COMPRESSÃO DE IMAGENS UTILIZANDO RECORRÊNCIA DE PADRÕES MULTIESCALAS COM CRITÉRIO DE CONTINUIDADE INTER-BLOCOS

EDDIE BATISTA DE LIMA FILHO

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. Murilo Bresciani de Carvalho, D.Sc.

Prof. Abraham Alcaim, Ph.D.

Prof. Weiler Alves Finamore, Ph.D.

Prof. Gelson Vieira Mendonça, Ph.D.

RIO DE JANEIRO, RJ - BRASIL

ABRIL DE 2004

LIMA FILHO, EDDIE BATISTA DE

Compressão de Imagens Utilizando Recorrência de Padrões Multiescalas com Critério de Continuidade Inter-blocos [Rio de Janeiro] 2004

X,120 pp 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia Elétrica, 2004)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1.Compressão de Imagens

2.Casamento de Padrões Multiescalas

3.Quantização Vetorial

4.Otimização Taxa-distorção

5.Side-Match

I.COPPE/UFRJ II.Título (série)

Agradecimentos

Os meus agradecimentos são para todos aqueles que suportaram o meu nervosismo e as minhas ausências durante a elaboração deste trabalho e, de alguma maneira, contribuíram para o sucesso do mesmo. Sem o seu apoio, nada teria sido possível, e eu não teria completado este importante passo para a continuação da minha carreira acadêmica. A todas essas pessoas, o meu muito obrigado.

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

COMPRESSÃO DE IMAGENS UTILIZANDO RECORRÊNCIA DE PADRÕES MULTIESCALAS COM CRITÉRIO DE CONTINUIDADE INTER-BLOCOS

Eddie Batista de Lima Filho

Abril/2004

Orientadores: Eduardo Antônio Barros da Silva
Murilo Bresciani de Carvalho

Programa: Engenharia Elétrica

Este trabalho propõe e analisa o desempenho de um método de compressão baseado em casamento aproximado de padrões multiescalas recorrentes obedecendo a um critério de continuidade inter-blocos, chamado de SM-MMP. A compressão da imagem ocorre através da sua divisão em blocos menores e posterior aproximação destes por vetores presentes em um dicionário, que é construído através de concatenações de expansões e contrações de blocos previamente codificados. Ao processar cada bloco, o SM-MMP cria um dicionário temporário, a partir do dicionário mencionado anteriormente, baseando-se na similaridade dos *pixels* de borda do bloco atual com os dos seus vizinhos superior e esquerdo já codificados. O vetor aproximado é escolhido dentre os elementos deste dicionário temporário. O SM-MMP é baseado no MMP e objetiva melhorar o seu desempenho na compressão de imagens suaves, apresentando melhora considerável na relação taxa×distorção. Os resultados das simulações comprovam a melhoria esperada em imagens suaves, sem haver comprometimento no desempenho para imagens de texto ou mistas.

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

COMPRESSION OF IMAGES BASED ON RECURRENT MULTISCALE PATTERNS WITH INTER BLOCK CONTINUITY CRITERION

Eddie Batista de Lima Filho

April/2004

Advisors: Eduardo Antônio Barros da Silva
Murilo Bresciani de Carvalho

Department: Electrical Engineering

This work proposes and analyzes the performance of a compression method based on approximate pattern matching with scales following an inter block continuity criterion, named SM-MMP. The compression of an image is performed by splitting it in smaller blocks and matching them to vectors from a dictionary, that is built with concatenations of expansions and contractions of previously encoded blocks. During the encoding process of each block, the SM-MMP algorithm generates a temporary dictionary using elements from the dictionary mentioned earlier, based on the similarity of border pixels of the current block with border pixels of his left and upper encoded neighbors. The matched vector is chosen among the elements from the temporary dictionary. The SM-MMP algorithm is based on the MMP and intends to improve its performance in compression of smooth images, presenting remarkable improvements in rate×distortion relation. The simulation results confirm the expected improvement in smooth images, without any performance degradation for text and mixed images.

Sumário

Lista de Figuras

Capítulo 1

Introdução

É notória a tendência atual rumo à digitalização, principalmente no que se refere a conteúdos visuais, pois a utilização desse tipo de informação é encontrada em uma infinidade de áreas, tais como medicina, eletrônica de consumo, geoprocessamento e entretenimento. Como conseqüência desse fato, surge a necessidade de técnicas eficientes e confiáveis para sua representação e seu armazenamento, levando-se em consideração a relação entre a qualidade do conteúdo e o espaço requerido, que nesse caso é consideravelmente alto.

O presente trabalho aborda a vertente de compressão de imagens e vídeo, propondo algoritmos capazes de reduzir o espaço necessário para o seu armazenamento e ainda manter uma boa qualidade para o conteúdo. Os resultados apresentados referem-se a um novo algoritmo chamado aqui de SM-MMP (*Side-Match Multidimensional Multiscale Parser*), classificado como um esquema de compressão com perdas, cuja principal característica é a excelente performance em imagens mais complexas, como as de texto, sem deixar de lado a preocupação com imagens suaves, apresentando resultados significativos para estas últimas.

O SM-MMP é baseado no já conhecido MMP e utiliza todos os seus conceitos, tais como a recorrência de padrões multiescalas e a otimização da árvore de segmentação, apresentando novas formas para a realização dessas tarefas e introduzindo um critério de continuidade inter-blocos para melhorar a performance na codificação de imagens suaves, principal deficiência do MMP padrão em relação aos codificadores baseados em *Wavelets* (e.g. SPIHT). Outro fato importante é a característica universal apresentada pelo SM-MMP, pois o mesmo é capaz de codificar

com ótima qualidade um grande espectro de imagens, indo desde as mais suaves, como *LENA* e *F-16*, até as mais complexas, ressaltando-se como exemplos páginas de periódicos repletas de texto.

Os resultados apresentados neste trabalho atestam tudo aquilo que foi exposto e ressaltam a importância do desenvolvimento de algoritmos baseados em casamento de padrões multiescalas recorrentes em substituição aos que utilizam *Wavelets*, pois estes últimos baseiam-se na asserção de que as imagens processadas são suaves (a maior parte da energia está concentrada em freqüências mais baixas), ou seja, possuem pouco conteúdo de alta freqüência, o que pode limitar a sua utilização. Isto reforça a importância de prosseguir no desenvolvimento de uma classe completamente nova de algoritmos para compressão de imagens e vídeo, com comportamento universal, adaptativo e não fortemente baseado em asserções de suavidade ou na suposição a respeito de estruturas presentes na imagem.

O capítulo 2 deste trabalho apresenta toda a teoria matemática básica necessária para a compreensão e o desenvolvimento de algoritmos de compressão com perdas, introduzindo os conceitos mais importantes utilizados durante este trabalho.

O capítulo 3 proporciona uma introdução ao problema da compressão de dados, apresentando os principais conceitos utilizados e os tipos de algoritmo, com uma explanação, um pouco mais detalhada, dos esquemas de compressão com perdas, classe na qual se posiciona o SM-MMP.

O capítulo 4 apresenta os critérios de avaliação objetivos aplicados a esquemas de compressão, ressaltando os mais importantes, e comenta a abordagem subjetiva para a avaliação de qualidade, esclarecendo seus limitantes. Além disso, comenta o sistema visual humano e mostra que o desenvolvimento de algoritmos de compressão com perdas, como o apresentado neste trabalho, é completamente válido e pode ainda fazer uso das características desse sistema para que as distorções inseridas não afetem, de forma agressiva, a qualidade percebida na imagem.

O capítulo 5 refere-se à implementação particular do MMP realizada, ressaltando as principais diferenças com relação ao algoritmo padrão e apresentando uma descrição com todos os passos necessários para o devido processamento. É necessário atenção especial quanto ao tipo de divisão de blocos, formação do dicionário inicial e transformação de escala utilizados. Resultados de simulações para comparação são

disponibilizados.

O capítulo 6 apresenta um novo algoritmo para a otimização da árvore de segmentação, chamado de algoritmo de otimização intermediário, ou RDI-MMP. O seu desenvolvimento ocorreu principalmente devido às restrições ocasionadas pela inserção do critério de continuidade inter-blocos (*Side-match*). São descritas suas características e vantagens em relação às implementações anteriores, bem como apresentados resultados de simulações.

O capítulo 7 aborda o objeto principal deste trabalho, que consiste na concepção e implementação do critério de continuidade inter-blocos, principal responsável pelo aumento de desempenho do SM-MMP, com relação aos seus antecessores, na codificação de imagens suaves. São apresentadas suas características e uma descrição detalhada do algoritmo, além de resultados para vários tipos de imagens.

O capítulo 8 trata de uma nova técnica para a atualização de dicionário, baseada em deslocamentos do bloco a ser incluído. Essa técnica, chamada aqui de super-atualização, é essencial para o perfeito funcionamento do SM-MMP, tornando imperativa a sua implementação em qualquer codificador desta classe. São realizadas comparações com relação à técnica de atualização anterior e comentadas suas vantagens e desvantagens. Para que seja possível um entendimento completo da sua importância, são apresentados resultados para comparação com aqueles obtidos pelo SM-MMP sem super-atualização do dicionário.

Por último, o capítulo 9 descreve todas as conclusões do trabalho, propõe novas abordagens para a implementação de alguns módulos do algoritmo, tais como o critério de continuidade e a atualização do dicionário, e ressalta os problemas em aberto.

Capítulo 2

Premissas matemáticas e teoria taxa-distorção

O campo da ciência que proporciona uma descrição quantitativa de informação, abordando aspectos como o compromisso entre taxa e distorção, capacidade de canal e modelos matemáticos, é chamado de Teoria da Informação [7, 4]. O pesquisador responsável pelas primeiras definições nesse campo e pela sua concretização foi *Claude Elwood Shannon*, um engenheiro eletricista dos Laboratórios *Bell*.

2.1 Introdução à teoria da informação

Shannon definiu, em [7], uma quantidade chamada de auto-informação, que consiste na medida de incerteza de um evento [4]. Supondo-se a existência de um evento A [4] e considerando-se uma medida de probabilidade $P(A)$, então a auto-informação associada ao evento A é dada por:

$$i\left(A\right) = \log_b \left(\frac{1}{P\left(A\right)}\right), \tag{2.1}$$

onde b define a unidade na qual a auto-informação é obtida; se b for igual a 2, a unidade é bits.

Ao se avaliar o comportamento do logaritmo, se um evento tem probabilidade baixa, a informação associada é alta; caso contrário, a informação associada é baixa, o que é bastante coerente. Por exemplo, se $P(A) = 1$, a auto-informação é zero, pois

o evento tem probabilidade de 100% de ocorrer, ou seja, já se sabe que o mesmo vai acontecer (não há incerteza), o que leva a nenhuma informação agregada.

Seja um experimento $\mathcal{S}$, com um conjunto de eventos A_i associados tais que

$$\bigcup A_i = S, \tag{2.2}$$

onde S é o espaço amostral [4]. Os eventos A_i serão independentes dois-a-dois se:

$$P\left(A_i \cap A_j\right) = P\left(A_i\right).P\left(A_j\right),\ i \neq j. \tag{2.3}$$

Neste caso, a auto-informação média associada ao experimento $\mathcal{S}$ é dada pela fórmula:

$$H\left(\mathcal{S}\right) = \sum_{i=1}^{n} P\left(A_i\right).i\left(A_i\right) = -\sum_{i=1}^{n} P\left(A_i\right).\log_b\left(P\left(A_i\right)\right), \tag{2.4}$$

também chamada de entropia associada ao experimento.

No jargão de Teoria da Informação, o experimento $\mathcal{S}$ é chamado de *fonte*, os eventos A_i de *símbolos* (ou *letras*) e o conjunto de símbolos $\mathcal{A} = \{A_i\}$ de *alfabeto da fonte*. Para uma fonte qualquer $\mathcal{S}$ com alfabeto $\mathcal{A} = \{1, 2, ..., m\}$, que gera uma seqüência $\{X1, X2, ...\}$, a entropia é dada por:

$$H\left(\mathcal{S}\right) = \lim_{n\to\infty}\left(\frac{1}{n}G_n\right), \tag{2.5}$$

onde

$$\begin{aligned} G_n = -\sum_{i_1=1}^{m}\sum_{i_2=1}^{m}\ldots\sum_{i_n=1}^{m} P\left(X_1 = i_1, X_2 = i_2, \ldots,\right. \\ \left. X_n = i_n\right).\log_b\left(P\left(X_1 = i_1, X_2 = i_2, \ldots, X_n = i_n\right)\right). \end{aligned}$$

Se cada elemento da seqüência for independente e identicamente distribuído (iid), a equação (2.5) reduz-se a (2.4). Como para a maioria das fontes isso não é verdade, a equação (2.5) é chamada de entropia e a equação (2.4) de entropia de primeira ordem.

Quando se analisam esquemas de compressão que inserem perdas, depreende-se que os alfabetos de reconstrução e da fonte podem ser distintos, o que leva à necessidade de se quantificar as relações de informação entre duas variáveis aleatórias que assumem valores de dois alfabetos diferentes. Uma medida de relação entre essas duas variáveis, muito utilizada, é a entropia condicional. Supondo-se uma variável aleatória $X = \{x_0, x_1, ..., x_{n-1}\}$, que assume valores do alfabeto da fonte, e outra $Y = \{y_0, y_1, ..., y_{m-1}\}$, que assume valores do alfabeto de reconstrução, a entropia condicional é dada pela fórmula:

$$H\left(X|Y\right) = -\sum_{i=0}^{n-1}\sum_{j=0}^{m-1} P\left(x_i|y_j\right).P\left(y_j\right).\log_b\left(P\left(x_i|y_j\right)\right), \tag{2.6}$$

onde $H(X|Y)$ é a entropia de X dado que Y ocorreu. A entropia condicional pode ser interpretada como a incerteza sobre a variável X dado que se conhecem os valores de reconstrução que Y assume [5]. Esse conhecimento adicional de Y reduz a incerteza sobre X.

Outra medida de incerteza sobre duas variáveis aleatórias, também muito utilizada, é a informação mútua média. Supondo-se as mesmas variáveis aleatórias X e Y definidas para a entropia condicional, a informação mútua média é dada por:

$$I\left(X;Y\right) = \sum_{i=0}^{n-1}\sum_{j=0}^{m-1} P\left(x_i|y_j\right).P\left(y_j\right).\log_b\left(\frac{P\left(x_i|y_j\right)}{P\left(x_i\right)}\right) = H\left(X\right) - H\left(X|Y\right). \tag{2.7}$$

A informação mútua média pode ser interpretada como a redução na incerteza de X devido ao conhecimento de Y [4].

2.2 Teoria taxa-distorção

A compactação obtida para um conjunto de dados, utilizando-se esquemas de compressão que inserem perdas, pode ser aumentada, desde que seja aceita uma quantidade maior de distorção, o que muitas vezes acontece. Entretanto, como decorrência desse fato, surge a necessidade de avaliação da quantidade de bits necessária para a representação desses dados após a compactação, de modo que se possa decidir se essa redução compensa o aumento da distorção.

O objetivo principal, para qualquer esquema de compressão com perdas, é provocar uma pequena distorção e comprimir à menor taxa possível, obedecendo sempre a um compromisso entre as duas variáveis. Os casos extremos ocorrem quando não se transmite qualquer informação, ou seja, quando a taxa é zero, e quando se transmite toda a informação, o que leva a uma distorção zero. O estudo das situações entre estes dois extremos é chamado de teoria taxa-distorção [4, 5, 8], cujo foco está no desenvolvimento de limites de desempenho para a relação entre taxa e distorção de uma determinada fonte. Na verdade, o que se pretende é obter a taxa explicitamente definida como função da distorção, para uma dada medida de distorção, o que nem sempre é possível.

A função taxa-distorção, ou $R(D)$, especifica a menor taxa média R na qual a saída de uma fonte pode ser codificada, mantendo-se uma distorção média menor ou igual a D [5]. A função $R(D)$ define o desempenho possível de ser atingido por qualquer código de compressão, ou seja, a desigualdade $R_{código} \geq R(D)$ é sempre satisfeita.

Em [8], que Shannon publicou em 1959, foi demonstrado que a taxa para uma dada distorção, supondo-se $X = \{x_0, x_1, ..., x_{n-1}\}$ que assume valores do alfabeto da fonte (imagem original) e $Y = \{y_0, y_1, ..., y_{m-1}\}$ que assume valores do alfabeto de reconstrução (imagem reconstruída), é obtida por:

$$R\left(D\right) = \min_{\{P(x_i|y_i)\}\in\Gamma} I\left(X;Y\right), \tag{2.8}$$

onde

$$\Gamma = \{\{P\left(x_i|y_i\right)\} \ tal \ que \ D\left(\{P\left(x_i|y_i\right)\}\right) \leq D^*\} \tag{2.9}$$

e D^* é a distorção alvo.

Para que a função taxa-distorção seja obtida, é mais simples encontrar um limite inferior para a informação mútua média e mostrar que o mesmo pode ser atingido. Tal desenvolvimento foge ao escopo deste trabalho, porém, um exemplo será mostrado, envolvendo uma fonte *Gaussiana* sem memória.

Supondo-se uma medida de distorção dada pelo erro quadrático,

$$d(x,y) = (x-y)^2, \tag{2.10}$$

e um distorção limite D, o que leva a

$$\frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2 \leq D, \tag{2.11}$$

pode-se provar que a função taxa-distorção é escrita como:

$$R(D) = \begin{cases} \frac{1}{2}\log_{10}\left(\frac{\sigma^2}{D}\right) & para\ D \leq \sigma^2, \\ 0 & para\ D > \sigma^2, \end{cases} \tag{2.12}$$

o que é mostrado na Figura 2.1 com $\sigma^2 = 1$.

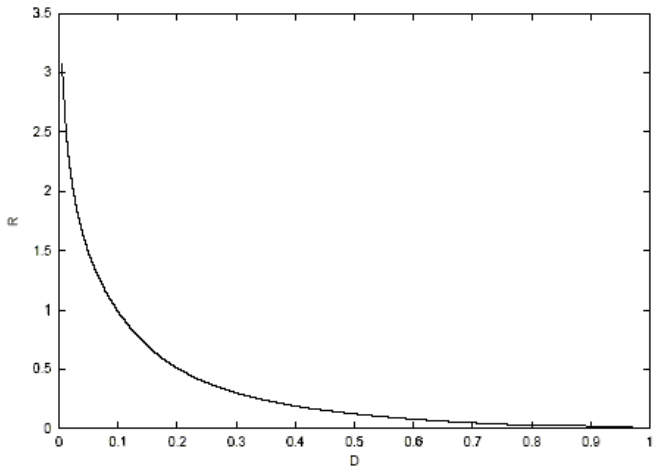


Figura 2.1: Função $R(D)$ para uma fonte Gaussiana sem memória.

Quando $R(D) \to \infty$, $D \to 0$, pois cada saída possível é um número real necessitando de precisão infinita para ser descrito com distorção zero [1]. Se a fonte não for Gaussiana, a obtenção da função $R(D)$ nem sempre é possível. Nesses casos, pode se recorrer ao algoritmo desenvolvido por Arimoto e Blahut [10, 9]. Vale ressaltar que $R(0)$ é exatamente a entropia da fonte.

Mesmo que se consiga encontrar a função $R(D)$ para uma dada fonte, pode não ser possível obter-se um método de compressão capaz de comprimir a saída

da mesma utilizando exatamente $R(D)$ bits e com distorção D. Entretanto, para $D = 0$ e fontes discretas, existem métodos de compressão capazes de atingir $R(0)$, pelo menos quando o número de símbolos da fonte tende ao infinito.

Capítulo 3

A compressão de dados

Os algoritmos de compressão são aplicados basicamente em padrões de compressão, tais como MPEG [21] e JPEG [20], com o objetivo de reduzir o número de bits necessário para a representação de uma dada informação.

A razão da sua necessidade reside no fato de que cada vez mais informações estão sendo geradas e utilizadas em formato digital, necessitando de espaço para o seu armazenamento ou banda para sua transmissão, que são recursos finitos e economicamente dispendiosos.

No caso de dados multimídia (e.g. imagens, vídeo e áudio), o espaço/banda necessário é muito grande, principalmente quando se trata de informações visuais, o que ressalta ainda mais a importância do desenvolvimento de técnicas de compressão mais eficazes e adequadas aos tipos de dado em questão.

