/C; (e) codificadores escalares e vetoriais SPIHT; (f) codificadores escalares e vetoriais MGE; (g) codificadores escalares e vectoriais SWEET


x 10^5
mapa de significancias
planos de bits
sinais
bits
2.5
2
1.5
1
0.5
0
EZW
EZW/C
SPIHT S/
SIHT C/
MGE S/
MGE C/
SWEET

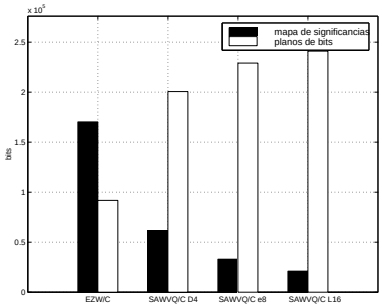
(a)


x 10^5
mapa de significancias
planos de bits
bits
2.5
2
1.5
1
0.5
0
SAWVQ
SAWVQ/C
SPIHTVQ S/
SPIHTVQ C/
MGEVQ S/
MGEVQ C/
SWEETVQ

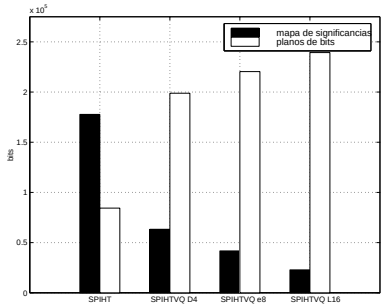
(b)



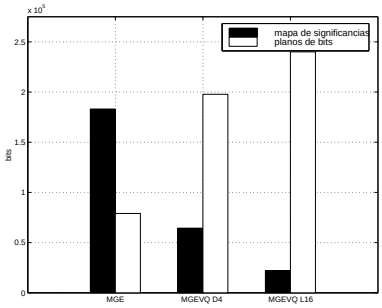
(c)



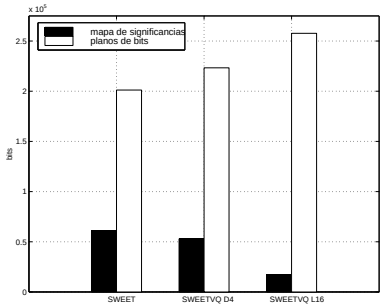
(d)



(e)

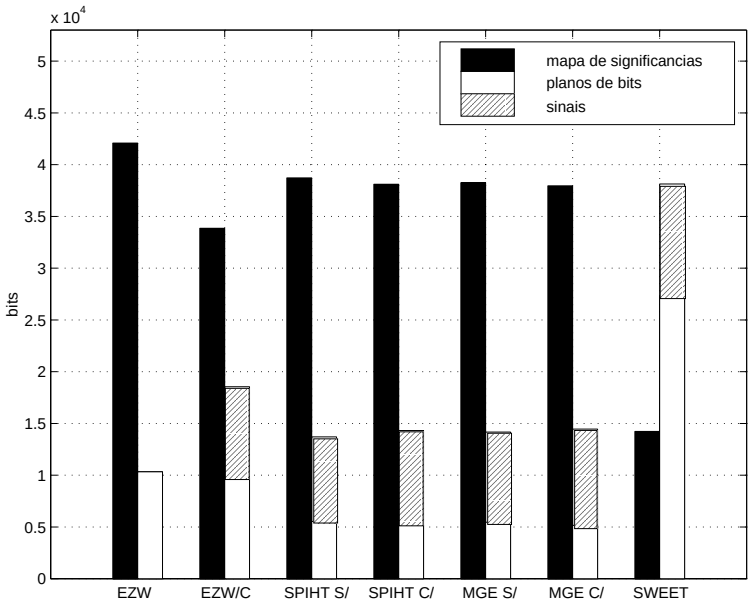


(f)

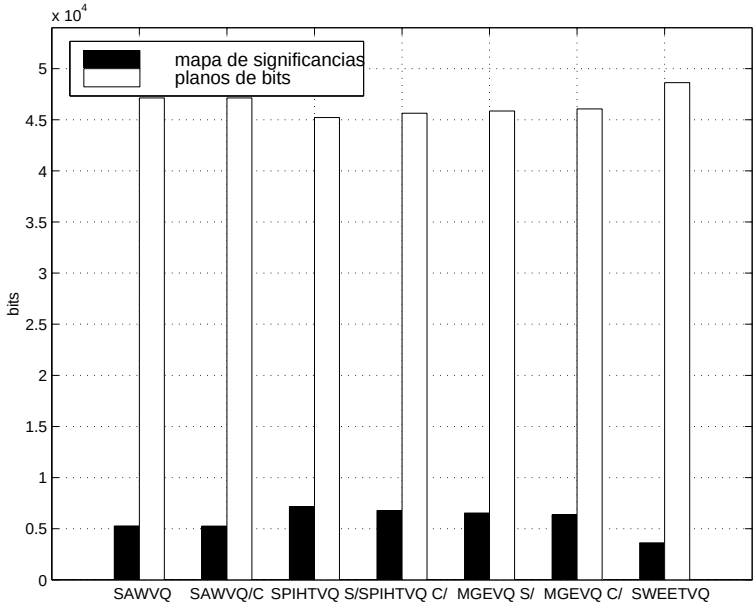


(d)

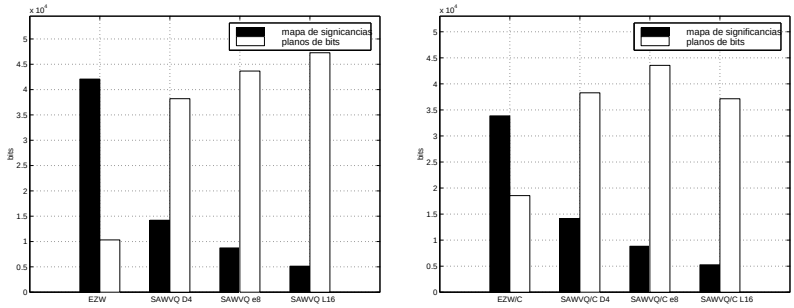
Figura C.9: Para a imagem Boats a 1 bits/pixel, comparação dos: (a) codificadores escalares; (b) codificadores vetoriais; (c) codificadores escalares e vetoriais EZW; (d) codificadores escalares e vetoriais EZW/C; (e) codificadores escalares e vetoriais SPIHT; (f) codificadores escalares e vetoriais MGE; (g) codificadores escalares e vectoriais SWEET


x 10^4
mapa de significancias
planos de bits
sinais
bits
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
EZW
EZW/C
SPIHT S/
SPIHT C/
MGE S/
MGE C/
SWEET

(a)

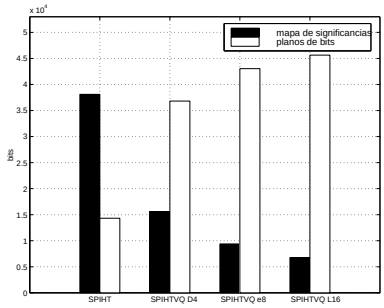

x 10^4
mapa de significancias
planos de bits
bits
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
SAWVQ
SAWVQ/C
SPIHTVQ S/
SPIHTVQ C/
MGEVQ S/
MGEVQ C/
SWEETVQ