3.1 O conceito de compressão

Comprimir um dado significa reescrevê-lo numa forma compacta, que utiliza um número menor de bits para a sua representação. Essas formas compactas são criadas através da identificação e utilização de estruturas comuns ou repetitivas presentes nos referidos dados, permitindo, por exemplo, sua reutilização, ou ainda predição, através de cálculos estatísticos.

Quando alguma técnica ou algoritmo de compressão é analisado, na verdade trabalha-se com dois elementos diferentes: aquele que processa os dados de entrada X e gera uma representação $\hat{X}$ que necessita de um número menor de bits para ser

armazenada, chamado de codificador, e aquele que processa a informação comprimida $\hat{X}$ e gera a imagem reconstruída Y, chamado de decodificador. O esquema completo é mostrado na Figura 3.1, tomando como exemplo uma imagem em níveis de cinza. Geralmente, quando se faz referência a um algoritmo de compressão, está implícita a existência dos dois elementos.

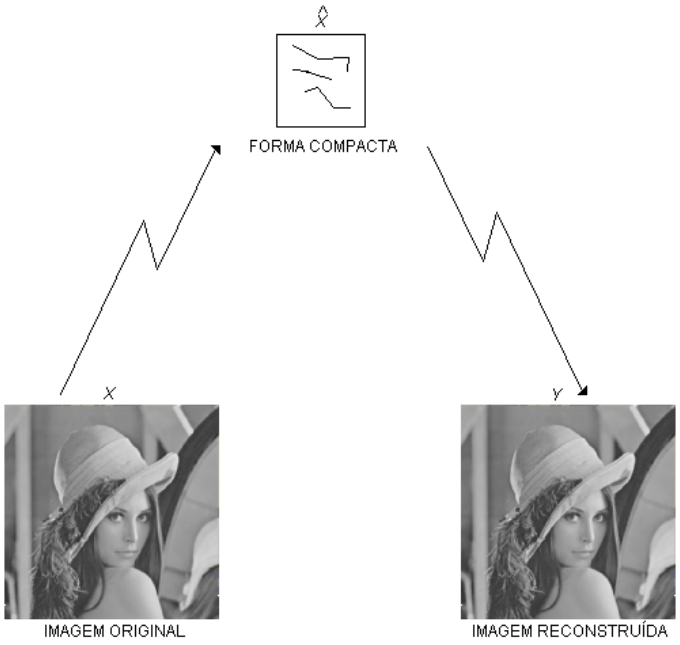


Figura 3.1: Compressão e reconstrução de uma imagem.

3.2 Técnicas de compressão

Dependendo das necessidades presentes na reconstrução (e.g. alta ou baixa qualidade da imagem reconstruída), os esquemas de compressão podem ser divididos em duas classes: com perdas e sem perdas. Nos algoritmos com perdas, que geralmente alcançam as maiores taxas de compressão, o resultado é uma imagem reconstruída Y diferente de X, devido ao fato de ter havido perda de informação (exatamente o fator responsável pala alta taxa de compactação) durante a compressão. Já nos algoritmos sem perdas, a imagem reconstruída Y é exatamente igual à imagem original X, sem qualquer perda de informação. Cada um dos esquemas encontra áreas específicas de aplicação, e sua escolha depende basicamente das características dos dados processados e utilização pretendida para os mesmos.

Os esquemas de compressão sem perdas são geralmente empregados em aplicações nas quais não pode ser tolerada qualquer diferença entre os dados originais e os reconstruídos. Por exemplo, arquivos de texto, nos quais qualquer diferença pode ocasionar uma mudança completa de sentido nas informações, e dados para pós-processamento (e.g. imagens de satélites, nas quais um pós-processamento pode resultar em acentuação das diferenças, ou imagens médicas, nas quais uma mínima diferença pode ocasionar um diagnóstico errôneo) necessitam de um esquema de compressão sem perdas. Outra utilização bastante comum para esse tipo de algoritmo é como uma camada de codificação auxiliar nos algoritmos de compressão com perdas. Exemplos de algoritmos de compressão sem perdas são o código de *Huffman* [5], o codificador aritmético [22] e o código de *Lempel-Ziv* [5]. A taxa resultante de tais métodos tende a aproximar a entropia da fonte ($R(0)$), ou seja, a quantidade média mínima de bits necessária para se codificar a saída da mesma. Devido a isso, os métodos de compressão sem perdas também são conhecidos como codificadores de entropia.

Alguns algoritmos de compressão sem perdas requerem o conhecimento de um modelo estatístico para a fonte (e.g. código de *Huffman* e codificador aritmético), porém, com respeito ao algoritmo de *Lempel-Ziv*, essa necessidade não existe, dando ao mesmo um comportamento universal. Além disso, quando o número de símbolos codificados tende ao infinito, sua taxa de código tende para a entropia da fonte.

Os esquemas de compressão com perdas, por sua vez, levam, inexoravelmente, a alguma perda de informação, resultando em uma reconstrução não exata da informação original. Como conseqüência disso, obtêm, geralmente, maiores taxas de compressão quando comparados aos esquemas sem perdas. A distorção resultante depende da fidelidade intencionada para o dado reconstruído. Algumas aplicações que toleram perdas e poderiam utilizar-se de tais técnicas de compressão são, por exemplo, transmissão de voz e sinais de vídeo para broadcasting (o olho humano acaba integrando a imagem e o erro só é perceptível em larga escala). Exemplos de esquemas de compressão com perdas são o JPEG [20] e o JPEG2000 [18], ambos desenvolvidos para imagens e baseados em transformada; o primeiro é baseado na DCT - *Discrete Cosine Transform* e o segundo na DWT - *Discrete Wavelet Transform*. É interessante observar que, uma vez ocorrida a perda de informação, a

forma original dos dados não pode ser recuperada.

Com relação aos algoritmos de compressão com perdas, não há um método universal que aproxime a curva $R(D)$ para qualquer fonte, como ocorre nos sem perdas. Além disso, mesmo as soluções aproximadas tendem a ser extremamente complexas [1]. Devido a isso, adotam-se soluções em dois ou três passos.

Na solução em dois passos, cria-se uma versão $\mathcal{X}$ da fonte X, com menor entropia, ou seja, $H(\mathcal{X}) < H(X)$. Este é o primeiro passo e é chamado de quantização, caracterizado pelo mapeamento $\mathcal{X} = Q(X)$. No segundo passo, chamado de codificação de entropia, aplica-se um algoritmo de compressão sem perdas a $\mathcal{X}$. A quantização pode ser escalar ou vetorial, dependendo do resultado pretendido e da complexidade esperada. Os quantizadores vetoriais apresentam melhor desempenho que os escalares, o que aumenta com o número de dimensões. É importante ressaltar que a etapa responsável pela alta compactação dos dados é a quantização.

Na solução em três passos, a primeira tarefa executada é uma transformação dos dados, objetivando-se encontrar um modelo estatístico mais simples para fonte. Por exemplo, a DCT gera um bloco de coeficientes de transformada a partir de um bloco de pixels. Como a maioria das imagens reais tem pouco conteúdo de alta freqüência, a maior parte da energia estará concentrada no coeficiente DC e nos coeficientes AC de freqüências mais baixas (compactação de energia), o que pode ser facilmente identificado e trabalhado para permitir uma compressão mais eficaz. O segundo passo consiste na quantização, com a qual a entropia da fonte transformada é significativamente reduzida. No terceiro passo, uma codificação de entropia é aplicada, diminuindo a redundância dos dados quantizados.

O algoritmo desenvolvido neste trabalho pertence à classe dos algoritmos de compressão com perdas e une as etapas de transformação, quantização e codificação de entropia em um único passo, atuando de forma adaptativa na compressão dos dados. Apesar disso, vale ressaltar que o algoritmo desenvolvido é capaz de comprimir os dados sem perdas, desde que sua distorção alvo seja ajustada para zero (ver seção 5.1).

Quando um novo método surge, é necessário posicioná-lo em relação aos demais existentes. Para que o desempenho do novo algoritmo desenvolvido neste trabalho seja medido e comparado ao de outros, é necessária a definição de critérios

e métricas, o que será discutido no próximo capítulo.

Capítulo 4

Avaliação de eficácia e medidas de fidelidade

Foi mostrado que alguns esquemas de compressão podem ocasionar perdas, como é o caso do algoritmo desenvolvido neste trabalho, o que permite atingirem-se taxas de compressão muito altas. Entretanto, tais perdas modificam os dados processados de forma permanente, o que pode invalidar a utilização dos mesmos. Devido a isso, é preciso entender como o olho humano percebe essas distorções, de modo que se possa avaliar a utilização e os resultados de tais algoritmos e projetar estratégias mais eficazes e adequadas ao usuário final: o ser humano.

4.1 O sistema visual humano

O olho humano é um objeto em forma de globo, com lentes em sua parte dianteira que projetam, na retina, os objetos visualizados. A retina contém dois tipos de receptores: os cones e os bastonetes. Os bastonetes são mais sensíveis à luz que os cones, sendo os responsáveis pela maior parte da visão quando o ambiente apresenta pouca luminosidade. Quanto aos cones, existem três tipos, cada um sensível a diferentes comprimentos de onda do espectro visível. O pico de sensibilidade de cada tipo de cone situa-se na região do vermelho, azul ou verde.

O olho humano é sensível a uma enorme escala de intensidades de luz, porém, num dado instante, não é possível para o mesmo perceber tal escala de brilho em sua totalidade. Na verdade, o olho se adapta ao brilho médio da cena [6]. Logo, a

extensão da escala que o olho pode perceber é muito menor que o total. Esse fato é demonstrado na Figura 4.1, onde B_a é o nível de adaptação ao brilho e B_b é o nível abaixo do qual todos os estímulos são percebidos como negro.

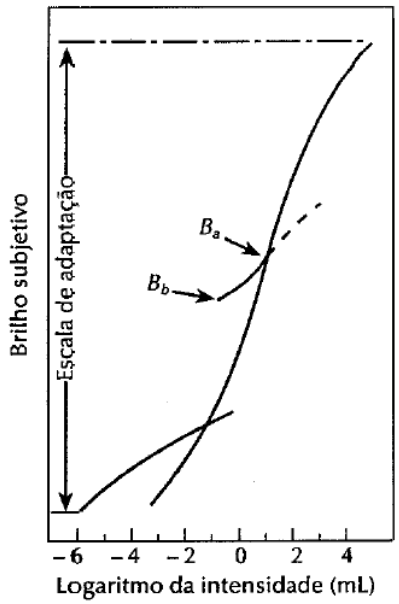


Figura 4.1: Escala de brilho subjetivo (adaptado de [6]).

Além das características já apresentadas, evidências experimentais indicam que o brilho percebido pelo sistema visual humano é uma função logarítmica da intensidade de luz incidente no olho [6], e sabe-se também que este último se comporta como um filtro passa-baixas espacial [5]. De posse dessas informações, é possível desenvolver-se o modelo mostrado na Figura 4.2.

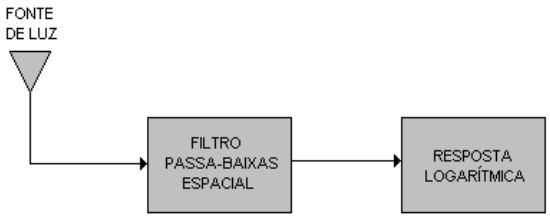


Figura 4.2: Modelo da visão humana.

A principal conclusão extraída, com respeito a tudo que foi apresentado neste capítulo, consiste no fato de que o cérebro não percebe tudo aquilo que o olho

vê. Esta constatação ratifica o desenvolvimento de algoritmos de compressão que inserem perdas e pode ser utilizada para se projetar sistemas de compressão tais que a distorção introduzida (e.g. excluir conteúdo de alta freqüência) não seja perceptível pelo ser humano, o que leva a uma alta taxa de compactação sem provocar grandes diferenças visíveis na imagem.

4.2 Métricas para avaliação

A melhor avaliação para um método de compressão seria aquela feita pelo usuário final, que analisaria a qualidade da reconstrução e proporcionaria um *feedback* para o projetista. Na prática, porém, isso usualmente não é possível, principalmente quando o usuário final é o ser humano, pois é bastante difícil incorporar a resposta humana a modelos matemáticos. Além disso, há dificuldade em reportar objetivamente os resultados, que podem variar de pessoa para pessoa. Por exemplo, um mesmo resultado pode ser considerado excelente por um observador e apenas aceitável por outro. Entretanto, isso pode ser amenizado, recrutando-se uma quantidade muito grande de observadores e esperando que, na média, as diferenças se cancelem.

Por ser uma solução extremamente dispendiosa, principalmente em termos de tempo, a avaliação subjetiva é geralmente deixada de lado e acaba sendo mais interessante avaliar, de forma objetiva, o quanto a imagem reconstruída é similar à original [5]. É claro que a qualidade subjetiva da imagem reconstruída é uma figura de mérito muito importante, porém, devido a todos os problemas apresentados, será apenas comentada durante o texto.

Um algoritmo de compressão pode ser avaliado objetivamente de diversas maneiras, como, por exemplo, medindo-se sua complexidade, rapidez, quantidade de memória necessária, taxa de compressão ou fidelidade à imagem original. A métrica escolhida depende basicamente do esquema utilizado e das características de uma determinada aplicação. Entretanto, dificilmente apenas um fator é levado em consideração, sendo geralmente utilizado um subconjunto. Por exemplo, para esquemas de compressão com perdas, não é suficiente ressaltar apenas a taxa de compressão sem avaliar a qualidade do dado reconstruído. Nesta seção, serão discutidos

os critérios mais importantes e também os utilizados durante o trabalho.

Com certeza, ao se avaliar um algoritmo de compressão, a primeira pergunta que surge é quão menor a imagem compactada está em relação à original. Essa medida é chamada de razão de compressão (CR - *Compression Ratio*) e é dada por:

$$CR = \frac{NBIO - NBIC}{NBIO}.100\%, \tag{4.1}$$

onde $NBIO$ é o Número de Bytes da Imagem Original e $NBIC$ é o Número de Bytes da Imagem Compactada. O valor calculado indica que o espaço ocupado pelo dado compactado é $x\%$ menor que o ocupado pela imagem original.

Uma outra maneira de se avaliar a taxa de compressão de uma imagem consiste no cálculo do número médio de bits (*bpp* - *Bits Per Pixel*) necessário para se representar um pixel, muito conhecido também como taxa (R - *Rate*). O seu cálculo pode ser escrito como:

$$R = \frac{8.TAICB}{NPIO}bpp, \tag{4.2}$$

onde $TAICB$ é o Tamanho do Arquivo da Imagem Compactada, em Bytes, e $NPIO$ é o Número de Pixels da Imagem Original.

Quando se trabalha com compressão com perdas, o dado reconstruído é diferente do original, como já foi comentado. Para que a eficácia do algoritmo seja avaliada, é necessário também quantificar essa diferença, comumente chamada de distorção.

Para se realizar a medida do erro médio do dado reconstruído, o critério mais utilizado é o erro quadrático médio, dado por:

$$\sigma_d^2 = \frac{1}{n}\sum_{i=1}^{n}\left(x_i - \hat{x}_i\right)^2, \tag{4.3}$$

onde x_i é o dado original, $\hat{x}_i$ o reconstruído e n o número total de elementos.

Se a figura de mérito intencionada é o erro relativo ao sinal original, uma das medidas mais utilizadas é a relação sinal ruído (SNR - *Signal to Noise Ratio*), que representa a razão entre o valor quadrático médio da fonte e o erro quadrático médio, dada por:

$$SNR\,(dB) = 10.log_{10}\left(\frac{\sigma_x^2}{\sigma_d^2}\right). \tag{4.4}$$

No presente trabalho, a medida objetiva adotada será a relação sinal ruído de pico ($PSNR$ - *Peak Signal to Noise Ratio*), dada pela fórmula:

$$PSNR\,(dB) = 10.log_{10}\left(\frac{x_{peak}^2}{\sigma_d^2}\right) = 10.log_{10}\left(\frac{255^2}{\frac{1}{m.n}\sum_{i=1}^{m}\sum_{j=1}^{n}\left(x_{i,j} - \hat{x}_{i,j}\right)^2}\right), \tag{4.5}$$

onde m é o número de colunas da imagem, n o número de linhas, $x_{i,j}$ os elementos da imagem original, $\hat{x}_{i,j}$ os elementos da imagem reconstruída, x_{peak}^2 o quadrado do maior valor de pixel da imagem e σ_d^2 o erro quadrático médio. A $PSNR$ mostra o erro relativo ao valor de pico do sinal analisado.

Todos os gráficos apresentados no decorrer deste trabalho mostrarão a $PSNR$ da imagem reconstruída em relação à taxa de compressão, dada em *bpp*.

Capítulo 5

O algoritmo MMP

Em [1], foi introduzido um novo algoritmo para compressão de imagens, baseado em recorrência de padrões multiescalas: o *Multidimensional Multiscale Parser*, ou MMP. Esse algoritmo tenta realizar casamentos entre um vetor $X_j^{m,n}$ a ser codificado, de dimensões $m \times n$ ($linhas \times colunas$), e vetores $S_i^{k,l}$ previamente codificados e agrupados em um dicionário $D^{k,l}$, podendo estes últimos ter dimensões diferentes das de $X_j^{m,n}$. Isso é possível através de uma transformação de escala. Se o casamento não obedecer a um critério de máxima distorção previamente especificado, o bloco é dividido em dois outros iguais e a análise reinicia.

A técnica de casamento com escalas (vetores com dimensões diferentes) é realmente mais vantajosa que o casamento sem escalas, como foi provado em [1] através de um estudo realizado utilizando-se vetores *Gaussianos*. Mais explicitamente, foi demonstrado que a probabilidade de casamento entre vetores *Gaussianos* com escalas diferentes e sujeita a um critério de fidelidade pode ser maior que a sem escalas, melhorando o desempenho do sistema.

Uma característica importante do MMP consiste no fato do mesmo não realizar qualquer processamento baseado em suposições sobre a imagem (e.g. imagem é suave), como nos algoritmos que utilizam *Wavelets*. Na verdade, tudo é feito de forma adaptativa, sem qualquer pressuposição. Isso confere ao MMP um comportamento que pode ser chamado de universal, levando-o a comprimir, com boa qualidade, um amplo espectro de imagens, inclusive as com grande conteúdo de alta freqüência (imagens mistas ou de texto), como demonstrado na seção 5.2.

5.1 Descrição e implementação do algoritmo

Com respeito ao seu funcionamento, o MMP desenvolvido neste trabalho comprime imagens através de sua divisão em blocos $X_0^{m,n}$, nesta implementação com dimensões 8×8 ($m = 8$ e $n = 8$), e posterior aproximação por algum elemento $S_i^{m,n}$ presente em seu dicionário $D^{m,n}$, que é construído durante todo o processo de compressão.

Em essência, estipula-se uma distorção alvo δ e o algoritmo tenta encontrar um elemento do dicionário que represente cada bloco de entrada com distorção média menor ou igual a δ, ou seja:

$$\left\| X_j^{m,n} - S_i^{m,n} \right\|^2 \leq \delta^2.m.n, \tag{5.1}$$

onde m e n são as dimensões do bloco, $X_j^{m,n}$ é o bloco original e $S_i^{m,n}$ a aproximação avaliada; caso não consiga, o bloco de entrada é dividido em dois outros blocos iguais e a procura reinicia, como mostrado na Figura 5.1.

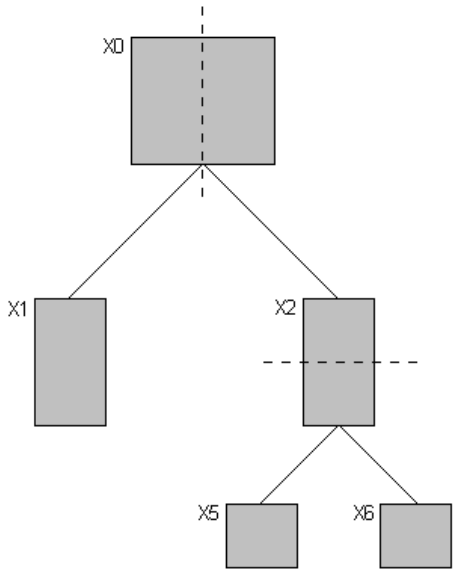


Figura 5.1: Divisão de blocos no MMP.