(b)

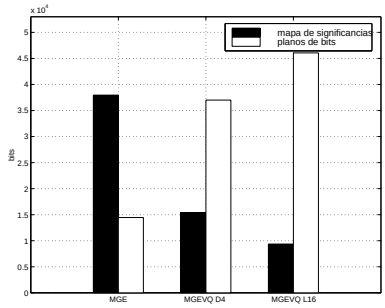


(c)

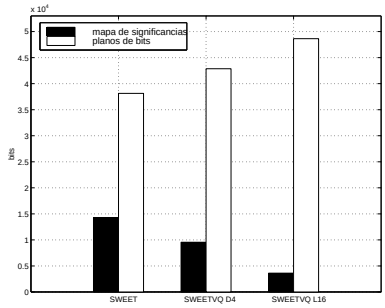
(d)



(e)

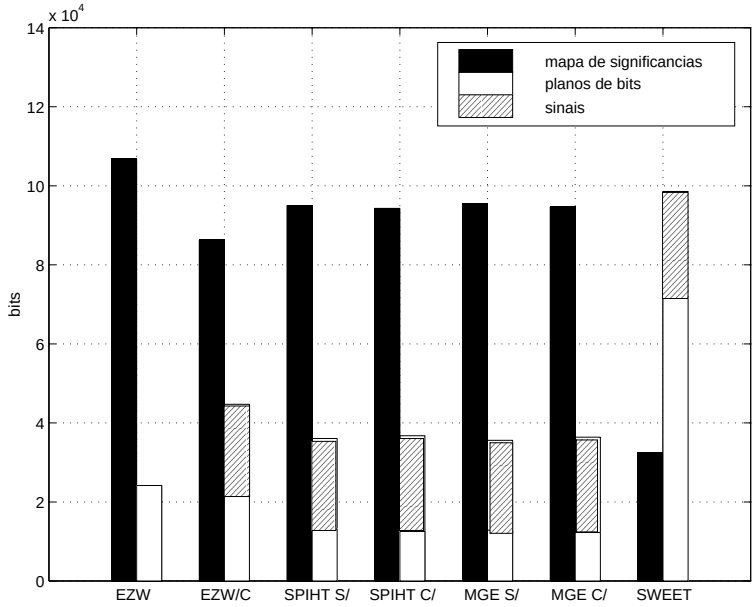


(f)

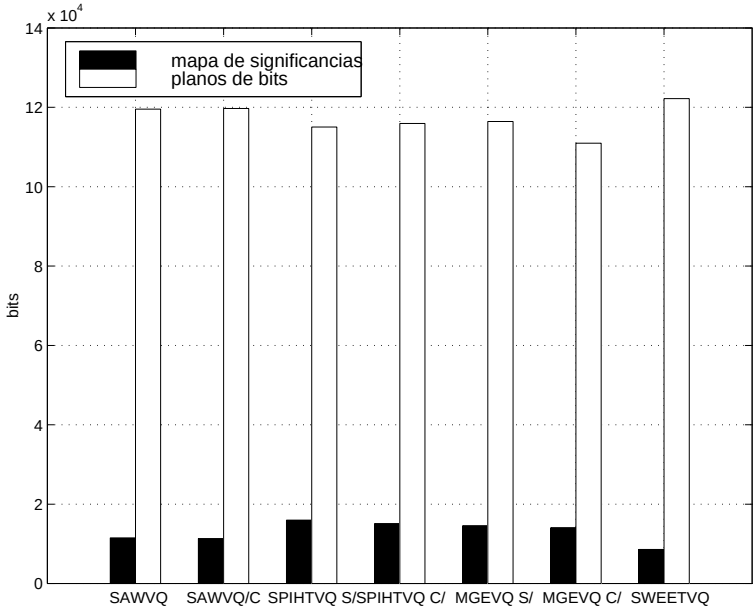


(d)

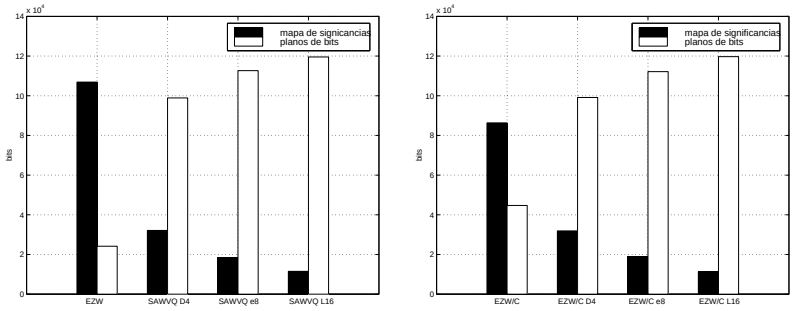
Figura C.10: Para a imagem Girl a 0,2 bits/pixel, comparação dos: (a) codificadores escalares; (b) codificadores vetoriais; (c) codificadores escalares e vetoriais EZW; (d) codificadores escalares e vetoriais EZW/C; (e) codificadores escalares e vetoriais SPIHT; (f) codificadores escalares e vetoriais MGE; (g) codificadores escalares e vectoriais SWEET


x 10^4
14
12
10
8
6
4
2
0
bits
mapa de significancias
planos de bits
sinais
EZW
EZW/C
SPIHT S/
SPIHT C/
MGE S/
MGE C/
SWEET

(a)

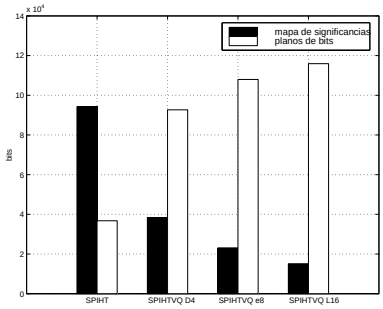

x 10^4
14
12
10
8
6
4
2
0
bits
mapa de significancias
planos de bits
SAWVQ
SAWVQ/C
SPIHTVQ S/
SPIHTVQ C/
MGEVQ S/
MGEVQ C/
SWEETVQ

(b)

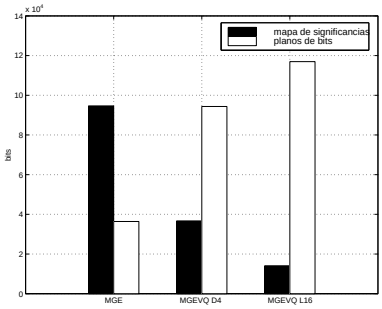


(c)

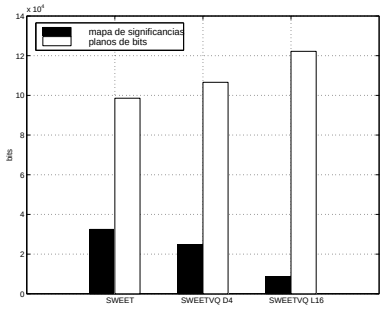
(d)



(e)

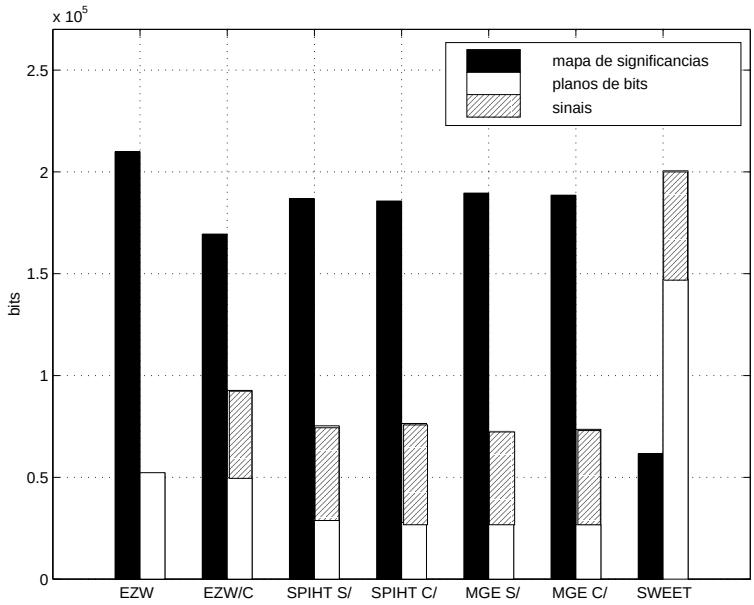


(f)

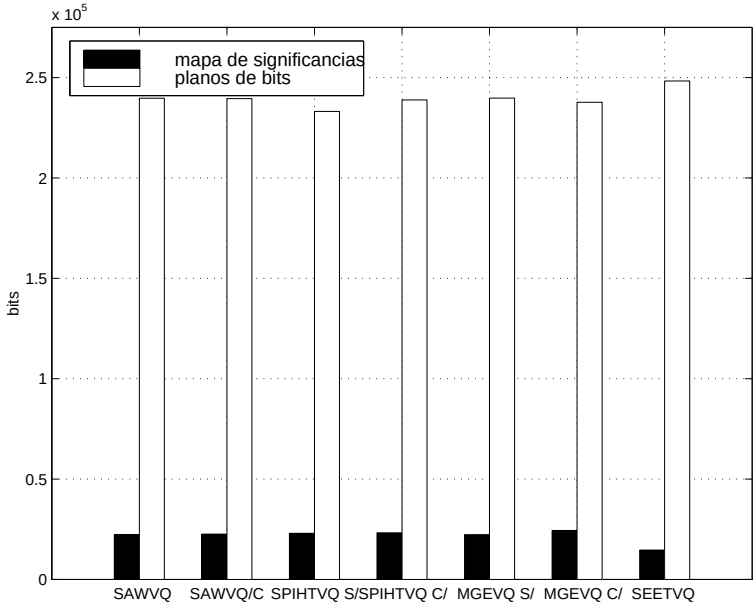


(d)

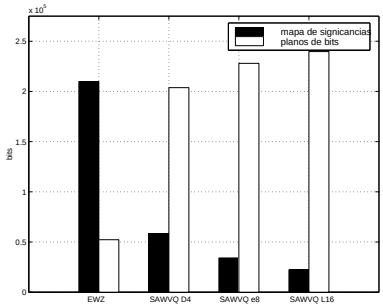
Figura C.11: Para a imagem Girl a 0,5 bits/pixel, comparação dos: (a) codificadores escalares; (b) codificadores vetoriais; (c) codificadores escalares e vetoriais EZW; (d) codificadores escalares e vetoriais EZW/C; (e) codificadores escalares e vetoriais SPIHT; (f) codificadores escalares e vetoriais MGE; (g) codificadores escalares e vectoriais SWEET


x 10^5
2.5
2
1.5
1
0.5
0
bits
mapa de significancias
planos de bits
sinais
EZW
EZW/C
SPIHT S/
SPIHT C/
MGE S/
MGE C/
SWEET

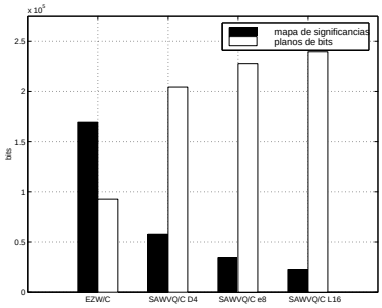
(a)


x 10^5
2.5
2
1.5
1
0.5
0
bits
mapa de significancias
planos de bits
SAWVQ
SAWVQ/C
SPIHTVQ S/
SPIHTVQ C/
MGEVQ S/
MGEVQ C/
SEETVQ

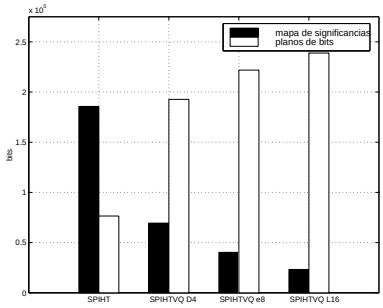
(b)



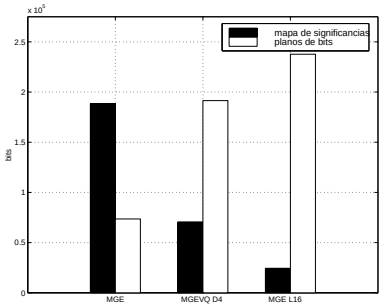
(c)



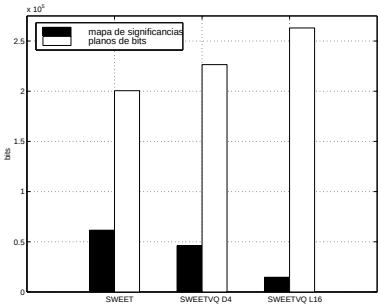
(d)



(e)

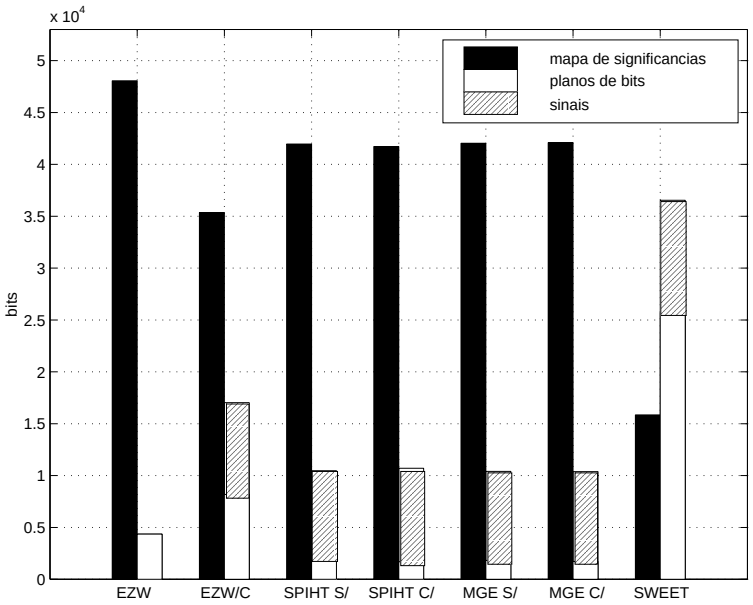


(f)