O critério empregado para a avaliação da distorção é dado por:

$$X_j^{m,n} = \begin{bmatrix} x_{1,1} & \cdots & x_{1,n} \\ \vdots & \ddots & \vdots \\ x_{m,1} & \cdots & x_{m,n} \end{bmatrix},\ S_i^{m,n} = \begin{bmatrix} s_{1,1} & \cdots & s_{1,n} \\ \vdots & \ddots & \vdots \\ s_{m,1} & \cdots & s_{m,n} \end{bmatrix},$$

$$\begin{aligned} \left\| X_j^{m,n} - S_i^{m,n} \right\|^2 &= (x_{1,1} - s_{1,1})^2 + \ldots + (x_{1,n} - s_{1,n})^2 + \ldots \\ &\quad + (x_{m,1} - s_{m,1})^2 + \ldots + (x_{m,n} - s_{m,n})^2 \\ &= \sum_{k=1}^{m} \sum_{l=1}^{n} (x_{k,l} - s_{k,l})^2 , \end{aligned} \tag{5.2}$$

que consiste simplesmente no erro quadrático do bloco.

O casamento multiescalas pode ser realizado basicamente através de duas estratégias: transformar os elementos $S_i^{k,l}$ do dicionário $D^{k,l}$ para que tenham as mesmas dimensões do bloco $X_j^{m,n}$ em codificação ou criar múltiplos dicionários ($D = \{D^{1,1}, D^{2,1}, D^{2,2}, \ldots, D^{m,n}\}$), com todas as escalas possíveis resultantes das divisões do bloco $X_0^{m,n}$. Assim, na primeira estratégia, procura-se o elemento $S_i^{m,n} = T_{k,l}^{m,n}\left[S_i^{k,l}\right]$ do único dicionário $D^{k,l}$ existente que melhor aproxima $X_j^{m,n}$ e, na segunda, procura-se o elemento $S_i^{m,n}$ do dicionário $D^{m,n}$, escolhido de D por ter as mesmas dimensões do bloco em codificação, que melhor aproxima $X_j^{m,n}$. A estratégia adotada neste trabalho foi a segunda, que é a mais rápida e possibilita uma atualização independente de todos os dicionários existentes.

Como existem N_D dicionários onde um dado bloco $X_j^{m,n}$ pode ser procurado, devido à implementação com múltiplos dicionários, o escolhido é aquele que possuir elementos de mesmas dimensões que $X_j^{m,n}$. O número N_D de dicionários existentes não é aleatório e depende das dimensões do maior bloco de entrada (divisão inicial da imagem, ou seja, $X_0^{m,n}$). Toda vez que uma divisão é necessária, verificam-se as dimensões do bloco e, se o número de linhas for maior que o de colunas, o mesmo é dividido em

$$X_j^{m,n} = \begin{pmatrix} X_{2j+1}^{\frac{m}{2},n} \\ X_{2j+2}^{\frac{m}{2},n} \end{pmatrix}; \tag{5.3}$$

caso contrário,

$$X_j^{m,n} = \left(X_{2j+1}^{m,\frac{n}{2}} \; X_{2j+2}^{m,\frac{n}{2}} \right). \tag{5.4}$$

Seguindo este algoritmo, as possíveis dimensões dos blocos, supondo-se a divisão inicial da imagem em blocos de 8×8, são 8×8, 8×4, 4×4, ... e 1x1. Se o maior bloco de entrada possui dimensões $m \times n$, onde $m = 2^k$ e $n = 2^l$, o número de dicionários e suas dimensões podem ser calculados por:

$$\mathsf{N}_D = k + l + 1 \tag{5.5}$$

e

$$\mathsf{DIM}[D] = \left\{ \left\{ 2^{\lfloor \frac{n+1}{2} \rfloor} \times 2^{\lfloor \frac{n}{2} \rfloor} \right\}, \; tal \; que \; n = 0, 1, \ldots \mathsf{N}_D - 1, \right\}, \tag{5.6}$$

onde N_D é o número de dicionários, DIM as dimensões possíveis e $\lfloor X \rfloor$ significa o maior inteiro menor ou igual a X.

Para indicar ao decodificador se um bloco foi dividido ou não, o algoritmo utiliza apenas um bit de *flag*. Se o valor deste for 1, o bloco é codificado como o índice i da sua aproximação $S_i^{m,n}$; caso contrário (o valor é 0), é dividido e a busca por um elemento aproximado reinicia para os novos blocos. Fica claro, então, que os *flags* permitem ao decodificador identificar a árvore de segmentação do bloco. Na Figura 5.1, a seqüência de *flags* para se representar a árvore de segmentação mostrada é 01011.

A construção dos dicionários $D^{k,l}$ é adaptativa e realizada com blocos $\hat{X}^{m,n}$ previamente codificados, atribuindo-se novos elementos a todas as escalas. A atualização em escalas diferentes é possível através de uma transformação de escala do tipo $\hat{X}^{k,l} = T_{m,n}^{k,l} \left[\hat{X}^{m,n} \right]$, a qual, por exemplo, pode converter um bloco de 8×8 num de 4×2. A transformação propriamente dita é decomposta em duas operações unidimensionais (transformação separável), transformando-se primeiro todas as linhas e depois todas as colunas, segundo as fórmulas (5.7) e (5.8). Estas fórmulas são aplicáveis a um vetor de N_0 posições sendo transformado para um de N posições, ou seja, uma linha do bloco por vez.

Para se aumentarem as dimensões de um bloco, o procedimento adotado é dado por:

$$
\begin{aligned}
&N_0 \to N,\ N > N_0 \\
&m_n^0 = \left\lfloor \frac{n.\,(N_0-1)}{N} \right\rfloor, \\
&m_n^1 = \begin{cases} m_n^0 + 1, & m_n^0 < N_0 - 1, \\ m_n^0, & m_n^0 = N_0 - 1, \end{cases} \\
&\alpha_n = n.\,(N_0-1) - N.m_n^0, \\
&S_n^s = \left\lfloor \frac{\alpha_n.\left(S_{m_n^1} - S_{m_n^0}\right)}{N} \right\rfloor + S_{m_n^0}, \\
&n = 0, 1, \ldots, N-1,
\end{aligned} \tag{5.7}
$$

onde S_n é o vetor original e S_n^s o escalonado.

Para se diminuírem as dimensões do bloco:

$$
\begin{aligned}
&N_0 \to N,\ N < N_0 \\
&m_{n,k}^0 = \left\lfloor \frac{n.\,(N_0-1)+k}{N} \right\rfloor, \\
&m_{n,k}^0 = \begin{cases} m_{n,k}^0, & m_{n,k}^0 < N_0, \\ N_0 - 1, & m_{n,k}^0 = N_0, \end{cases} \\
&m_{n,k}^1 = \begin{cases} m_{n,k}^0 + 1, & m_{n,k}^0 < N_0 - 1, \\ m_{n,k}^0, & m_{n,k}^0 = N_0 - 1, \end{cases} \\
&\alpha_{n,k} = n.\,(N_0-1) + k - N.m_{n,k}^0, \\
&S_n^s = \frac{1}{N_0+1} \sum_{k=0}^{N_0} \left(\left\lfloor \frac{\alpha_{n,k}.\left(S_{m_{n,k}^1} - S_{m_{n,k}^0}\right)}{N} \right\rfloor + S_{m_{n,k}^0} \right), \\
&n = 0, 1, \ldots, N-1.
\end{aligned} \tag{5.8}
$$

A transformação bidimensional empregada neste trabalho pode ser obtida, para um bloco $X^{m,n}$ sendo transformado em um bloco $X^{k,l}$, através do procedimento:

$$
\begin{aligned}
Y_i^{m,l} &= T_n^l \left[X_i^{m,n} \right], \ i = 0, 1, \ldots, m-1, \\
Z_j^{l,k} &= T_m^k \left[\left(\left(Y^{m,l} \right)^T \right)_j \right], \ j = 0, 1, \ldots, l-1, \\
X_h^{k,l} &= \left(\left(Z^{l,k} \right)^T \right)_h, \ h = 0, 1, \ldots, k-1,
\end{aligned} \tag{5.9}
$$

onde $X_i^{m,n}$ representa a linha i do bloco X com dimensões $m \times n$.

A transformação de escala é uma das características mais importantes do MMP e talvez a que apresenta o maior número de possibilidades. No presente caso, a transformação foi implementada utilizando-se operações de mudança de taxa de amostragem, como pode ser percebido nas equações (5.7) e (5.8). Em essência, muda-se o tamanho do vetor original de N_0 para N, utilizando-se interpolação linear. Então, aplica-se um filtro de média e amostra-se o resultado à taxa N_0, o que gera o vetor transformado com dimensão N.

A grande vantagem da operação descrita reside no fato de ser simples, eficaz e calculada rapidamente. Entretanto, outras transformações poderiam ter sido utilizadas, como a DCT. Nesse caso, o bloco de *pixels* seria convertido em um bloco de coeficientes de transformada, reduzido/ampliado (através do descarte de elementos/*padding*) e convertido novamente para um bloco de *pixels*, com dimensões diferentes das originais.

Outra característica importante do algoritmo é o dicionário inicial D_o, que deve existir para todas as escalas. O dicionário inicial não pode ser muito pequeno, nem muito grande, pois isso acarreta grandes erros de reconstrução. Por exemplo, imagens mais complexas, que para serem comprimidas a uma dada taxa (e.g. 0,5bpp) exigem valores maiores de distorção alvo δ, necessitam de dicionários iniciais menores. Logo, o tamanho do dicionário inicial é dependente da distorção alvo δ (na verdade, inversamente proporcional) e tem L elementos igualmente espaçados de P. A regra de formação do mesmo é mostrada na equação (5.10).

Em tese, o dicionário não precisa ser limitado, porém, nos experimentos realizados, seu tamanho máximo foi fixado em 200000 elementos. O dicionário é determinado segundo está descrito a seguir:

$$\text{valor\_pixel\_mínimo} = 0,$$

$$\text{valor\_pixel\_máximo} = 255,$$

$$\mathsf{L} = \left\lfloor \frac{2.\left(\text{valor\_pixel\_máximo} - \text{valor\_pixel\_mínimo}\right)}{\delta} \right\rfloor,$$

$$\mathsf{P} = \frac{\left(\text{valor\_pixel\_máximo} - \text{valor\_pixel\_mínimo}\right)}{\mathsf{L} - 1},$$

Para $n = 0, 1, \ldots, \mathsf{N}_D - 1$,

$$p = 2^{\left\lfloor \frac{n+1}{2} \right\rfloor},\ q = 2^{\left\lfloor \frac{n}{2} \right\rfloor},$$

$$D_o^{p,q} = \left\{ T_{1,1}^{p,q}\left[0\right], T_{1,1}^{p,q}\left[\left\lfloor 1.\mathsf{P} \right\rfloor\right], \ldots, T_{1,1}^{p,q}\left[\left\lfloor \left(\mathsf{L}-2\right).\mathsf{P} \right\rfloor\right], T_{1,1}^{p,q}\left[255\right] \right\} \tag{5.10}$$

onde L é o número de elementos do dicionário inicial, P é o *offset* para a geração dos elementos do dicionário e $D_o^{p,q}$ é o dicionário inicial com blocos de dimensões $p \times q$ propriamente dito.

Ao contrário de [1], o dicionário inicial utilizado não tem apenas vetores entre o maior e o menor valor de *pixel* presentes na imagem. Na verdade, o dicionário inicial abrange toda a escala possível, ou seja, de 0 a 255 (todas as imagens testadas possuem 256 níveis de cinza). A restrição utilizada em [1] pode prejudicar a codificação dos primeiros blocos de imagem, como foi observado durante as simulações, levando a um desempenho abaixo do que seria possível.

O tamanho de cada dicionário $D^{m,n}$ foi limitado em 200000 elementos para atender a limitações tanto do compilador quanto do *hardware* utilizado. Por isso, criou-se uma estratégia para atualização caso todas as posições sejam preenchidas: procuram-se os elementos $S_i^{p,q}$ menos utilizados e, a partir deste novo conjunto, o mais antigo, substituindo-o pela atualização.

Por último, vale lembrar que os índices i e os *flags* '1' e '0' resultantes da codificação não são escritos diretamente no arquivo, mas sim codificados por um codificador aritmético (atuando como uma camada auxiliar de compressão), proporcionando uma performance ainda melhor. O algoritmo de compressão está descrito na próxima página.

Procedimento $\hat{X}_j^{m,n} = \text{codifica}\left(X_j^{m,n}, \eta_o, \delta\right)$

Passo 1: Procura, no dicionário $D^{m,n}$, onde $m = 2^{\left\lfloor \frac{\eta_o+1}{2} \right\rfloor}$ e $n = 2^{\left\lfloor \frac{\eta_o}{2} \right\rfloor}$, o elemento $S_i^{m,n}$ que representa $X_j^{m,n}$ com menor distorção $\left\|X_j^{m,n} - S_i^{m,n}\right\|^2$, armazenando-o em $\hat{X}_j^{m,n}$.

Passo 2: Se a escala atual for $\eta_o == 0$, ou seja, 1×1 ($m == 1$ e $n == 1$), codifica o índice i do elemento $S_i^{m,n}$ escolhido e retorna $\hat{X}_j^{m,n}$.

Senão, vai para o **Passo 3**.

Passo 3: Se a distorção $\left\|X_j^{m,n} - S_i^{m,n}\right\|^2$ causada pelo elemento $S_i^{m,n}$ do dicionário $D^{m,n}$ for menor ou igual à distorção alvo δ $\left(\left\|X_j^{m,n} - S_i^{m,n}\right\|^2 \leq \delta^2.m.n\right)$, codifica o *flag* '1', codifica o índice i do elemento $S_i^{m,n}$ escolhido e retorna $\hat{X}_j^{m,n}$.

Senão, vai para o **Passo 4**.

Passo 4: Codifica o *flag* '0'

Passo 5: Se $m > n$, divide $X_j^{m,n}$ em $\begin{pmatrix} X_{2j+1}^{k,l} \\ X_{2j+2}^{k,l} \end{pmatrix}$, onde $k = \frac{m}{2}$ e $l = n$.

Senão, divide $X_j^{m,n}$ em $\left(X_{2j+1}^{k,l}\ X_{2j+2}^{k,l}\right)$, onde $k = m$ e $l = \frac{n}{2}$.

Passo 6: Computa $\hat{X}_{2j+1}^{k,l} = \text{codifica}\left(X_{2j+1}^{k,l}, \eta_o - 1, \delta\right)$

Passo 7: Computa $\hat{X}_{2j+2}^{k,l} = \text{codifica}\left(X_{2j+2}^{k,l}, \eta_o - 1, \delta\right)$

Passo 8: Se $m > n$, faz $\hat{X}_j^{m,n} = \begin{pmatrix} \hat{X}_{2j+1}^{k,l} \\ \hat{X}_{2j+2}^{k,l} \end{pmatrix}$.

Senão, faz $\hat{X}_j^{m,n} = \left(\hat{X}_{2j+1}^{k,l}\ \hat{X}_{2j+2}^{k,l}\right)$.

Passo 9: Atualiza o dicionário D em todas as escalas com $\hat{X}_j^{m,n}$, ou seja:

Para $n = 0, 1, \ldots, \mathsf{N}_D - 1$,

$p = 2^{\left\lfloor \frac{n+1}{2} \right\rfloor}$, $q = 2^{\left\lfloor \frac{n}{2} \right\rfloor}$,

$D^{p,q} = D^{p,q} \cup T_{m,n}^{p,q}\left[\hat{X}_j^{m,n}\right]$.

Passo 10: Retorna $\hat{X}_j^{m,n}$.

5.2 Resultados de simulações

As simulações apresentadas nesta seção foram obtidas através de uma implementação do MMP em C, rodando em ambiente *Linux*.

As imagens comprimidas para teste foram *LENA*, *BABOON*, *F-16*, *BRIDGE*, *AERIAL*, *BARBARA*, *GOLD*, *PP1209* e *PP1205*, todas de 512×512 *pixels*. As sete primeiras imagens, mostradas de A.1 a A.7, foram obtidas no *site* http://sipi.usc.edu /services/database/Database.html, e as duas últimas, mostradas de A.8 a A.9, foram digitalizadas do IEEE *Transactions on Image Processing*, volume 9, número 7, de julho de 2000. As páginas escolhidas foram as de número 1209 e 1205, que dão nome às imagens.

Os gráficos apresentados nesta seção mostram os resultados para o MMP e também para o SPIHT [17] e o JPEG [20]. Os resultados para o JPEG2000 [18] não foram apresentados, devido ao fato destes serem equivalentes aos do SPIHT (na verdade, levemente menores). Deste modo, uma comparação instantânea pode ser feita com um codificador baseado em DCT e outro em *Wavelets*.

Todas as imagens foram inicialmente divididas em blocos de 8×8, sendo estes, então, processados em seqüência pelo algoritmo, no sentido de leitura, ou seja, da esquerda para a direita e de cima para baixo. Devido ao fato da divisão inicial da imagem ser quadrada (8×8), a primeira partição do bloco ocorre na vertical[1], conforme mostrado na Figura 5.1.

Nos testes realizados, conforme a divisão inicial da imagem $X_0^{m,n}$ utilizava blocos maiores (2×2, 4×4,...), a $PSNR$ resultante das imagens reconstruídas aumentava, com exceção de blocos maiores que 8×8. Isso ocorre devido ao fato dos casamentos ocorrerem em escalas menores, o que faz com que o maior custo para indicar a árvore de segmentação acabe reduzindo a eficiência do sistema.

[1] O tipo de partição utilizado pode ter razoável influência nos resultados apresentados pelo algoritmo. Em [1], a primeira partição realizada pelo MMP ocorria na horizontal, ao contrário da apresentada neste trabalho, que ocorre na vertical. Durante as simulações, percebeu-se que imagens como *LENA* e *BARBARA* são beneficiadas pela partição na vertical, ao passo que *BABOON* e *BRIDGE* pela partição na horizontal. Este é um dos motivos da implementação do MMP realizada neste trabalho apresentar vantagens de até 0,3 e 0,4 dB nas imagens *LENA* e *BARBARA*, respectivamente.

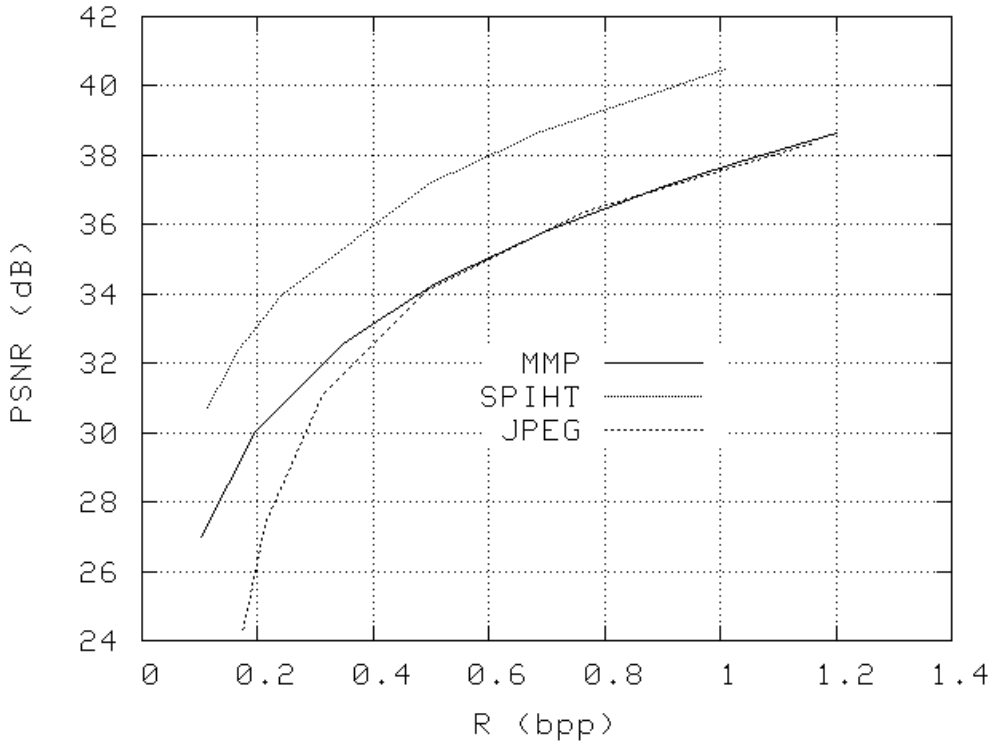


Figura 5.2: Taxa×distorção para *LENA* 512×512 comprimida com MMP.