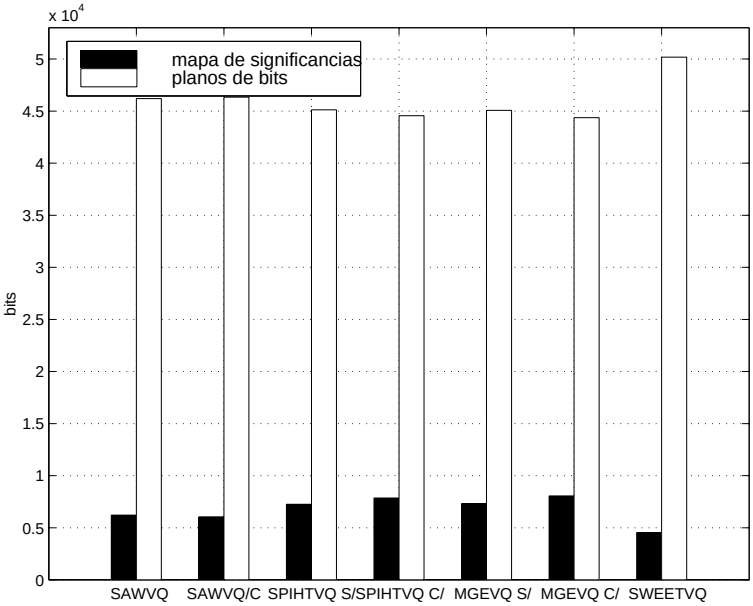


(d)

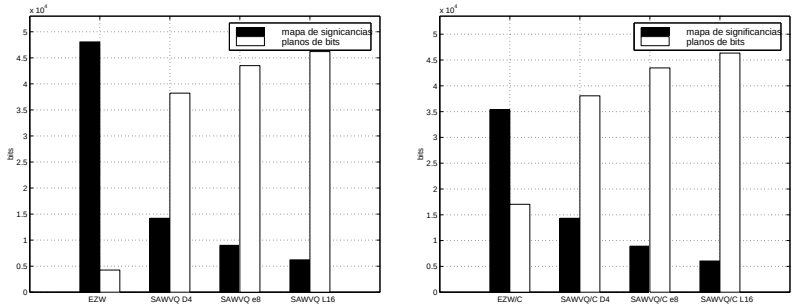
Figura C.12: Para a imagem Girl a 1 bits/pixel, comparação dos: (a) codificadores escalares; (b) codificadores vetoriais; (c) codificadores escalares e vetoriais EZW; (d) codificadores escalares e vetoriais EZW/C; (e) codificadores escalares e vetoriais SPIHT; (f) codificadores escalares e vetoriais MGE; (g) codificadores escalares e vectoriais SWEET


x 10^4
mapa de significancias
planos de bits
sinais
bits
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
EZW
EZW/C
SPIHT S/
SPIHT C/
MGE S/
MGE C/
SWEET

(a)

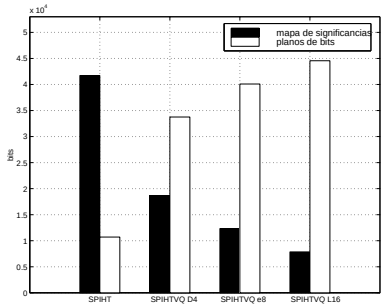

x 10^4
mapa de significancias
planos de bits
bits
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
SAWVQ
SAWVQ/C
SPIHTVQ S/
SPIHTVQ C/
MGEVQ S/
MGEVQ C/
SWEETVQ

(b)

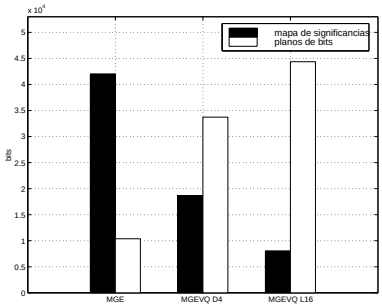


(c)

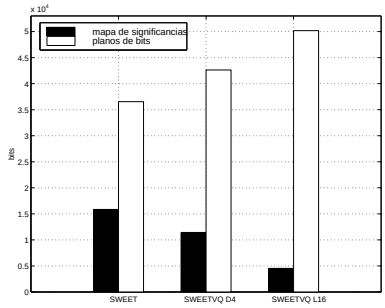
(d)



(e)

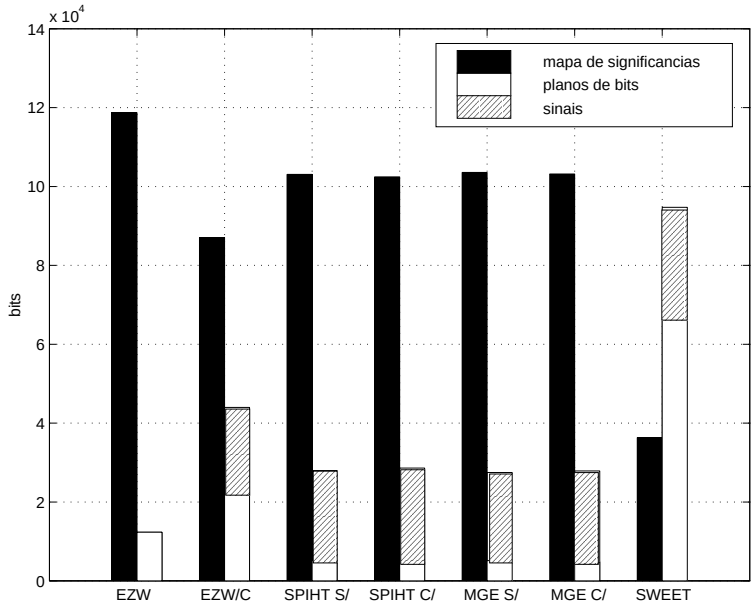


(f)

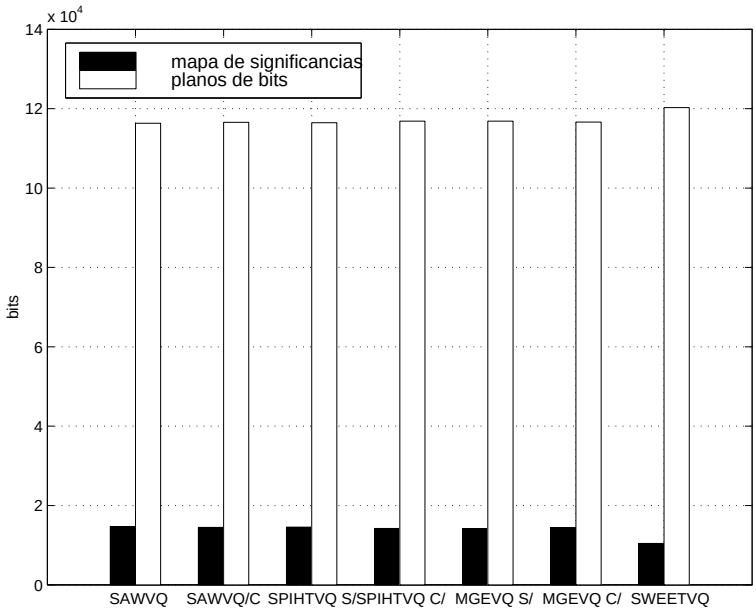


(d)

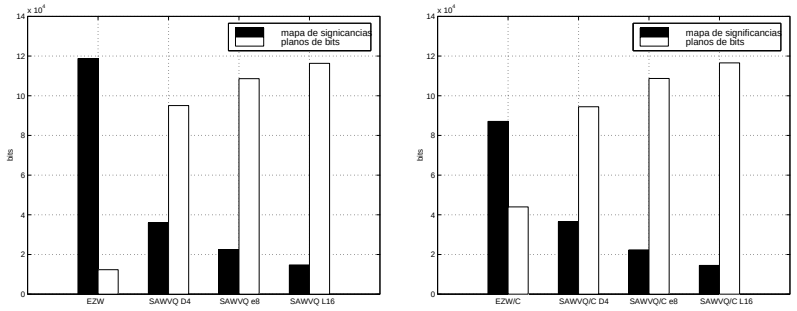
Figura C.13: Para a imagem Gold a 0,2 bits/pixel, comparação dos: (a) codificadores escalares; (b) codificadores vetoriais; (c) codificadores escalares e vetoriais EZW; (d) codificadores escalares e vetoriais EZW/C; (e) codificadores escalares e vetoriais SPIHT; (f) codificadores escalares e vetoriais MGE; (g) codificadores escalares e vectoriais SWEET


x 10^4
14
12
10
8
6
4
2
0
bits
mapa de significancias
planos de bits
sinais
EZW
EZW/C
SPIHT S/
SPIHT C/
MGE S/
MGE C/
SWEET

(a)

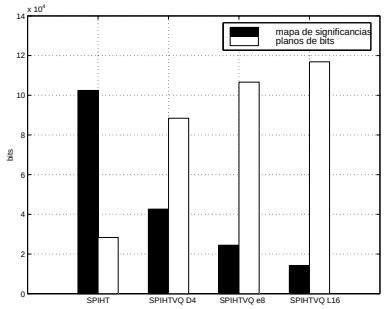

x 10^4
14
12
10
8
6
4
2
0
bits
mapa de significancias
planos de bits
SAWVQ
SAWVQ/C
SPIHTVQ S/
SPIHTVQ C/
MGEVQ S/
MGEVQ C/
SWEETVQ

(b)

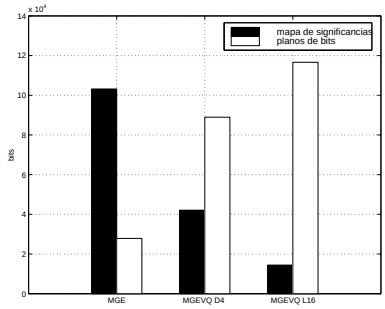


(c)

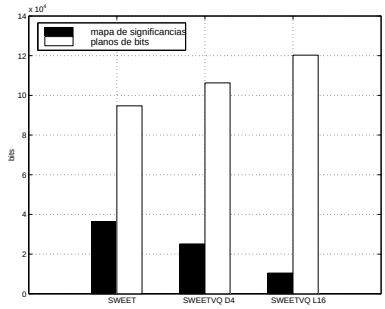
(d)



(e)

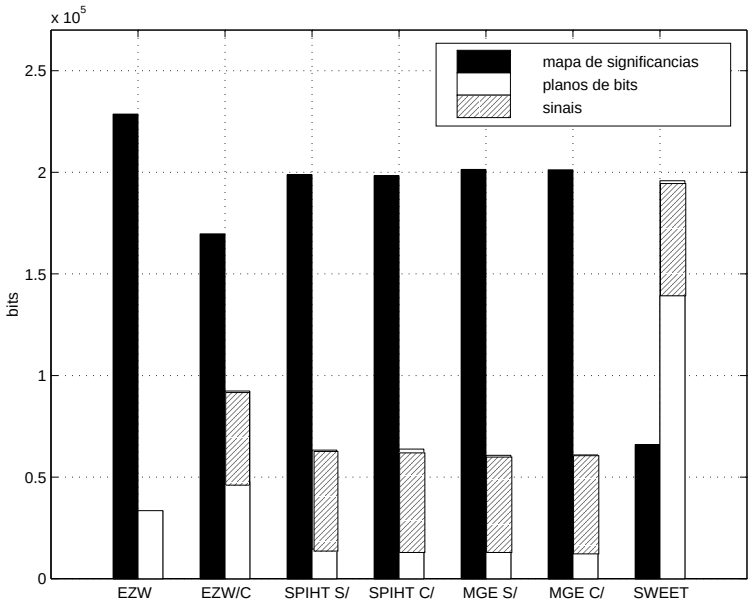


(f)

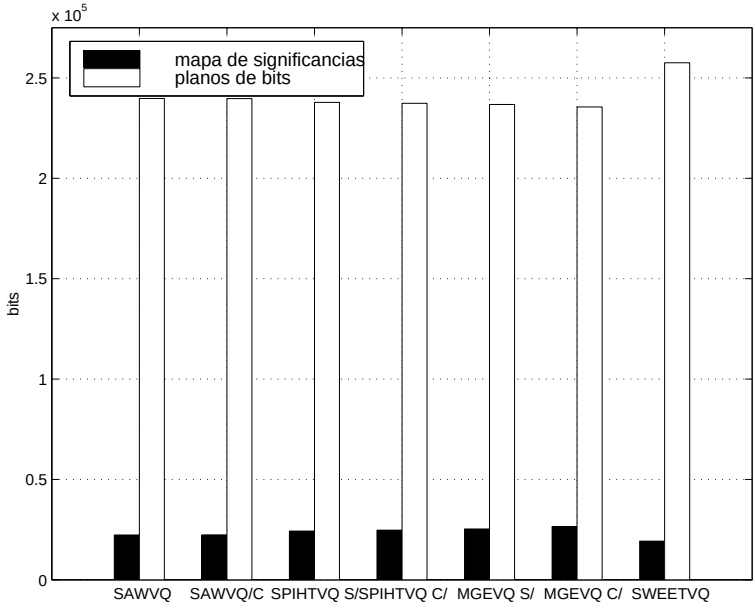


(d)