O SPIHT apresentou resultados melhores em imagens mais suaves, perdendo apenas nas imagens que continham texto (*PP1209* e *PP1205*). Este comportamento deve-se ao fato desse codificador assumir que as imagens processadas têm uma alta concentração de energia em freqüências mais baixas, com uma queda aproximadamente exponencial em freqüências mais altas, o que não é verdade para imagens como a *PP1205*. O MMP, por sua vez, não faz qualquer asserção com respeito às características da imagem processada e adapta o seu dicionário ao tipo de dado sendo codificado. A imagem *PP1205*, comprimida a 0,5*bpp* pelo SPIHT e pelo MMP, é mostrada nas Figuras 5.11 e 5.12, respectivamente. A imagem *PP1209*, por sua vez, é mostrada nas Figuras 5.13 e 5.14.

A imagem *PP1209* é composta por duas regiões, sendo uma de texto e gráficos e outra de imagens em níveis de cinza (duas versões comprimidas da *LENA*). Como pode se perceber nos resultados, o ganho na região de texto compensa a perda na região das imagens *LENA*, o que acaba conferindo ao MMP um desempenho melhor.

Apesar dos bons resultados, o algoritmo MMP ainda apresenta um problema bastante inconveniente nas imagens processadas: o efeito de blocagem, exemplificado

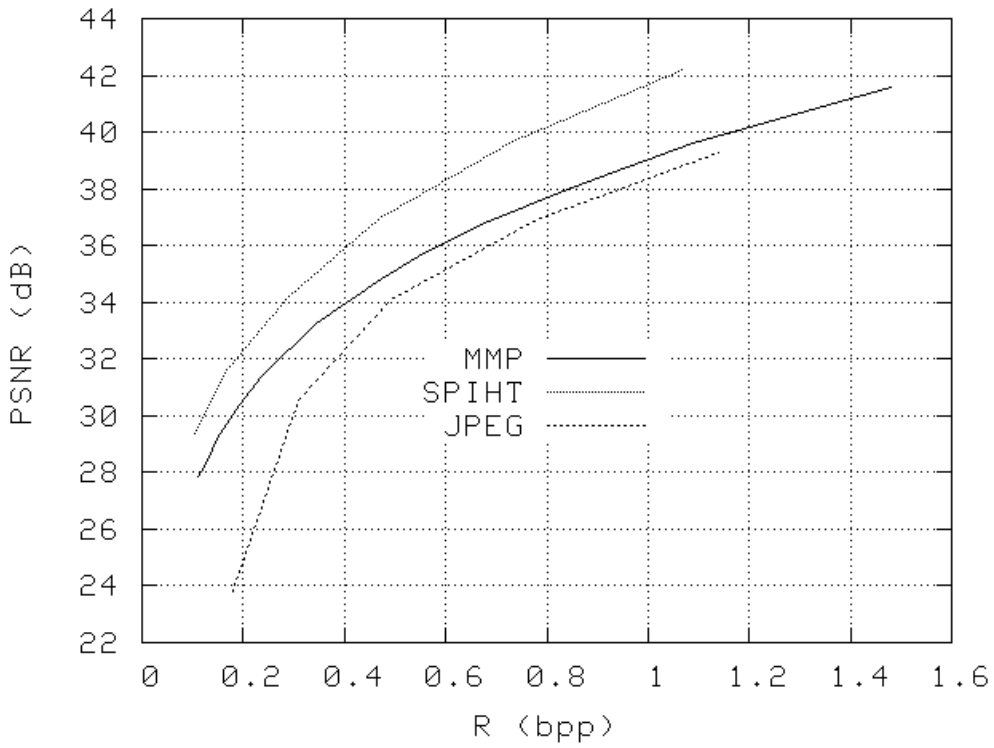


Figura 5.3: Taxa×distorção para *F-16* 512×512 comprimida com MMP.

na Figura 5.16 (em comparação com a Figura 5.15, que apresenta a imagem *LENA* comprimida com o SPIHT). Isto ocorre devido ao processamento independente sofrido por cada bloco de entrada codificado, não havendo qualquer mecanismo para checagem dos valores nas bordas dos blocos. Em [1], este problema foi resolvido com a utilização de filtragem na reconstrução da imagem, não sendo uma tarefa inerente ao algoritmo. No capítulo 7, será apresentado um algoritmo capaz de reduzir o efeito de blocagem e ainda proporcionar um aumento significativo na $PSNR$ de imagens suaves codificadas com o MMP.

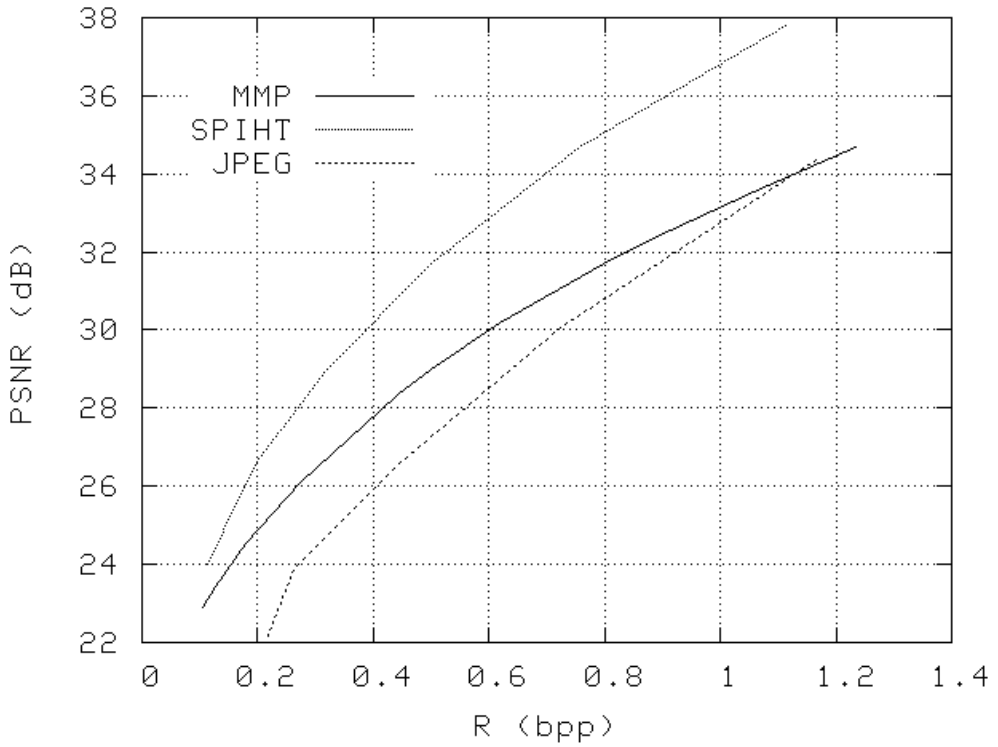


Figura 5.4: Taxa×distorção para *BARBARA* 512×512 comprimida com MMP.

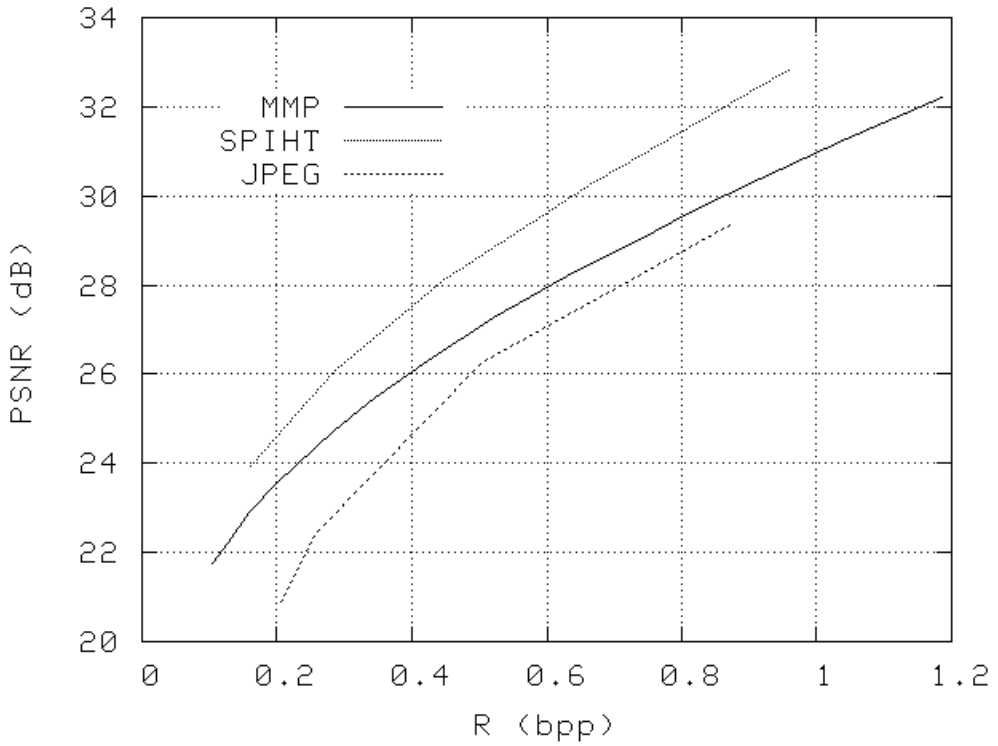


Figura 5.5: Taxa×distorção para *AERIAL* 512×512 comprimida com MMP.

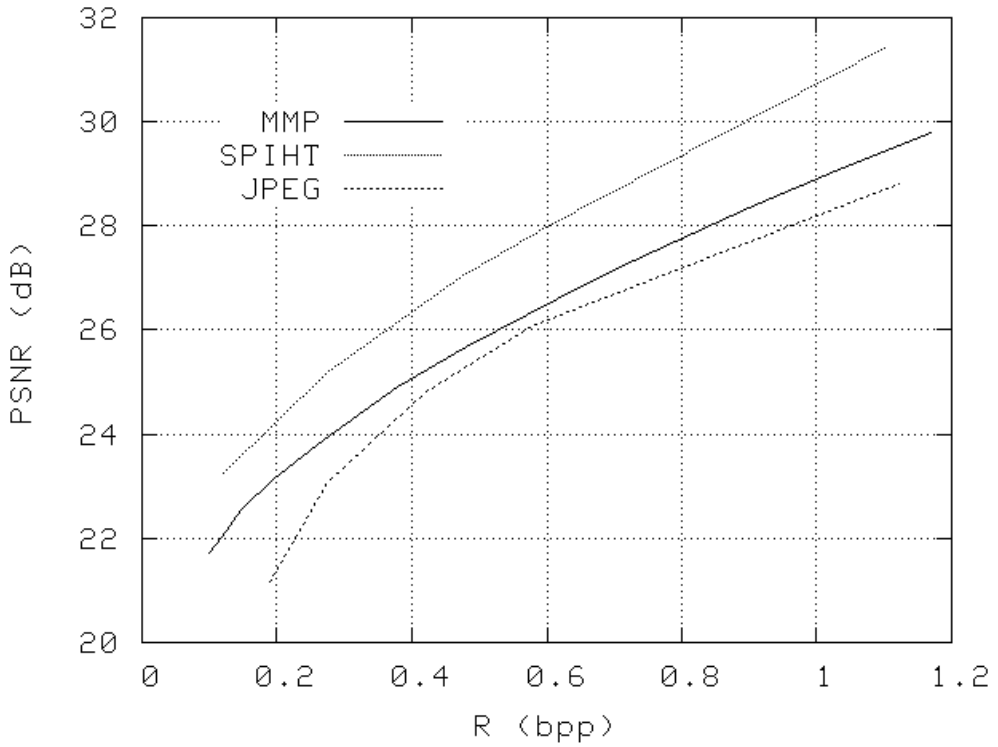


Figura 5.6: Taxa×distorção para *BRIDGE* 512×512 comprimida com MMP.

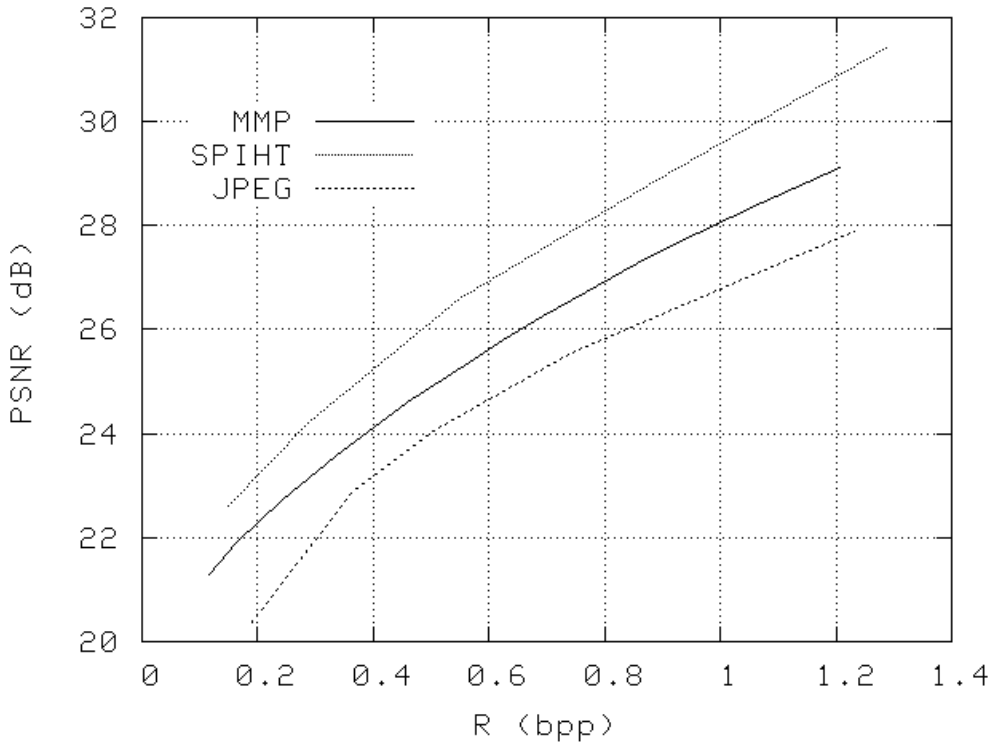


Figura 5.7: Taxa×distorção para *BABOON* 512×512 comprimida com MMP.

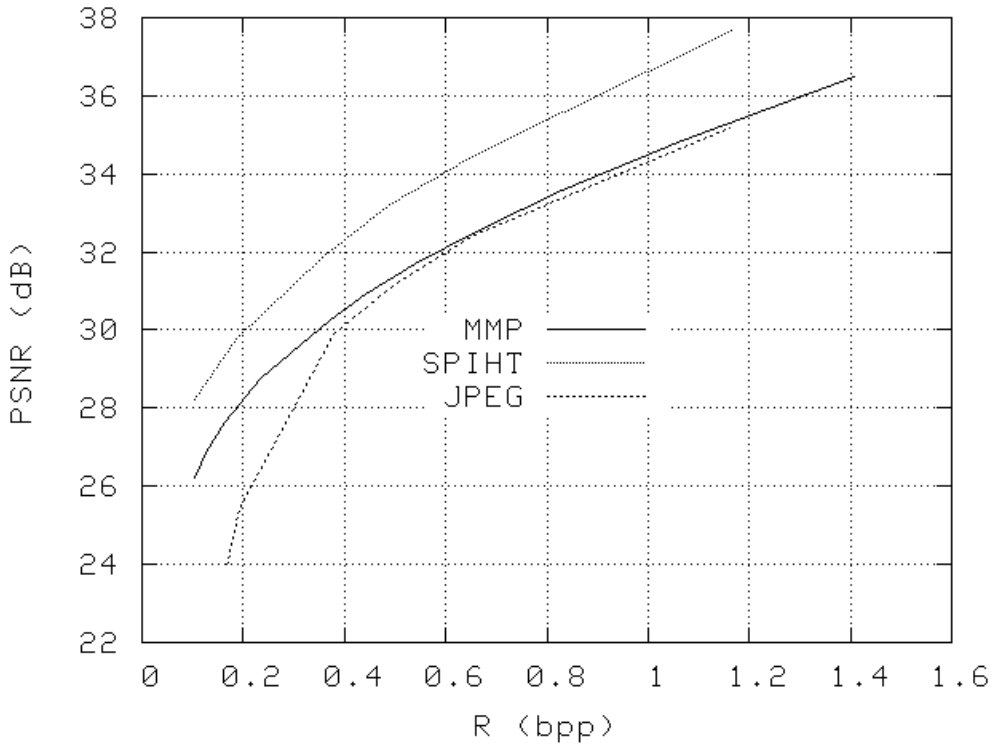


Figura 5.8: Taxa×distorção para *GOLD* 512×512 comprimida com MMP.

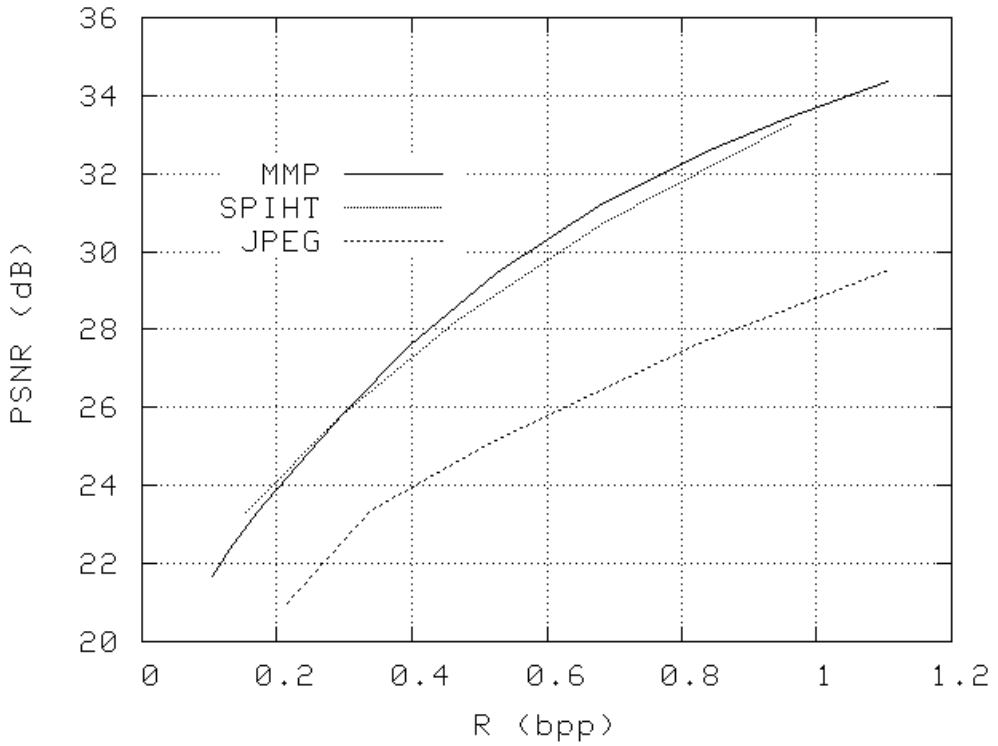


Figura 5.9: Taxa×distorção para *PP1209* 512×512 comprimida com MMP.

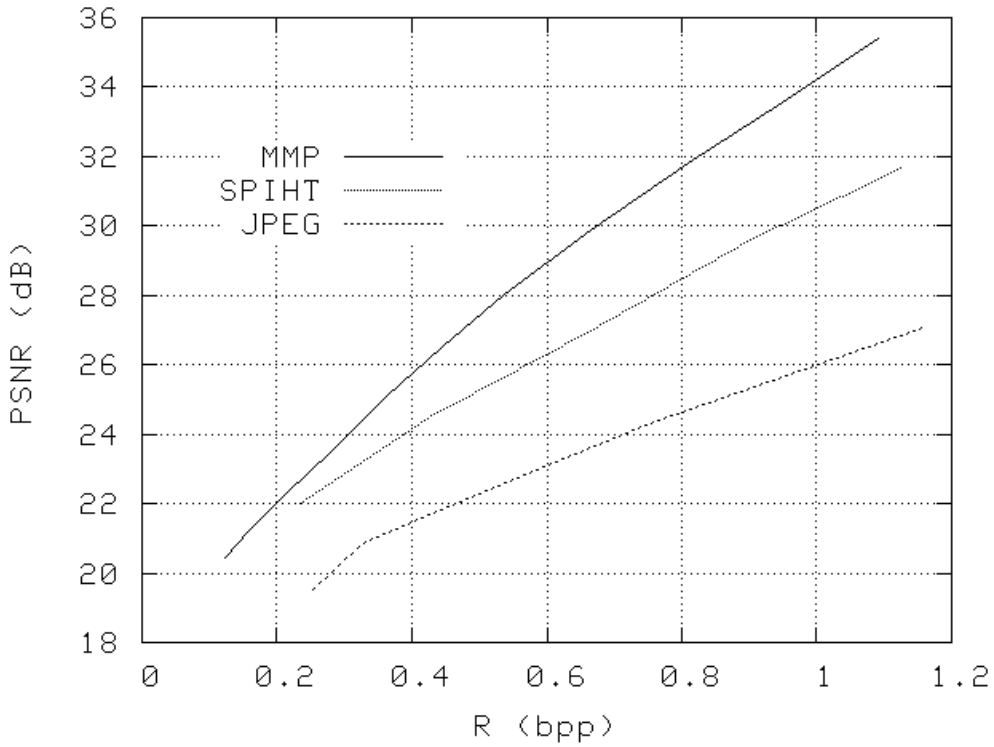
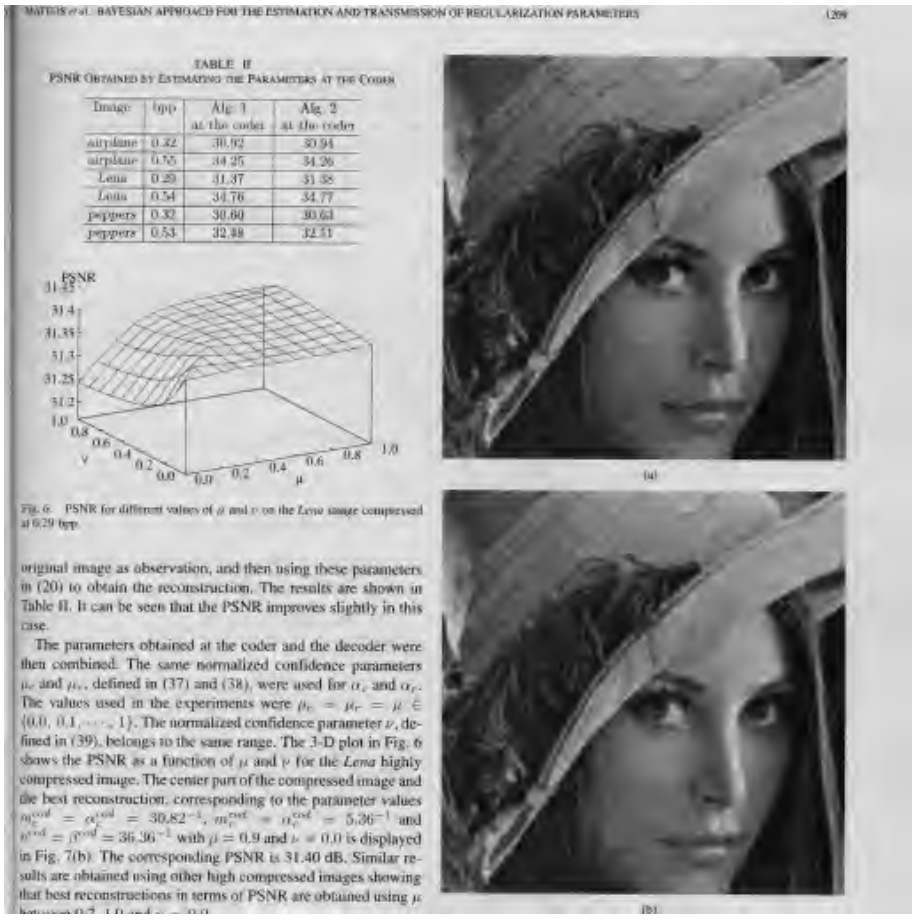


Figura 5.10: Taxa×distorção para *PP1205* 512×512 comprimida com MMP.

Figura 5.11: *PP1205* comprimida a 0,5*bpp* pelo SPIHT. PSNR=25,21dB.

Figura 5.12: *PP1205* comprimida a 0,5*bpp* pelo MMP. PSNR=27,45dB.



MATEOS *et al.*: BAYESIAN APPROACH FOR THE ESTIMATION AND TRANSMISSION OF REGULARIZATION PARAMETERS 1209

TABLE II
PSNR OBTAINED BY ESTIMATING THE PARAMETERS AT THE CODER

Image	bpp	Alg. 1 at the coder	Alg. 2 at the coder
airplane	0.32	30.92	30.94
airplane	0.76	34.25	34.26
Lena	0.29	31.37	31.38
Lena	0.54	34.76	34.77
peppers	0.32	30.60	30.61
peppers	0.53	32.49	32.51

Fig. 6. PSNR for different values of μ and ν on the *Lena* image compressed at 0.29 bpp.

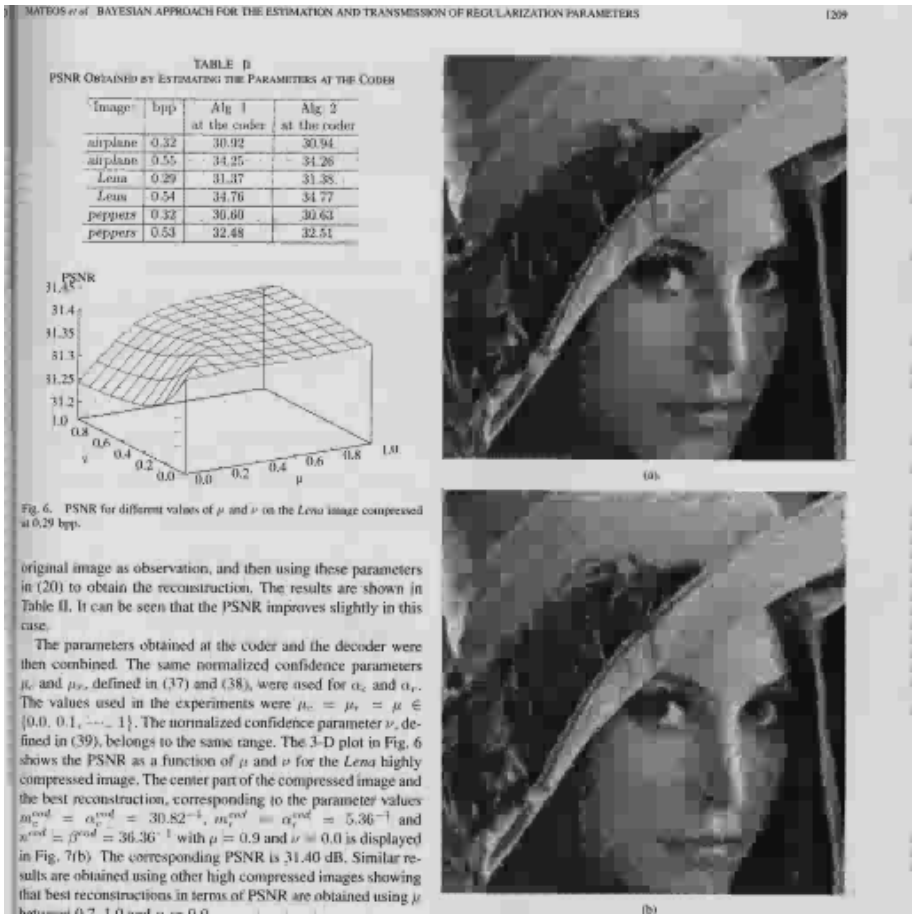
original image as observation, and then using these parameters in (20) to obtain the reconstruction. The results are shown in Table II. It can be seen that the PSNR improves slightly in this case.

The parameters obtained at the coder and the decoder were then combined. The same normalized confidence parameters μ_c and μ_r, defined in (37) and (38), were used for α_c and α_r. The values used in the experiments were $\mu_c = \mu_r = \mu \in \{0.0, 0.1, \cdots, 1\}$. The normalized confidence parameter ν, defined in (39), belongs to the same range. The 3-D plot in Fig. 6 shows the PSNR as a function of μ and ν for the *Lena* highly compressed image. The center part of the compressed image and the best reconstruction, corresponding to the parameter values $\alpha_c^{cod} = \alpha_c^{cod} = 30.82^{-1}$, $\alpha_r^{cod} = \alpha_c^{cod} = 5.36^{-1}$ and $\beta^{cod} = \beta^{cod} = 36.36^{-1}$ with $\mu = 0.9$ and $\nu = 0.0$ is displayed in Fig. 7(b). The corresponding PSNR is 31.40 dB. Similar results are obtained using other high compressed images showing that best reconstructions in terms of PSNR are obtained using μ

(a)

(b)

Figura 5.13: *PP1209* comprimida a 0,5*bpp* pelo SPIHT. PSNR=28,73dB.



MATEOS *et al.* BAYESIAN APPROACH FOR THE ESTIMATION AND TRANSMISSION OF REGULARIZATION PARAMETERS 1209

TABLE II
PSNR OBTAINED BY ESTIMATING THE PARAMETERS AT THE CODER

Image	bpp	Alg. 1 at the coder	Alg. 2 at the coder
airplane	0.32	30.92	30.94
airplane	0.55	34.25	34.26
Lena	0.29	31.37	31.38
Lena	0.54	34.76	34.77
peppers	0.32	30.60	30.63
peppers	0.53	32.48	32.51

Fig. 6. PSNR for different values of μ and ν on the *Lena* image compressed at 0.29 bpp.

original image as observation, and then using these parameters in (20) to obtain the reconstruction. The results are shown in Table II. It can be seen that the PSNR improves slightly in this case.

The parameters obtained at the coder and the decoder were then combined. The same normalized confidence parameters μ_c and μ_r, defined in (37) and (38), were used for α_c and α_r. The values used in the experiments were $\mu_c = \mu_r = \mu \in \{0.0, 0.1, \cdots, 1\}$. The normalized confidence parameter ν, defined in (39), belongs to the same range. The 3-D plot in Fig. 6 shows the PSNR as a function of μ and ν for the *Lena* highly compressed image. The center part of the compressed image and the best reconstruction, corresponding to the parameter values $m_c^{cod} = \alpha_c^{cod} = 30.82^{-1}$, $m_r^{cod} = \alpha_r^{cod} = 5.36^{-1}$ and $n^{cod} = \beta^{cod} = 36.36^{-1}$ with $\mu = 0.9$ and $\nu = 0.0$ is displayed in Fig. 7(b). The corresponding PSNR is 31.40 dB. Similar results are obtained using other high compressed images showing that best reconstructions in terms of PSNR are obtained using μ

(a)

(b)

Figura 5.14: *PP1209* comprimida a 0,5*bpp* pelo MMP. PSNR=29,08dB.

Figura 5.15: *LENA* comprimida a 0,5*bpp* pelo SPIHT. PSNR=37,22dB.

Figura 5.16: *LENA* comprimida a 0,5*bpp* pelo MMP. PSNR=34,25dB.

Capítulo 6

O algoritmo RDI-MMP

A decisão de partir ou não um bloco de entrada $X_l^{m,n}$, tomada pelo algoritmo de compressão MMP a cada iteração, é baseada em um critério local (distorção alvo δ estipulada para os blocos), levando a uma árvore de segmentação resultante de decisões tomadas individualmente para cada bloco. Se a distorção causada pelo bloco aproximado $S_i^{m,n}$ for maior que a distorção alvo δ, o bloco é dividido em dois outros iguais; caso contrário, não. Um exemplo de árvore de segmentação e blocos resultantes é mostrado na Figura 6.1, onde os nós-folhas, ou seja, as codificações efetivas, em cinza, são apresentados juntamente com seus respectivos blocos dentro da divisão inicial da imagem (bloco de maior hierarquia, ou seja, $X_0^{m,n}$).

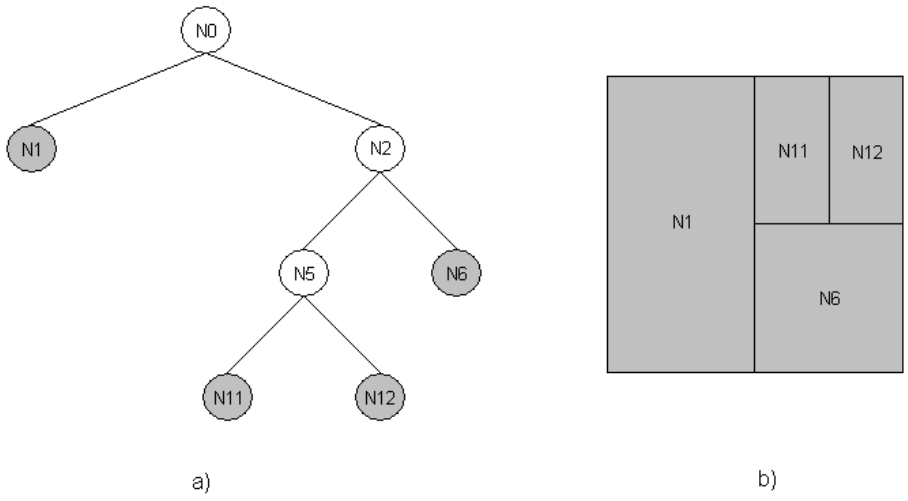


Figura 6.1: Exemplo: (a) Árvore de segmentação; (b) Blocos resultantes do MMP.

Deste modo, os bits disponíveis para a codificação são distribuídos sem se le-

var em consideração a necessidade global do bloco de entrada $X_0^{m,n}$, ou seja, não há um critério que permita disponibilizar um número maior de bits onde for necessário e um número menor onde não for. Por exemplo, na codificação de um dado bloco $X_l^{m,n}$, caso a distorção alvo seja α e a causada pela aproximação $S_i^{m,n}$ $\left(\|X_l^{m,n} - S_i^{m,n}\|^2\right)$ seja $\alpha + \epsilon$, com $\epsilon << \alpha$, o mesmo será dividido. Isto poderá levar a um consumo de bits muito maior que o do primeiro caso, pois agora é necessário codificar dois *flags* e dois índices, apesar da distorção ser apenas um pouco maior. Seria mais conveniente se a decisão de dividir o bloco de entrada fosse tomada apenas nos casos em que a distorção causada pelas novas aproximações compensasse o maior gasto de bits para a representação dos índices e dos *flags*, de acordo com a qualidade desejada para a imagem reconstruída.

Com certeza, resultados melhores podem ser obtidos através de um método que seja capaz de decidir, se um bloco deve ser dividido ou não, com base num critério que leve em consideração a relação entre distorção resultante e número de bits utilizado, encontrando a solução ótima (otimização) para as duas variáveis (distorção e taxa).

6.1 Descrição e implementação do algoritmo

Em [1], na parte que abrange a otimização da árvore de segmentação do MMP, dois algoritmos são apresentados: o aproximado e o modificado, também chamado de RD-MMP. No algoritmo aproximado, a otimização é realizada supondo-se que o dicionário D permanece estático durante a codificação e não há dependência entre os custos dos nós n_l da árvore de segmentação, o que não é exato. Em essência, os nós n_l da árvore de segmentação são podados ou não de acordo com um critério de menor custo, implementado com o uso de multiplicadores de *Lagrange* [19, 5]. Se o custo $\mathcal{J} = \mathcal{D}_{n_l} + \lambda.\mathcal{R}_{n_l}$ (onde $\mathcal{D}_{n_l}$ é a distorção, $\mathcal{R}_{n_l}$ o número de bits, ou taxa, para a representação do índice i da aproximação $S_i^{m,n}$ do bloco/nó n_l e λ o fator multiplicador) do nó-pai n_l (aproximação do bloco de entrada atual) for menor que a soma dos custos dos nós-filhos n_{2l+1} e n_{2l+2} (aproximações dos blocos resultantes da divisão), estes últimos serão podados. Isto significa que os mesmos serão retirados da árvore de segmentação e a representação do bloco de imagem será dada pelo

nó-pai. Caso a asserção anterior não seja verdade, os nós-filhos, ou as duas sub-árvores $S(n_{2l+1})$ e $S(n_{2l+2})$ a partir destes, serão mantidos e representarão o bloco de imagem. Por exemplo, na Figura 6.1, um nó-pai seria N_2 e seus nós-filhos seriam N_5, que também é o nó-raiz de uma sub-árvore, e N_6.

O algoritmo modificado, por sua vez, além de realizar todas as tarefas do aproximado, explora o fato de que a não inclusão de um dado elemento no dicionário pode afetar a codificação dos nós à direita do atual, levando a uma aproximação de maior custo para o bloco de entrada $X_0^{m,n}$ como um todo. Esta tarefa é chamada aqui de análise de dependência. Assim, um dado elemento que não foi incluído no dicionário, devido a uma poda, poderia representar, com menor custo, um ou mais dos nós a sua direita, e a sua ausência acaba ocasionando um resultado de maior custo, reduzindo o desempenho do sistema.

A análise de dependência entre os nós da árvore de segmentação proporciona um bom aumento na $PSNR$ das imagens codificadas, porém, ocasiona também um alto custo computacional, haja vista que todos os nós à direita do atual devem ser analisados com e sem o elemento que seria incluído no dicionário. Por exemplo, se o nó N_1 da Figura 6.2 estivesse sendo analisado, o custo com e sem a inclusão, no dicionário D, do elemento resultante da concatenação dos seus nós-filhos deveria ser calculado para N_2, N_5, N_6, N_{11}, N_{12}, N_{13} e N_{14}. Se, por exemplo, o custo do nó-pai for menor que o dos nós-filhos e a não inclusão da concatenação destes ocasionar um custo que não compensa a sua poda, as sub-árvores $S(n_3)$ e $S(n_4)$ serão mantidas e o bloco resultante da sua concatenação será incluído no dicionário [1].

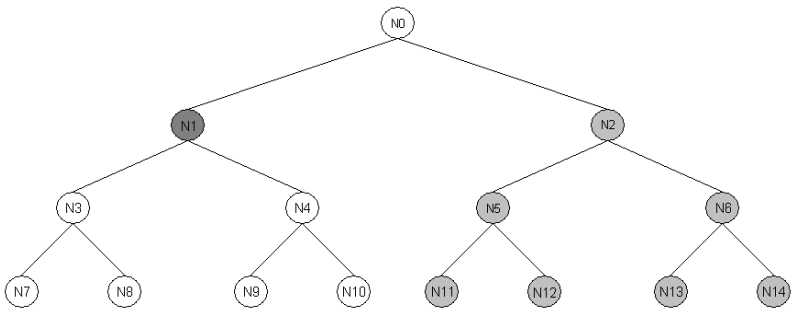


Figura 6.2: Análise de dependência realisada pelo RD-MMP desenvolvido em [1].

A fórmula do custo *Lagrangeano* pode ser visualizada como uma ponderação

entre a taxa $\mathcal{R}$ e a distorção $\mathcal{D}$. Quanto maior o valor do multiplicador λ, maior é a importância da taxa na decisão de poda, ou seja, preservam-se os elementos com menor taxa; quanto menor o valor do multiplicador, maior é a importância da distorção, ou seja, preservam-se os elementos com menor distorção. Logo, imagens comprimidas a baixas taxas necessitam de valores de λ altos e imagens comprimidas com pouca distorção (altas taxas) levam a valores de λ mais baixos.