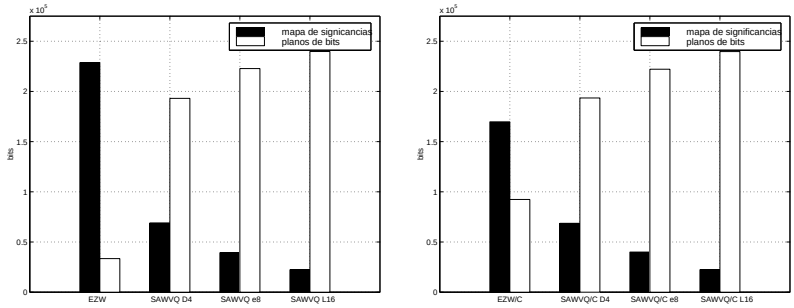
Figura C.14: Para a imagem Gold a 0,5 bits/pixel, comparação dos: (a) codificadores escalares; (b) codificadores vetoriais; (c) codificadores escalares e vetoriais EZW; (d) codificadores escalares e vetoriais EZW/C; (e) codificadores escalares e vetoriais SPIHT; (f) codificadores escalares e vetoriais MGE; (g) codificadores escalares e vectoriais SWEET


x 10^5
2.5
2
1.5
1
0.5
0
bits
mapa de significancias
planos de bits
sinais
EZW
EZW/C
SPIHT S/
SPIHT C/
MGE S/
MGE C/
SWEET

(a)

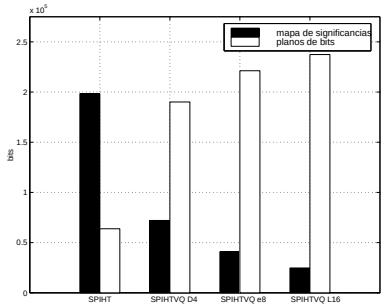

x 10^5
2.5
2
1.5
1
0.5
0
bits
mapa de significancias
planos de bits
SAWVQ
SAWVQ/C
SPIHTVQ S/
SPIHTVQ C/
MGEVQ S/
MGEVQ C/
SWEETVQ

(b)

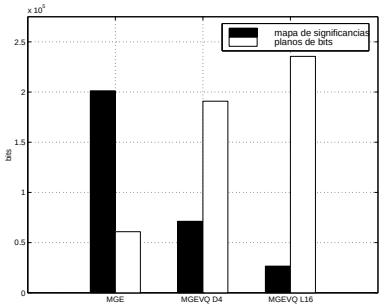


(c)

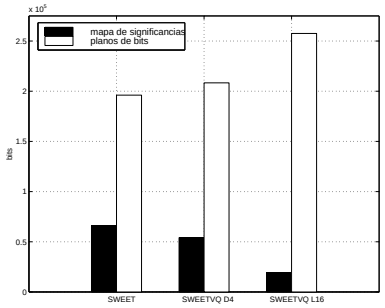
(d)



(e)

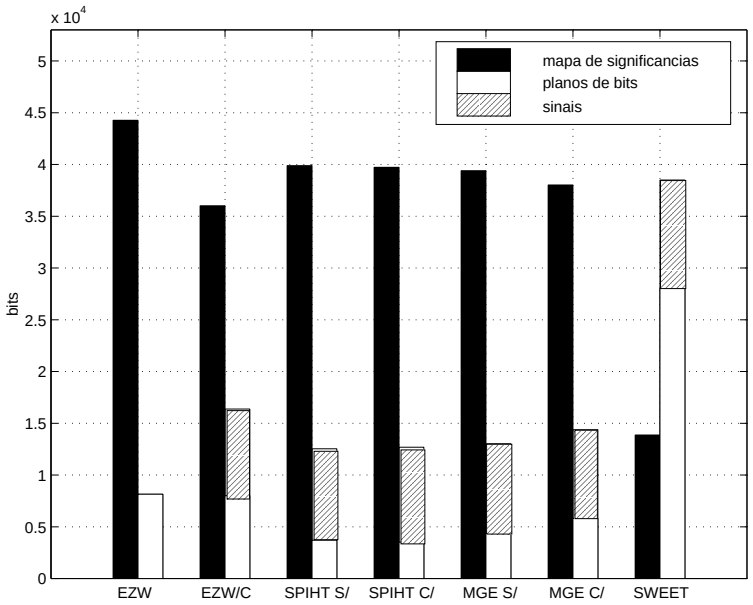


(f)

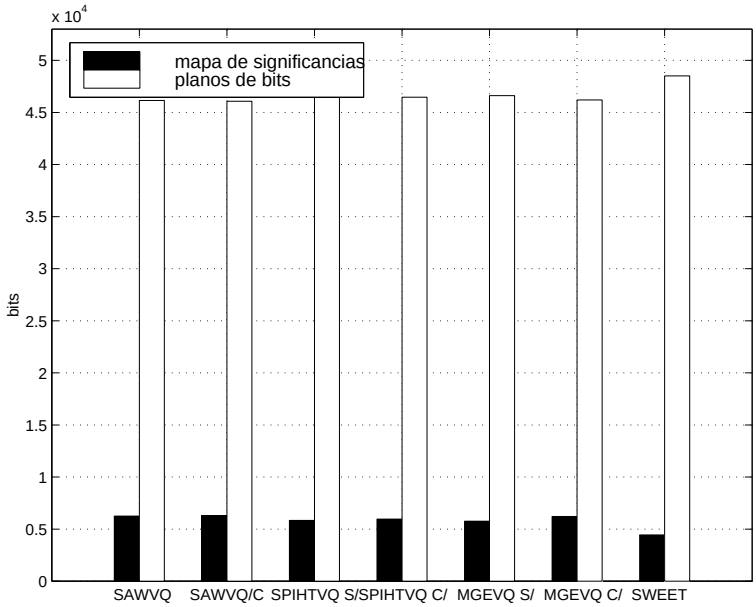


(d)

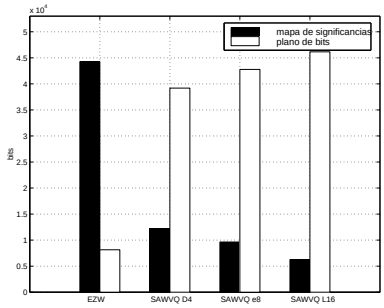
Figura C.15: Para a imagem Gold a 1 bits/pixel, comparação dos: (a) codificadores escalares; (b) codificadores vetoriais; (c) codificadores escalares e vetoriais EZW; (d) codificadores escalares e vetoriais EZW/C; (e) codificadores escalares e vetoriais SPIHT; (f) codificadores escalares e vetoriais MGE; (g) codificadores escalares e vectoriais SWEET


x 10^4
mapa de significancias
planos de bits
sinais
bits
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
EZW
EZW/C
SPIHT S/
SPIHT C/
MGE S/
MGE C/
SWEET

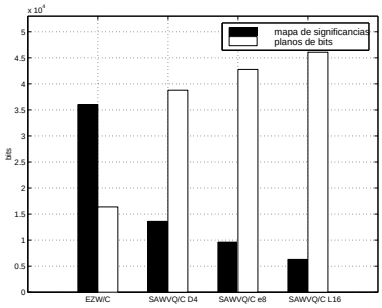
(a)


x 10^4
mapa de significancias
planos de bits
bits
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
SAWVQ
SAWVQ/C
SPIHTVQ S/
SPIHTVQ C/
MGEVQ S/
MGEVQ C/
SWEET