É importante ressaltar que a distorção deve ser calculada exatamente como na fórmula (5.2), mostrada no capítulo anterior. Se for calculado, por exemplo, o erro médio quadrático, a avaliação para a poda perderá o sentido, pois quando os custos dos nós-filhos forem somados, não mais equivalerão ao do nó-pai, haja vista que a distorção do nó-pai seria uma média e as dos nós-filhos seriam duas médias. Assim, a soma das distorções dos nós-filhos deve equivaler quantitativamente à do nó-pai, pois são apenas divisões deste último. Em outras palavras, por exemplo, se a distorção do nó-pai é resultado da avaliação de 256 *pixels* (blocos de 16×16), a de cada nó-filho deve ser resultado da avaliação de 128 *pixels*, sem qualquer média, resultando a sua soma em uma avaliação de 256 *pixels*.

A simples implementação da análise de dependência, principal característica do RD-MMP [1], já provoca um aumento significativo, com relação ao algoritmo aproximado, na $PSNR$ das imagens codificadas. Porém, quando o dicionário atinge tamanhos razoáveis, a dependência entre os nós da árvore de segmentação perde importância e não mais justifica o alto custo computacional da sua análise. Isso ocorre devido ao fato das novas inclusões não serem mais tão diferentes dos inúmeros elementos que já existem em D.

Para se contornar esse problema, criou-se o algoritmo de otimização intermediário, chamado aqui de RDI-MMP, cuja principal característica consiste em tornar a otimização um espelho da codificação, realizando todos os passos executados por esta última. Isto significa que a otimização passa a simular todas as atualizações de dicionário, codificações de *flag* e codificações de índice executadas pelo algoritmo de codificação, realizando uma predição do seu comportamento. As tarefas adicionais executadas pelo RDI-MMP, em relação ao algoritmo aproximado, são as seguintes:

- Cria um dicionário rascunho $D_{\mathcal{R}}$ independente do dicionário oficial D e comple-

mentar a este, cujo objetivo é receber as atualizações que ocorreriam durante a codificação, aumentando o dicionário total;

- Cria contadores $C_{h_R}^{m,n}$ para totalizar as freqüências de utilização dos índices h_R de blocos $m \times n$ do dicionário rascunho, possibilitando a utilização desses valores para o cálculo dos custos $\mathcal{J}$, principalmente quando algum elemento do dicionário rascunho for escolhido, durante a otimização, para representar um dado bloco;

- Cria contadores complementares $\overline{C}_h^{m,n}$ e $\overline{C}_{h_F}^{m,n}$ para armazenar as freqüências adicionais de utilização dos índices h de blocos $m \times n$ do dicionário oficial e dos *flags* descritores da árvore de segmentação na escala $m \times n$, respectivamente, computando sua utilização durante todo o processo de otimização e adequando o modelo de freqüências ao formato atual da árvore de segmentação.

Cria-se, assim, uma função de otimização similar à de codificação e executada antes desta, cujo objetivo é montar a árvore de segmentação $\mathcal{T}$ completa e podar seus nós de acordo com o critério de mínimo custo *Lagrangeano*. A árvore $\mathcal{T}$ é montada e os custos calculados no mesmo sentido da codificação (nó 0→nó 1→nó 3→...), até chegar ao último nível (blocos de 1×1), onde se analisam os dois nós-filhos e retorna-se ao nó-pai. Nesse momento, avalia-se se o custo para a representação do nó-pai n_l é menor que o custo para a representação dos nós-filhos n_{2l+1} e n_{2l+2} e, caso isso seja verdade, os nós-filhos são retirados da árvore de segmentação e o nó-pai torna-se um nó-folha (bloco correspondente ao nó-pai será codificado através de um índice i ou i_R) temporário; caso contrário, os nós-filhos tornam-se os nós-folhas ou as sub-árvores $S(n_{2l+1})$ e $S(n_{2l+2})$ a partir destes são mantidas, atualizando-se o dicionário rascunho $D_{\mathcal{R}}$, com $\hat{X}_l^{m,n}$, e o custo do nó-pai.

Para se calcular o custo para a representação de um dado nó n_l (ou seja, o bloco $X_l^{m,n}$), a fórmula utilizada é:

$$\mathcal{J}(n_l) = \mathcal{D}_{n_l} + \lambda.\mathcal{R}_{n_l} \tag{6.1}$$

onde $\mathcal{J}(n_l)$ é o custo *Lagrangeano* do nó n_l, $\mathcal{D}_{n_l}$ é a distorção devida à aproximação de $X_l^{m,n}$ por $S_i^{m,n}$ ou $S_{i_R}^{m,n}$, dependendo do dicionário escolhido, e $\mathcal{R}_{n_l}$ é o número

de bits necessário para a representação do índice (i ou i_R) do elemento aproximado escolhido.

Para que o decodificador seja capaz de acompanhar a segmentação dos blocos, *flags* descritores da árvore de segmentação são codificados juntamente com os índices. Logo, a representação de um nó-folha deve levar em consideração a existência desses *flags*, sendo dada por:

$$\mathcal{J}_l = \mathcal{J}\left(n_l\right) + \lambda.\mathcal{R}_{1_l} \tag{6.2}$$

onde $\mathcal{J}_l$ é o custo *Lagrangeano* do nó-folha n_l e $\mathcal{R}_{1_l}$ é o número de bits necessário para a representação do *flag* '1', responsável por indicar que o nó n_l é um nó-folha. O custo do nó-pai é calculado com (6.2).

O custo para a representação dos nós-filhos (n_{2l+1} e n_{2l+2}), como nós-folhas, também deve levar em consideração os *flags* descritores da árvore de segmentação. O cálculo é dado por:

$$\mathcal{J}_{2l+1,2l+2} = \mathcal{J}_{2l+1} + \mathcal{J}_{2l+2} + \lambda.\mathcal{R}_{0_l} \tag{6.3}$$

onde $\mathcal{J}_{2l+1,2l+2}$ é o custo *Lagrangeano* para a representação dos nós-filhos n_{2l+1} e n_{2l+2} e $\mathcal{R}_{0_l}$ é o número de bits necessário para a representação do *flag* '0', responsável por indicar que houve uma divisão do nó n_l.

O número de contadores complementares $\overline{C}_h^{m,n}$, para cada escala, é igual ao tamanho máximo do dicionário oficial (fixado em 200000 elementos), e o tamanho do dicionário rascunho $D_{\mathcal{R}}$, também para cada escala, é igual ao número de linhas multiplicado pelo número de colunas do bloco de entrada de maior hierarquia (nesta implementação de 16×16), subtraindo-se 1 do resultado. Isto significa que todos os nós da árvore de segmentação são considerados, exceto os do último nível, pois estes não ocasionam inclusões no dicionário. O número de contadores do dicionário rascunho $C_{h_R}^{m,n}$ é igual ao tamanho do dicionário rascunho $D_{\mathcal{R}}$. É importante ressaltar que tanto o dicionário rascunho quanto os contadores devem existir para todas as escalas.

Apesar das vantagens, este novo modelo para a otimização da árvore de segmentação exige uma maior complexidade de programação. Em toda e qualquer poda

de nós-filhos (ou sub-árvores), é necessário destruir-se completamente a estrutura de análise criada por estes, ou seja, decrementar os contadores $\overline{C}_h^{m,n}$ e $C_{h_R}^{m,n}$, retirar os elementos inseridos no dicionário $D_{\mathcal{R}}$ e adequar os contadores $\overline{C}_{h_F}^{m,n}$ à árvore de segmentação resultante.

Ao se implementar estas modificações, a função de otimização torna-se um espelho da de codificação, acomodando toda a dinamicidade desta última e permitindo a obtenção de uma árvore de segmentação $\mathcal{T}$ mais adequada ao bloco de entrada $X_0^{m,n}$ e à evolução do dicionário D. Além disso, obtêm-se resultados similares aos do RD-MMP [1], com um custo computacional extremamente menor.

Alguns outros aspectos do algoritmo de otimização intermediário também devem ser observados, tais como:

- A árvore de segmentação $\mathcal{T}$ é iniciada com todos os seus nós n_l considerados como válidos (todas as posições possuem valor 1). Sua real condição será decidida somente após a avaliação dos custos;

- No momento do cálculo dos custos dos elementos $S_i^{m,n}$ e $S_{i_R}^{m,n}$, deve ser utilizada a somatória dos contadores $C_h^{m,n}$ (oficial) , $\overline{C}_h^{m,n}$ e $C_{h_R}^{m,n}$; no caso dos *flags*, deve ser utilizada a somatória dos contadores $C_{h_F}^{m,n}$ (oficial) e $\overline{C}_{h_F}^{m,n}$;

- Apesar da função retornar um bloco aproximado $X_l^{m,n}$, este não é utilizado para se codificar a imagem, mas sim apenas para possibilitar as concatenações e atualizações no dicionário $D_{\mathcal{R}}$;

- As únicas saídas desta função são o bloco aproximado $\hat{X}_l^{m,n}$ e a árvore de segmentação $\mathcal{T}$, que é utilizada pela função de codificação para decidir se um dado bloco $X_l^{m,n}$ deve ser codificado ou dividido, sendo esta tarefa, agora, dependente da relação entre taxa e distorção para a representação dos nós. Cada nó n_l é representado na posição l da árvore de segmentação $\mathcal{T}$ por 1 ou 0, sendo que o primeiro valor significa a existência do nó e o segundo a sua poda.

A transformação de escala utilizada no MMP com algoritmo de otimização intermediário, ou RDI-MMP, é a mesma do MMP padrão. Entretanto, devido à segmentação otimizada, o dicionário inicial perde grande parte da dependência com

a distorção alvo δ e passa a ser de 64 vetores igualmente espaçados entre 0 e 252 para todas as imagens. A equação de formação do dicionário inicial é dada por:

$$
\begin{aligned}
&\text{valor\_pixel\_mínimo} = 0,\\
&\text{valor\_pixel\_máximo} = 255,\\
&\mathsf{L} = 64\\
&\mathsf{P} = \left\lfloor \frac{(\text{valor\_pixel\_máximo} - \text{valor\_pixel\_mínimo})}{\mathsf{L}-1} \right\rfloor = 4,\\
&\text{valor\_pixel\_máximo} = \mathsf{P}.(\mathsf{L}-1) = 252,
\end{aligned}
$$

$$
\begin{aligned}
&\text{Para } n = 0, 1, \ldots, \mathsf{N}_D - 1,\\
&p = 2^{\left\lfloor \frac{n+1}{2} \right\rfloor},\ q = 2^{\left\lfloor \frac{n}{2} \right\rfloor},\\
&D_o^{p,q} = \left\{ T_{1,1}^{p,q}\left[0\right], T_{1,1}^{p,q}\left[4\right], \ldots, T_{1,1}^{p,q}\left[248\right], T_{1,1}^{p,q}\left[252\right] \right\}
\end{aligned}
\tag{6.4}
$$

O algoritmo de otimização intermediário está descrito na próxima página, com todos os detalhes aqui apresentados.

Procedimento $\left\{\hat{X}_l^{m,n}, \mathcal{T}\right\} = \text{otimiza}\left(X_l^{m,n}, \mathcal{T}_o, \eta_o\right)$

Passo 1: Faz $\mathcal{T} = \mathcal{T}_0$.

Passo 2: Procura no dicionário $D^{m,n}$, onde $m = 2^{\left\lfloor \frac{\eta_o+1}{2} \right\rfloor}$ e $n = 2^{\left\lfloor \frac{\eta_o}{2} \right\rfloor}$, o elemento $S_i^{m,n}$ que representa $X_l^{m,n}$ com menor custo $\mathcal{J}(n_l)$, armazenando i, $\mathcal{J}(n_l)$ e dicionário de origem. O elemento $S_i^{m,n}$ é armazenado em $\hat{X}_l^{m,n}$. O custo $\mathcal{J}(n_l)$ é calculado levando-se em consideração $\sum C_h^{m,n}$, $\sum \overline{C}_h^{m,n}$, $\sum C_{h_R}^{m,n}$, $C_i^{m,n}$ e $\overline{C}_i^{m,n}$.

Passo 3: Varre o dicionário $D_R^{m,n}$ e verifica se o elemento $S_{i_R}^{m,n}$ de menor custo representa $X_l^{m,n}$ com custo $\mathcal{J}(n_l)$ menor que o do escolhido no **Passo 2**. Se isso ocorrer, substitui $S_i^{m,n}$ por $S_{i_R}^{m,n}$, armazenando i_R, $\mathcal{J}(n_l)$ e dicionário de origem. O custo $\mathcal{J}(n_l)$ é calculado levando-se em consideração $\sum C_h^{m,n}$, $\sum \overline{C}_h^{m,n}$, $\sum C_{h_R}^{m,n}$ e $C_{i_R}^{m,n}$.

Passo 4: Se a escala atual for $\eta_o == 0$, ou seja, 1×1 ($m == 1$ e $n == 1$), incrementa $\overline{C}_i^{m,n}$ ou $C_{i_R}^{m,n}$, dependendo da origem do elemento escolhido, e retorna $\hat{X}_l^{m,n}$ e $\mathcal{T}$.

Senão, vai para o **Passo 5**.

Passo 5: Acrescenta, ao custo $\mathcal{J}(n_l)$ calculado, o valor $\lambda.\mathcal{R}_{1_l}$, para representar completamente o custo do nó-folha. A taxa do *flag* '1' deve ser calculada com $\sum C_{h_F}^{m,n}$, $\sum \overline{C}_{h_F}^{m,n}$, $C_{1_F}^{m,n}$ e $\overline{C}_{1_F}^{m,n}$.

Passo 6: Calcula e armazena, separadamente, o valor $\lambda.\mathcal{R}_{0_l}$, que posteriormente complementará o custo dos nós-filhos. A taxa do *flag* '0' deve ser calculada com $\sum C_{h_F}^{m,n}$, $\sum \overline{C}_{h_F}^{m,n}$, $C_{0_F}^{m,n}$ e $\overline{C}_{0_F}^{m,n}$.

Passo 7: Incrementa o contador $\overline{C}_{0_F}^{m,n}$.

Passo 8: Se $m > n$, divide $X_l^{m,n}$ em $\begin{pmatrix} X_{2l+1}^{k,j} \\ X_{2l+2}^{k,j} \end{pmatrix}$, onde $k = \frac{m}{2}$ e $j = n$.

Senão, divide $X_l^{m,n}$ em $\left(X_{2l+1}^{k,j}\ X_{2l+2}^{k,j}\right)$, onde $k = m$ e $j = \frac{n}{2}$.

Passo 9: Computa $\left\{\hat{X}_{2l+1}^{k,j}, \mathcal{T}_1\right\} = \text{otimiza}\left(X_{2l+1}^{k,j}, \mathcal{T}, \eta_o - 1\right)$

Passo 10: Computa $\left\{\hat{X}_{2l+2}^{k,j}, \mathcal{T}_2\right\} = \text{otimiza}\left(X_{2l+2}^{k,j}, \mathcal{T}, \eta_o - 1\right)$

Passo 11: Faz $\mathcal{T} = (\mathcal{T}_1)\,\text{AND}\,(\mathcal{T}_2)$.

Passo 12: Se o $\mathcal{J}_l \leq \mathcal{J}_{2l+1} + \mathcal{J}_{2l+2} + \lambda.\mathcal{R}_{0_l}$, vai para o **Passo 13**.

Senão, vai para o **Passo 19**.

Passo 13: Decrementa os contadores $\overline{C}_{0_F}^{w,y}$ em todas as escalas $w \times y$ relacionadas aos nós das sub-árvores $S(n_{2l+1})$ e $S(n_{2l+2})$ a serem podadas.

Passo 14: Decrementa os contadores $\overline{C}_{1_F}^{w,y}$ e $\overline{C}_{h}^{w,y}$ ou $\overline{C}_{h_R}^{w,y}$ nas escalas $w \times y$ referentes aos nós-folhas das sub-árvores $S(n_{2l+1})$ e $S(n_{2l+2})$ e nas posições dos seus índices (deve ser verificada a origem de cada elemento, ou seja, se é de D ou de D_R).

Passo 15: Decrementa o contador $\overline{C}_{0_F}^{m,n}$.

Passo 16: Incrementa o contador $\overline{C}_{1_F}^{m,n}$ e o contador $\overline{C}_{i}^{m,n}$ ou $C_{i_R}^{m,n}$, dependendo da origem do elemento $S^{m,n}$ utilizado como aproximação.

Passo 17: Elimina as atualizações do dicionário D_R ocasionadas pelas sub-árvores $S(n_{2l+1})$ e $S(n_{2l+2})$ a serem podadas.

Passo 18: Indica, em $\mathcal{T}$, que as sub-árvores $S(n_{2l+1})$ e $S(n_{2l+2})$ foram podadas e retorna $\hat{X}_l^{m,n}$ e $\mathcal{T}$.

Passo 19: Se $m > n$, faz $\hat{X}_l^{m,n} = \begin{pmatrix} \hat{X}_{2l+1}^{k,j} \\ \hat{X}_{2l+2}^{k,j} \end{pmatrix}$.

Senão, faz $\hat{X}_l^{m,n} = \begin{pmatrix} \hat{X}_{2l+1}^{k,j} & \hat{X}_{2l+2}^{k,j} \end{pmatrix}$.

Passo 20: Atualiza o dicionário D_R em todas as escalas com $\hat{X}_l^{m,n}$, ou seja:

Para $n = 0, 1, \ldots, \mathsf{N}_D - 1$,

$p = 2^{\left\lfloor \frac{n+1}{2} \right\rfloor}$, $q = 2^{\left\lfloor \frac{n}{2} \right\rfloor}$,

$D^{p,q} = D^{p,q} \cup T_{m,n}^{p,q} \left[\hat{X}_l^{m,n} \right]$.

Passo 21: Faz $\mathcal{J}_l = \mathcal{J}_{2l+1} + \mathcal{J}_{2l+2} + \lambda.\mathcal{R}_{0_l}$

Passo 22: Retorna $\hat{X}_l^{m,n}$ e $\mathcal{T}$.

Como agora a divisão de um bloco de entrada está condicionada à árvore de segmentação $\mathcal{T}$ construída pela função de otimização e não mais à distorção alvo δ, e o elemento procurado é o de menor custo e não mais o de menor distorção, o algoritmo de codificação precisa sofrer algumas modificações, mostradas na próxima página.

Procedimento $\hat{X}_l^{m,n} = \text{codifica}\,(X_l^{m,n}, \eta_o, \mathcal{T})$

Passo 1: Se $\mathcal{T}_{2l+1} == 0$ e $\mathcal{T}_{2l+2} == 0$, ou $\eta_o == 0$, vai para o **Passo 2**.

Senão, vai para o **Passo 5**.

Passo 2: Procura, no dicionário $D^{m,n}$, onde $m = 2^{\lfloor \frac{\eta_o+1}{2} \rfloor}$ e $n = 2^{\lfloor \frac{\eta_o}{2} \rfloor}$, o elemento $S_i^{m,n}$ que representa $X_l^{m,n}$ com menor custo $\mathcal{J}(n_l)$, armazenando-o em $\hat{X}_l^{m,n}$.

Passo 3: Se a escala atual for $\eta_o == 0$, ou seja, 1×1 ($m == 1$ e $n == 1$), codifica o índice i do elemento $S_i^{m,n}$ escolhido e retorna $\hat{X}_l^{m,n}$.

Senão, vai para o **Passo 4**.

Passo 4: Codifica o *flag* '1', codifica o índice i do elemento $S_i^{m,n}$ escolhido e retorna $\hat{X}_l^{m,n}$.

Passo 5: Codifica o *flag* '0'.

Passo 6: Se $m > n$, divide $X_l^{m,n}$ em $\begin{pmatrix} X_{2l+1}^{k,j} \\ X_{2l+2}^{k,j} \end{pmatrix}$, onde $k = \frac{m}{2}$ e $j = n$.

Senão, divide $X_l^{m,n}$ em $\begin{pmatrix} X_{2l+1}^{k,j} & X_{2l+2}^{k,j} \end{pmatrix}$, onde $k = m$ e $j = \frac{n}{2}$.

Passo 7: Computa $\hat{X}_{2l+1}^{k,j} = \text{codifica}\left(X_{2l+1}^{k,j}, \eta_o - 1, \mathcal{T}\right)$

Passo 8: Computa $\hat{X}_{2l+2}^{k,j} = \text{codifica}\left(X_{2l+2}^{k,j}, \eta_o - 1, \mathcal{T}\right)$

Passo 9: Se $m > n$, faz $\hat{X}_l^{m,n} = \begin{pmatrix} \hat{X}_{2l+1}^{k,j} \\ \hat{X}_{2l+2}^{k,j} \end{pmatrix}$.