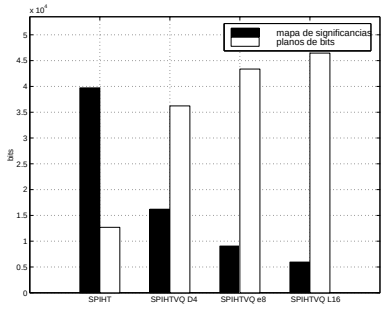
(b)



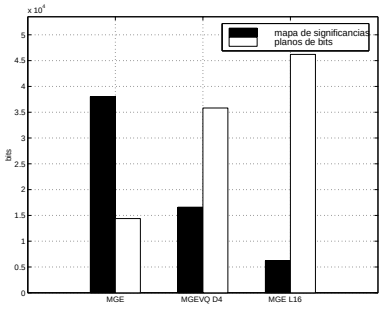
(c)



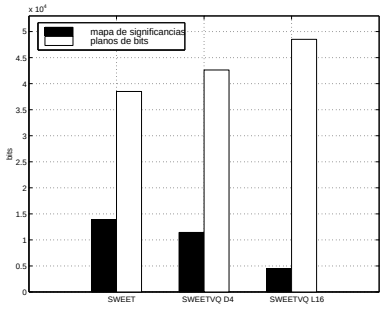
(d)



(e)

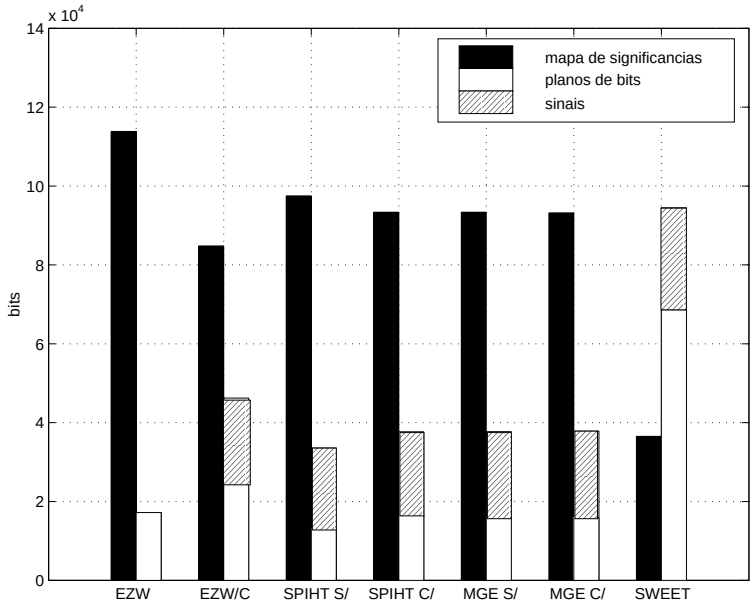


(f)

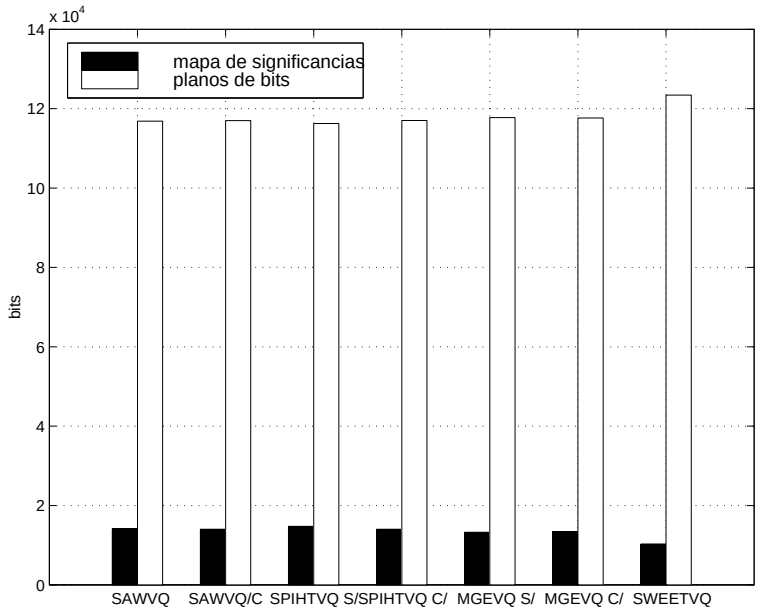


(d)

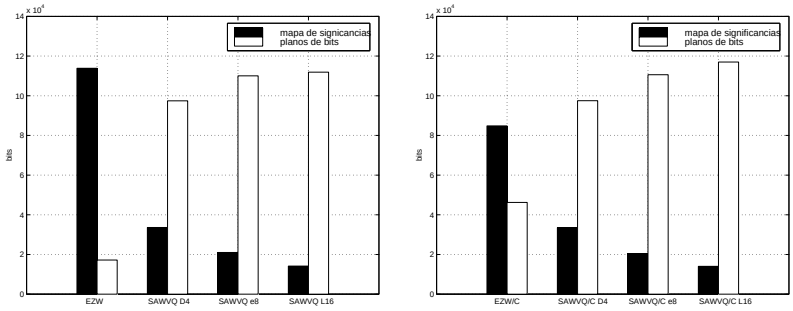
Figura C.16: Para a imagem Zelda a 0,2 bits/pixel, comparação dos: (a) codificadores escalares; (b) codificadores vetoriais; (c) codificadores escalares e vetoriais EZW; (d) codificadores escalares e vetoriais EZW/C; (e) codificadores escalares e vetoriais SPIHT; (f) codificadores escalares e vetoriais MGE; (g) codificadores escalares e vectoriais SWEET


x 10^4
14
12
10
8
6
4
2
0
bits
mapa de significancias
planos de bits
sinais
EZW
EZW/C
SPIHT S/
SPIHT C/
MGE S/
MGE C/
SWEET

(a)

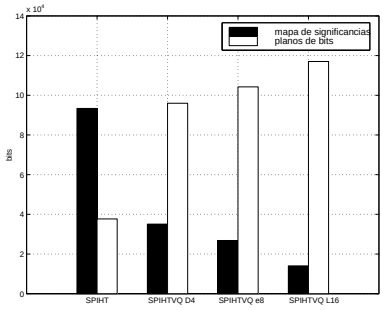

x 10^4
14
12
10
8
6
4
2
0
bits
mapa de significancias
planos de bits
SAWVQ
SAWVQ/C
SPIHTVQ S/
SPIHTVQ C/
MGEVQ S/
MGEVQ C/
SWEETVQ

(b)

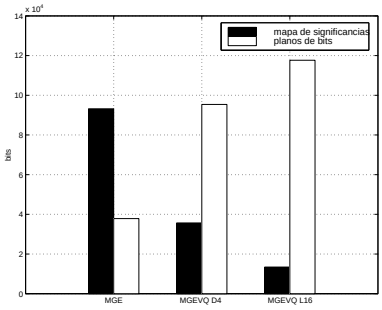


(c)

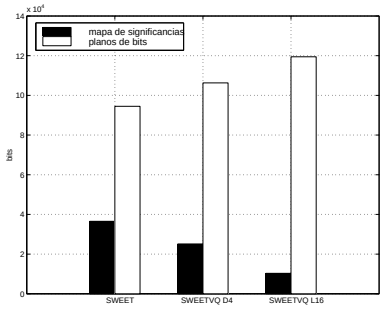
(d)



(e)

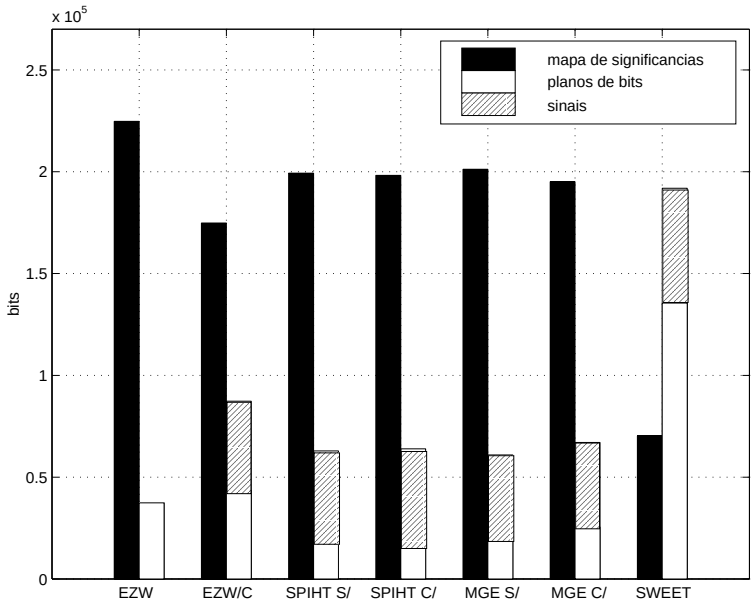


(f)

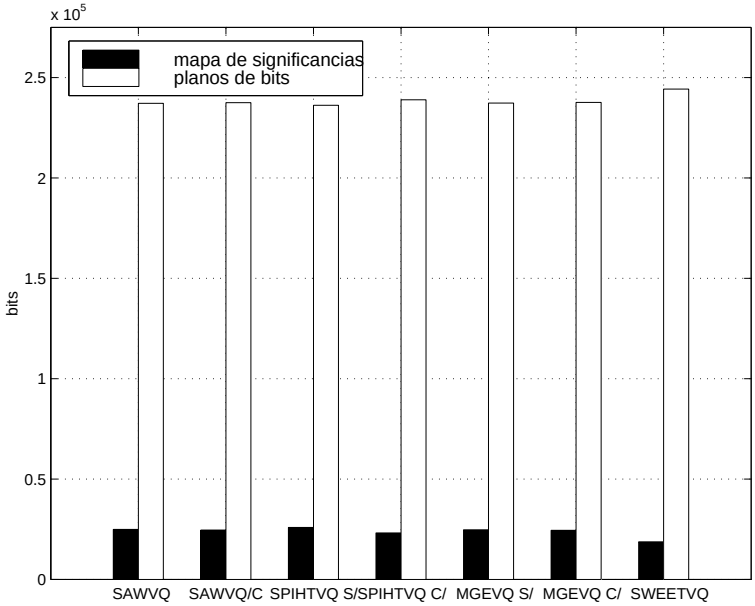


(d)

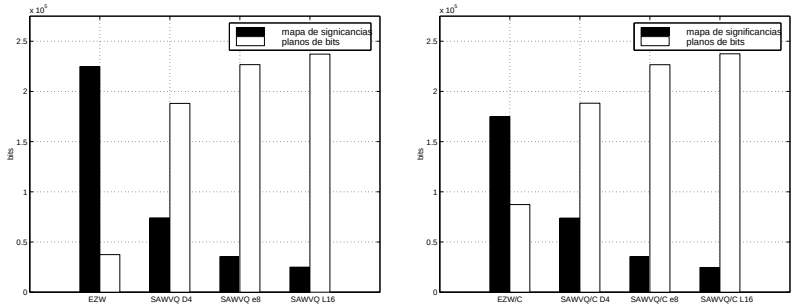
Figura C.17: Para a imagem Zelda a 0,5 bits/pixel, comparação dos: (a) codificadores escalares; (b) codificadores vetoriais; (c) codificadores escalares e vetoriais EZW; (d) codificadores escalares e vetoriais EZW/C; (e) codificadores escalares e vetoriais SPIHT; (f) codificadores escalares e vetoriais MGE; (g) codificadores escalares e vectoriais SWEET


x 10^5
2.5
2
1.5
1
0.5
0
bits
mapa de significancias
planos de bits
sinais
EZW
EZW/C
SPIHT S/
SPIHT C/
MGE S/
MGE C/
SWEET

(a)

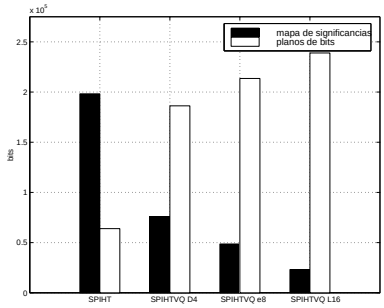

x 10^5
2.5
2
1.5
1
0.5
0
bits
mapa de significancias
planos de bits
SAWVQ
SAWVQ/C
SPIHTVQ S/
SPIHTVQ C/
MGEVQ S/
MGEVQ C/
SWEETVQ

(b)

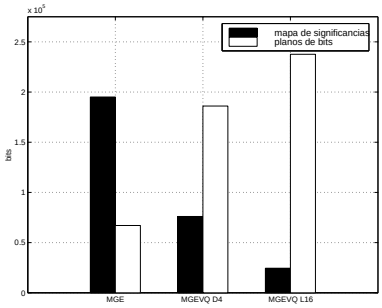


(c)

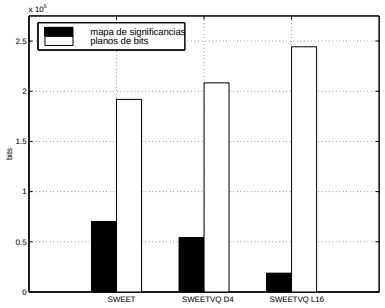
(d)



(e)



(f)



(d)

Figura C.18: Para a imagem Zelda a 1 bits/pixel, comparação dos: (a) codificadores escalares; (b) codificadores vetoriais; (c) codificadores escalares e vetoriais EZW; (d) codificadores escalares e vetoriais EZW/C; (e) codificadores escalares e vetoriais SPIHT; (f) codificadores escalares e vetoriais MGE; (g) codificadores escalares e vectoriais SWEET