Senão, faz $\hat{X}_l^{m,n} = \begin{pmatrix} \hat{X}_{2l+1}^{k,j} & \hat{X}_{2l+2}^{k,j} \end{pmatrix}$.

Passo 10: Atualiza o dicionário D em todas as escalas com $\hat{X}_l^{m,n}$, ou seja:

Para $n = 0, 1, \ldots, \mathsf{N}_D - 1$,

$p = 2^{\lfloor \frac{n+1}{2} \rfloor}$, $q = 2^{\lfloor \frac{n}{2} \rfloor}$,

$D^{p,q} = D^{p,q} \cup T_{m,n}^{p,q} \left[\hat{X}_l^{m,n}\right]$.

Passo 11: Retorna $\hat{X}_l^{m,n}$.

6.2 Resultados de simulações

As simulações apresentadas nesta seção foram obtidas através de uma implementação do RDI-MMP em C, rodando em ambiente *Linux*.

As imagens comprimidas para teste foram as mesmas utilizadas no capítulo anterior, ou seja, *LENA*, *BABOON*, *F-16*, *BRIDGE*, *AERIAL*, *BARBARA*, *GOLD*, *PP1209* e *PP1205*, todas de 512×512 *pixels*.

Os gráficos apresentados nesta seção mostram os resultados para o RDI-MMP e também para o SPIHT, o JPEG, o MMP e o RD-MMP [1]. Deste modo, uma comparação instantânea pode ser feita com um codificador baseado em DCT, outro em *Wavelets* e também com aqueles desenvolvidos anteriormente em [1] e no capítulo 5.

Todas as imagens foram inicialmente divididas em blocos de 16×16, sendo estes, então, processados em seqüência pelo algoritmo, no sentido de leitura, ou seja, da esquerda para a direita e de cima para baixo. Apesar do MMP apresentar uma redução de desempenho ao utilizar blocos de 16×16, a otimização da árvore de segmentação permitiu ao RDI-MMP processar blocos com estas dimensões, apresentando melhora de desempenho com relação ao processamento de blocos 8×8.

As taxas $\mathcal{R}_{n_l}$, $\mathcal{R}_{1_l}$ e $\mathcal{R}_{0_l}$ foram estimadas com os dados de freqüência relativa disponveis no codificador aritmético e nos contadores criados, utilizando-se a fórmula (2.1).

Apesar do SPIHT ainda ser o responsável pelos melhores resultados para imagens mais suaves, como *LENA* e *F-16*, o RDI-MMP apresentou melhoras significativas com relação ao MMP padrão, com resultados similares aos do RD-MMP desenvolvido em [1], que utiliza uma rotina de otimização bem mais complexa computacionalmente. As únicas imagens onde o RDI-MMP obteve resultados abaixo do RD-MMP foram *BABOON*, *BRIDGE* e *AERIAL*. Este fato é explicado pela orientação da segmentação utilizada, que no presente caso é na vertical (primeira partição na vertical). Utilizando-se a mesma segmentação, o algoritmo torna-se superior.

Além do aumento na $PSNR$ de todas as imagens, em relação ao MMP padrão, também houve uma melhora na qualidade subjetiva, podendo ser percebida analisando-se as Figuras 5.16 e 6.12, que mostram a imagem *LENA* comprimida

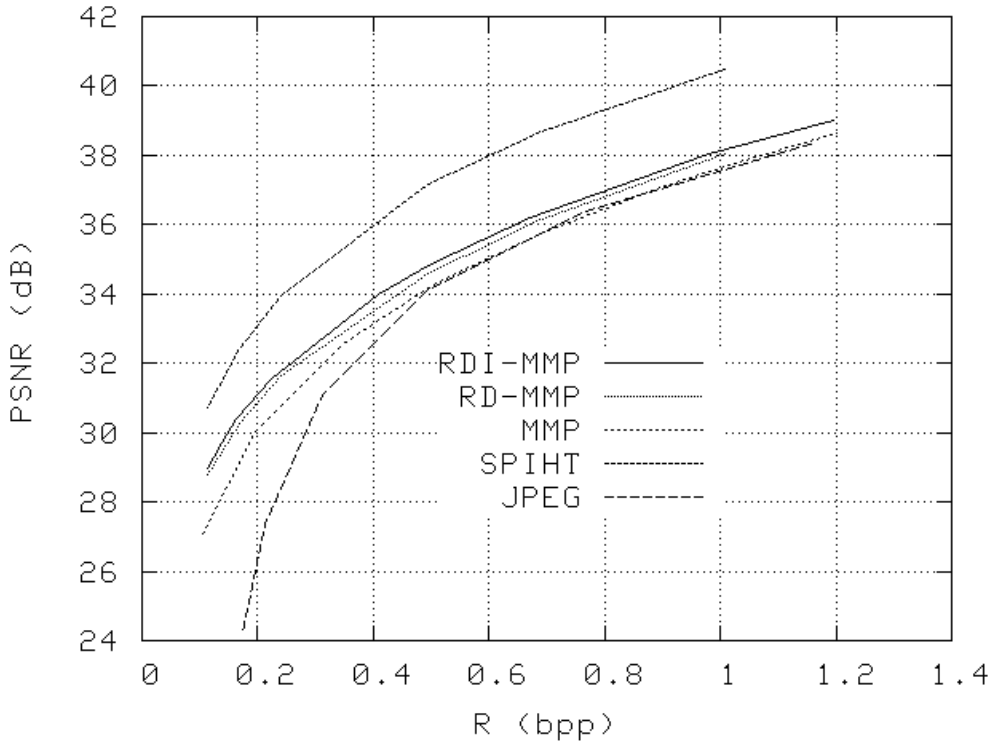


Figura 6.3: Taxa×distorção para *LENA* 512×512 comprimida com RDI-MMP.

com o MMP e o RDI-MMP, respectivamente. Parte desta melhora é devida à maior $PSNR$ obtida, porém, a segmentação otimizada, que permitiu a utilização de blocos maiores, também levou a uma grande redução no efeito de blocagem. As imagens *PP1205* e *PP1209*, comprimidas pelo RDI-MMP a 0,5*bpp*, estão disponíveis para comparação nas Figuras 6.13 e 6.14, respectivamente.

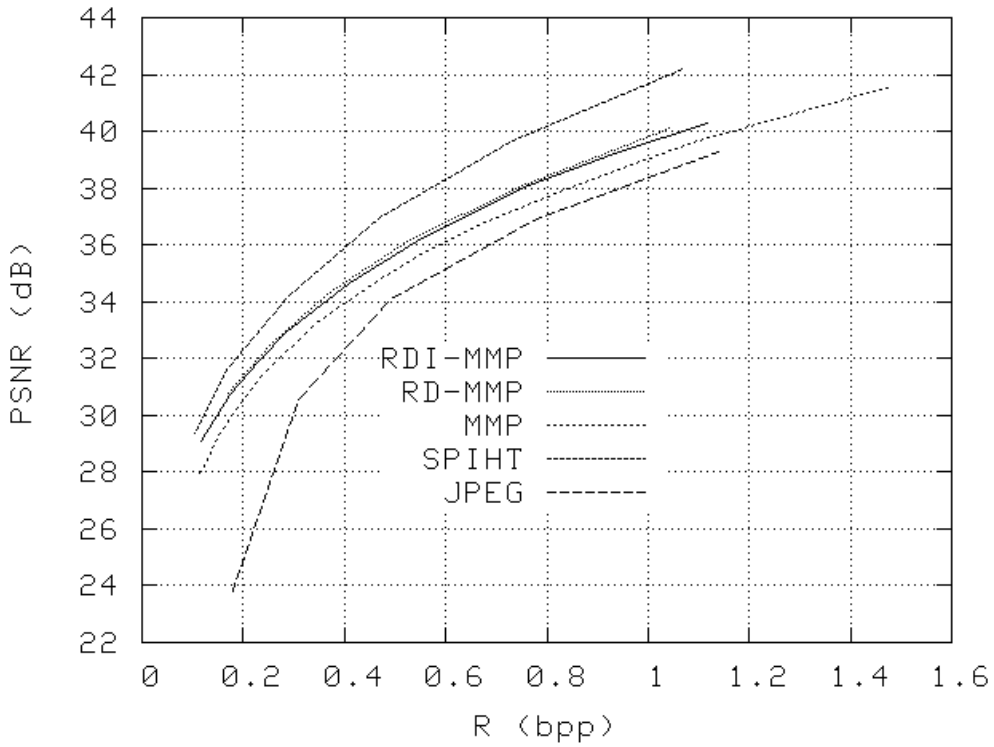


Figura 6.4: Taxa×distorção para *F-16* 512×512 comprimida com RDI-MMP.

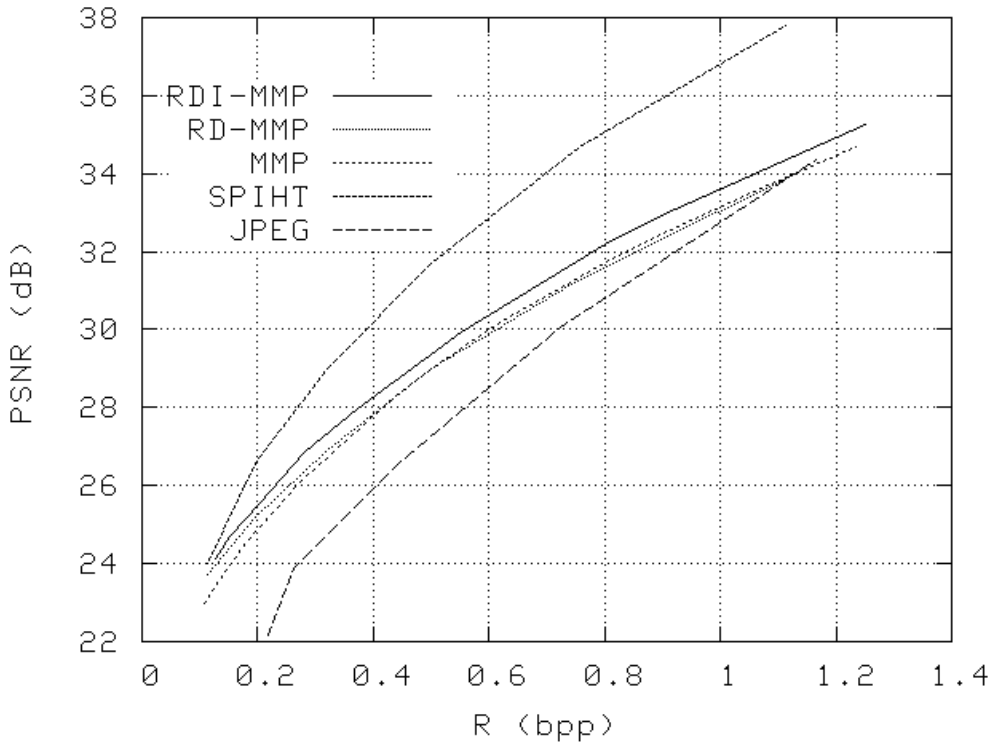


Figura 6.5: Taxa×distorção para *BARBARA* 512×512 comprimida com RDI-MMP.

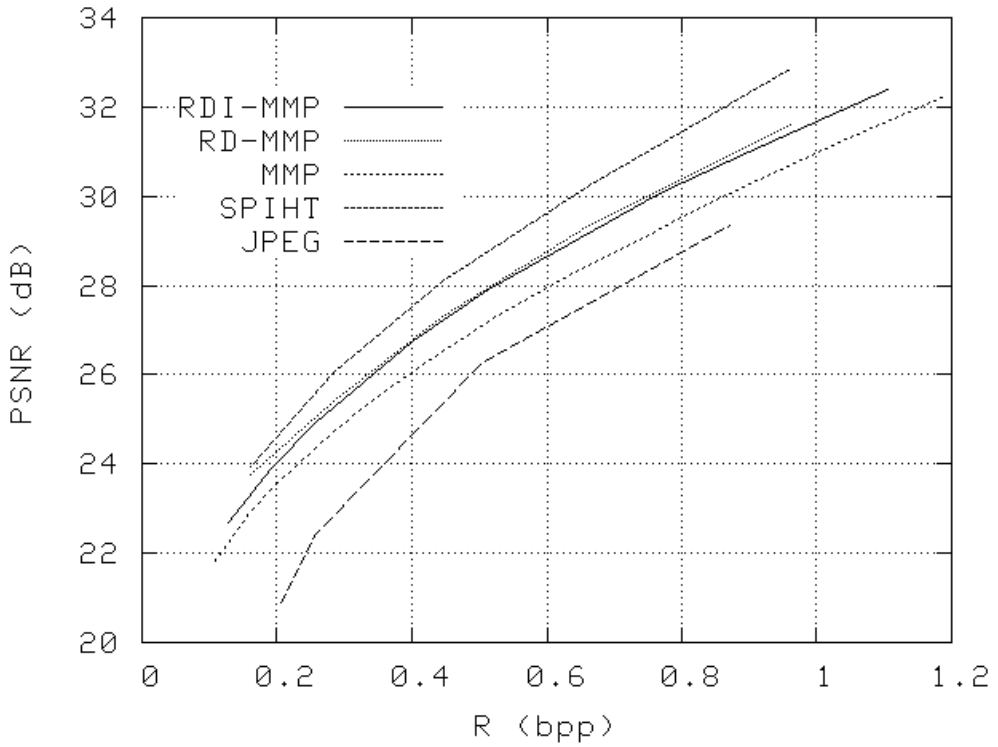


Figura 6.6: Taxa×distorção para *AERIAL* 512×512 comprimida com RDI-MMP.

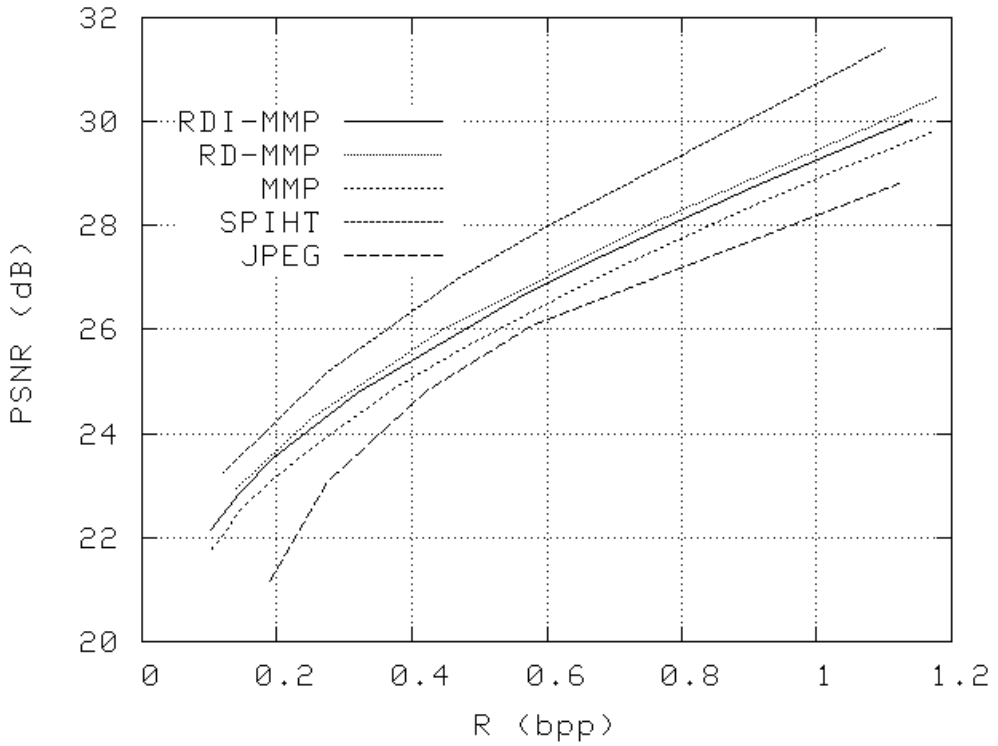


Figura 6.7: Taxa×distorção para *BRIDGE* 512×512 comprimida com RDI-MMP.

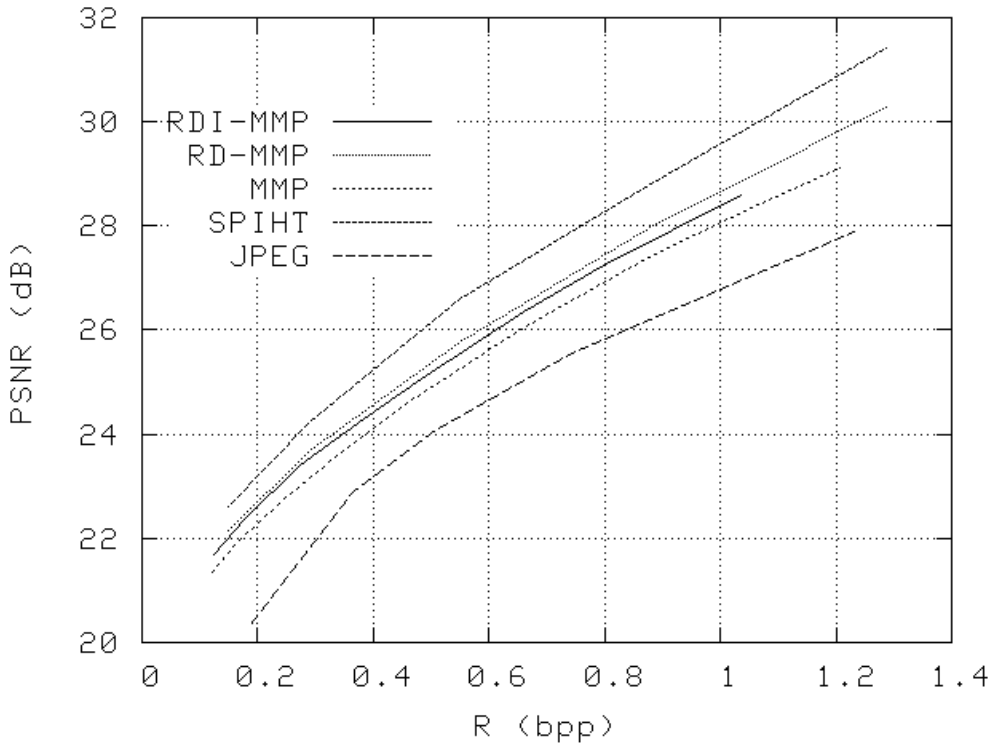


Figura 6.8: Taxa×distorção para *BABOON* 512×512 comprimida com RDI-MMP.

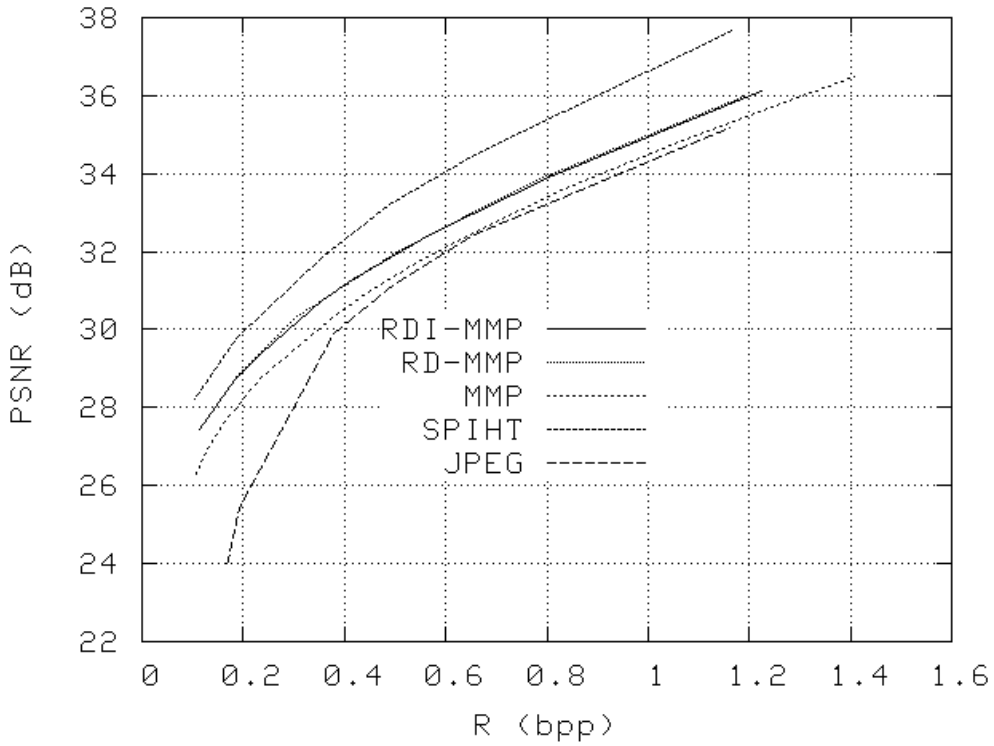


Figura 6.9: Taxa×distorção para *GOLD* 512×512 comprimida com RDI-MMP.

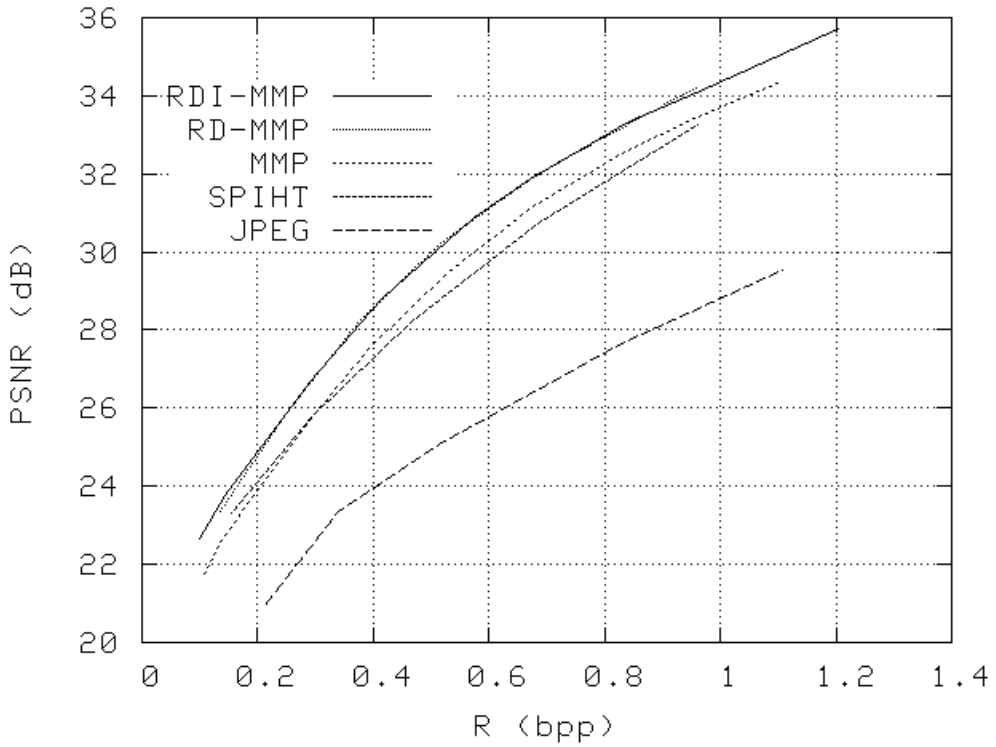


Figura 6.10: Taxa×distorção para *PP1209* 512×512 comprimida com RDI-MMP.

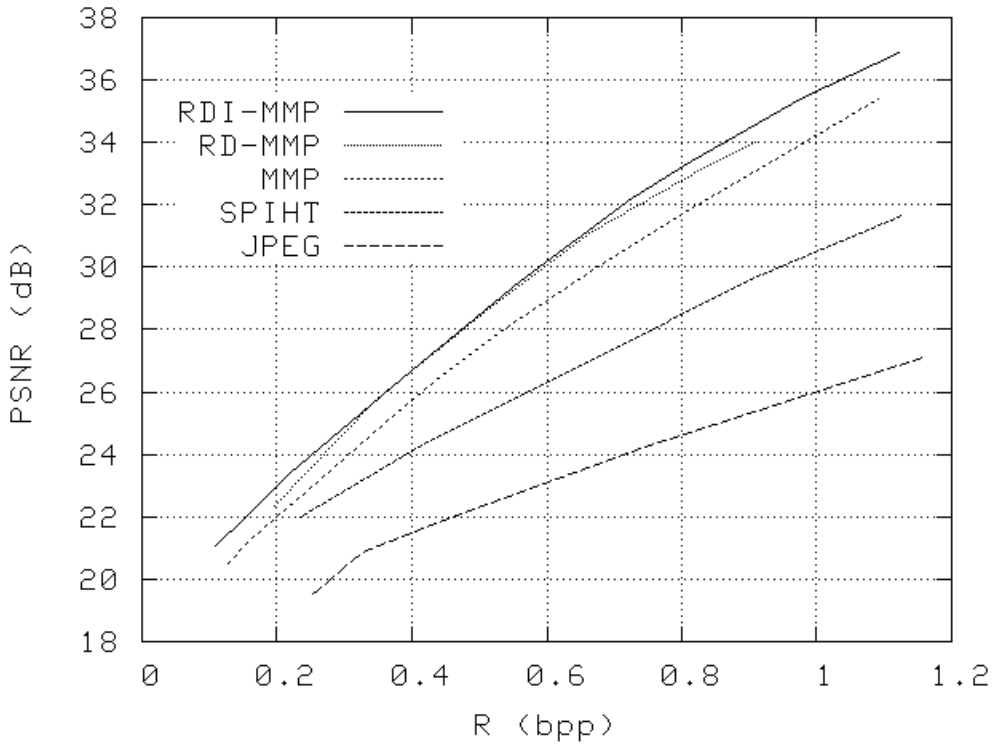
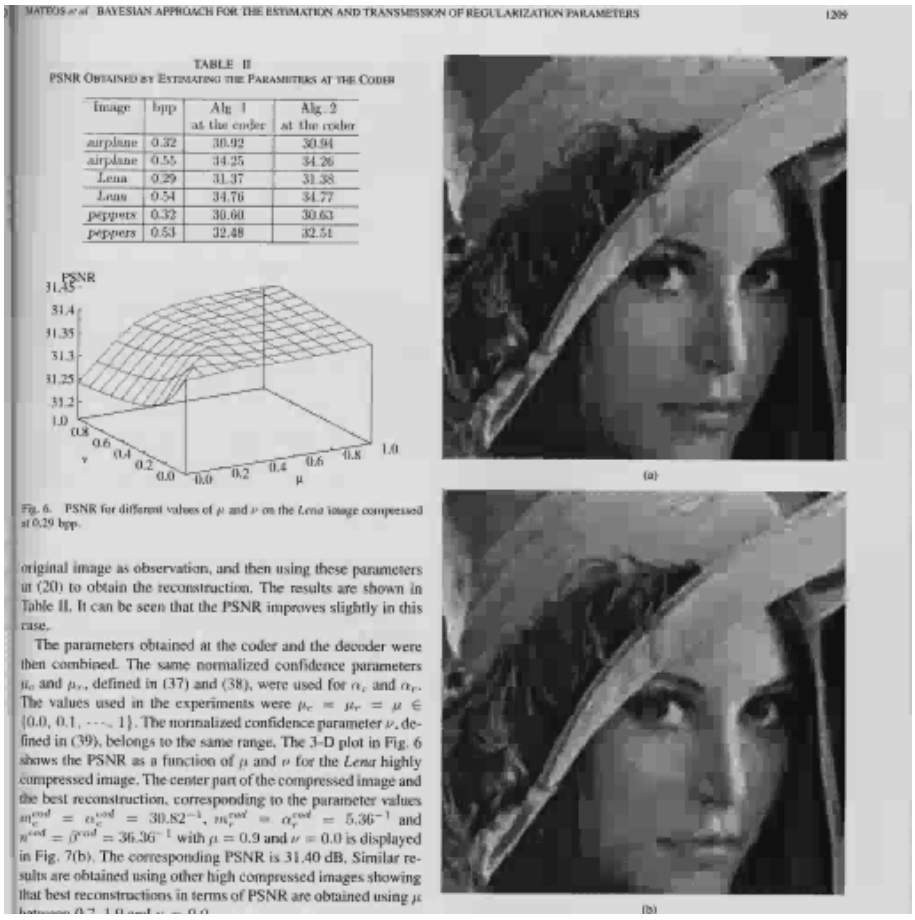


Figura 6.11: Taxa×distorção para *PP1205* 512×512 comprimida com RDI-MMP.

Figura 6.12: *LENA* comprimida a 0,5*bpp* pelo RDI-MMP. PSNR=34,88dB.

Figura 6.13: *PP1205* comprimida a 0,5*bpp* pelo RDI-MMP. PSNR=28,50dB.



MATEOS *et al.* BAYESIAN APPROACH FOR THE ESTIMATION AND TRANSMISSION OF REGULARIZATION PARAMETERS 1209

TABLE II
PSNR OBTAINED BY ESTIMATING THE PARAMETERS AT THE CODER

Image	bpp	Alg. 1 at the coder	Alg. 2 at the coder
airplane	0.32	30.92	30.94
airplane	0.55	34.25	34.26
Lena	0.29	31.37	31.38
Lena	0.54	34.76	34.77
peppers	0.32	30.60	30.63
peppers	0.53	32.48	32.51

Fig. 6. PSNR for different values of μ and ν on the *Lena* image compressed at 0.29 bpp.

original image as observation, and then using these parameters in (20) to obtain the reconstruction. The results are shown in Table II. It can be seen that the PSNR improves slightly in this case.

The parameters obtained at the coder and the decoder were then combined. The same normalized confidence parameters μ_c and μ_r, defined in (37) and (38), were used for α_c and α_r. The values used in the experiments were $\mu_c = \mu_r = \mu \in \{0.0, 0.1, \cdots, 1\}$. The normalized confidence parameter ν, defined in (39), belongs to the same range. The 3-D plot in Fig. 6 shows the PSNR as a function of μ and ν for the *Lena* highly compressed image. The center part of the compressed image and the best reconstruction, corresponding to the parameter values $m_c^{cod} = \alpha_c^{cod} = 30.82^{-1}$, $m_r^{cod} = \alpha_r^{cod} = 5.36^{-1}$ and $n^{cod} = \beta^{cod} = 36.36^{-1}$ with $\mu = 0.9$ and $\nu = 0.0$ is displayed in Fig. 7(b). The corresponding PSNR is 31.40 dB. Similar results are obtained using other high compressed images showing that best reconstructions in terms of PSNR are obtained using μ

(a)

(b)

Figura 6.14: *PP1209* comprimida a 0,5*bpp* pelo RDI-MMP. PSNR=30,01dB.

Capítulo 7

O algoritmo SM-MMP

Nos capítulos anteriores, foi demonstrado que o MMP e o RDI-MMP proporcionam bons resultados na compressão de imagens mistas ou de texto, o que é devido principalmente à inexistência de suposições quanto às características da imagem em processamento e à adapatabilidade apresentada pelo dicionário à mesma. Provou-se também que a otimização da árvore de segmentação melhora significativamente o rendimento do algoritmo, proporcionando a utilização de blocos maiores e reduzindo e efeito de blocagem, uma conseqüência do processamento independente aplicado aos blocos pelo algoritmo, que reduz bastante a qualidade subjetiva da imagem reconstruída.

Entretanto, dois problemas ainda perisistem: o baixo desempenho em imagens mais suaves e a presença, ainda perceptível, do efeito de blocagem. A solução do primeiro problema requer uma nova abordagem para a escolha dos elementos do dicionário, ao passo que o segundo pode ser bastante reduzido com a aplicação de filtragem no decodificador. Porém, esta técnica acaba reduzindo, em muitos casos, a $PSNR$ da imagem reconstruída, como visto no capítulo 7 de [1], além do fato de não ser uma tarefa inerente ao algoritmo MMP.

Este capítulo trata de uma nova abordagem para a solução dos dois problemas apresentados, aumentando o rendimento do algoritmo na codificação de imagens suaves e tornando a diminuição do efeito de blocagem uma tarefa inerente à codificação, o que reduz bastante a necessidade de filtragem na reconstrução. A principal característica dessa técnica reside no fato da seleção de um dado elemento para a codificação ser dependente das características de blocos anteriormente codificados e

vizinhos ao atual.

7.1 Descrição e implementação do algoritmo

Após a concepção e a implementação do algoritmo de otimização intermediário, procedeu-se à concepção de um algoritmo de *Side-match* [11, 12, 13, 14, 15, 16] adequado à estrutura do MMP, com o objetivo de melhorar o seu rendimento na codificação de imagens suaves, nas quais o mesmo é menor que o apresentado por algoritmos baseados em *Wavelets*. Além disso, essa melhoria não poderia acarretar queda de desempenho na codificação de imagens mais complexas, tais como imagens de texto ou mistas, nas quais o MMP já apresenta resultados diferenciados.

A essência do *Side-match* consiste em se fazer uma predição, a partir de elementos $\hat{X}_{up}^{m,n}$ e $\hat{X}_{left}^{m,n}$ previamente codificados e vizinhos ao bloco $X_l^{m,n}$ atual, mostrados na Figura 7.1 (igual à Figura 7.3, repetida aqui por conveniência), de quais seriam os elementos $S_i^{m,n}$ do dicionário $D^{m,n}$ e $S_{i_R}^{m,n}$ do dicionário $D_R^{m,n}$, no caso da otimização, mais propícios para a sua codificação. O resultado desta análise é a formação de um dicionário $D_S^{m,n}$ temporário e menor que $D^{m,n}$ ou $D^{m,n} \cup D_R^{m,n}$, onde provavelmente estaria o elemento $S_i^{m,n}$ ou $S_{i_R}^{m,n}$ escolhido caso fosse avaliado o dicionário em sua totalidade. Com um número menor de elementos no dicionário $D_S^{m,n}$ utilizado, o número de bits $\mathcal{R}$ necessário para se codificar um dado índice i é menor, resultando em imagens, a uma dada taxa (e.g. $0{,}5bpp$), com maior qualidade.

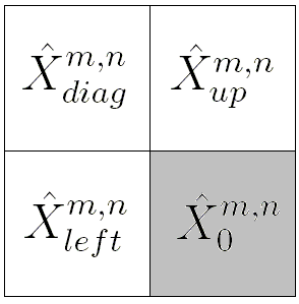


Figura 7.1: Blocos vizinhos utilizados para a formação do dicionário temporário.

Para a escolha dos elementos que compõem o dicionário $D_S^{m,n}$, adota-se um critério baseado no fato de que, em imagens suaves, os valores de *pixels* vizinhos são similares, ou até mesmo iguais, como demonstrado na Figura 7.2 para blocos de

4×4, com valores reais retirados da imagem *LENA* testada neste trabalho. Logo, os elementos escolhidos para o dicionário temporário $D_S^{m,n}$, comumente chamado de dicionário de estado, são aqueles que têm *pixels* de borda mais similares aos dos blocos vizinhos $\hat{X}_{up}^{m,n}$ e $\hat{X}_{left}^{m,n}$ já codificados.

Em princípio, além do aumento da qualidade objetiva da imagem ($PSNR$), obtém-se também um aumento da qualidade subjetiva, decorrente da diminuição do efeito de blocagem causado pelo processamento independente, realizado pelo MMP padrão, de cada bloco. Entretanto, em imagens nas quais a asserção de suavidade não é verdadeira, como as imagens de texto ou mistas, a escolha de blocos $S_i^{m,n}$ e $S_{i_R}^{m,n}$com valores de *pixels* de borda similares aos dos blocos $\hat{X}_{up}^{m,n}$ e $\hat{X}_{left}^{m,n}$ já codificados pode levar a decisões errôneas e a uma grande propagação de erro durante a codificação, resultando em imagens reconstruídas com baixa qualidade.

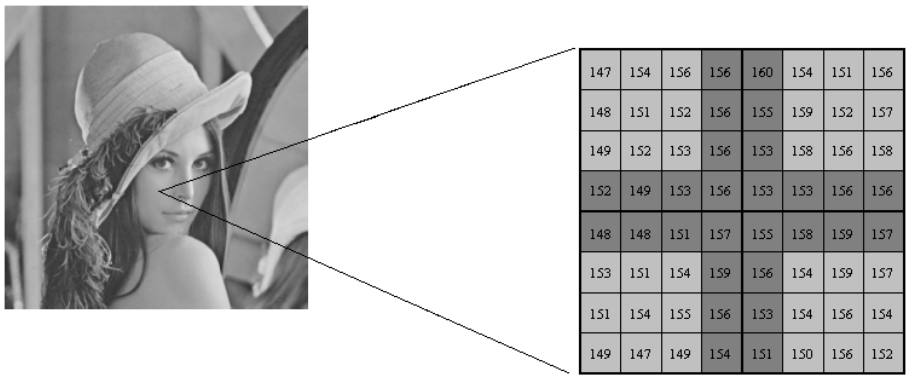


Figura 7.2: Demonstração da similaridade entre *pixels* em imagens suaves.

Apesar do que foi exposto, é possível conceber um algoritmo que funcione de forma adequada para um grande espectro de imagens, desde que o critério de continuidade seja bem projetado e lide com os casos de bordas de objetos ou transições de níveis de cinza nos limites dos blocos de entrada $X_l^{m,n}$. Tal critério foi empregado neste trabalho, o que é ainda facilitado pela construção adaptativa do dicionário D.

Outro aspecto de grande importância, presente em qualquer algoritmo de *Side-match*, é o dimensionamento do dicionário de estado $D_S^{m,n}$, que deve ter seu tamanho adequado à complexidade de codificação de cada bloco $X_l^{m,n}$, estimada através dos seus blocos vizinhos $\hat{X}_{up}^{m,n}$ e $\hat{X}_{left}^{m,n}$. Esta estimação deve ser feita devido ao não conhecimento, no decodificador, dos valores dos *pixels* não pertencentes às

bordas do bloco em decodificação. Assim, devido a esta incerteza, mais elementos são incluídos no dicionário de estado $D_S^{m,n}$, na tentativa de encontrar aquele que resulta no menor custo $\mathcal{J}(n_l)$.

A abordagem clássica para este problema dimensiona os dicionários $D_S^{m,n}$ baseando-se nas variâncias dos blocos vizinhos $\hat{X}_{up}^{m,n}$ e $\hat{X}_{left}^{m,n}$, porém, essa técnica não descreve com precisão a presença de bordas e detalhes nos mesmos [16]. Devido a isso, decidiu-se optar pelo cálculo da atividade de cada bloco (*AC - Activity*), mostrada a seguir para um vetor X de n posições:

$$AC(X) = \sum_{i=1}^{n-1} |x_i - x_{i+1}| \tag{7.1}$$

Esta nova medida permite identificar, de modo bastante eficaz, toda e qualquer atividade ou variação presente nos blocos vizinhos $\hat{X}_{up}^{m,n}$ e $\hat{X}_{left}^{m,n}$ analisados, não apresentando a diluição que ocorre no cálculo da variância, devido à média aritmética aplicada às diferenças quadráticas. Este último fato prejudica bastante o dimensionamento dos dicionários $D_S^{m,n}$. A fórmula para o cálculo da variância, considerando-se um vetor X de n posições, é mostrada abaixo:

$$\sigma^2(X) = \frac{1}{n} \sum_{i=1}^{n} (x_i - m(X))^2, \tag{7.2}$$

onde

$$m(X) = \frac{1}{n} \sum_{i=1}^{n} x_i.$$

Com o que foi apresentado, chega-se à conclusão de que, aparentemente, o critério de continuidade pode ser incorporado ao algoritmo MMP original sem muita dificuldade. Entretanto, poderia haver problemas devido ao fato do algoritmo ser capaz de partir o bloco de entrada, o que dificultaria a aplicação do critério de continuidade. Além disso, o algoritmo de *Side-match* deve ser executado tanto na codificação quanto na otimização, pois a árvore de segmentação também deve ser adequada aos dicionários de estado $D_S^{m,n}$ escolhidos para cada bloco $X_l^{m,n}$, ou seja, no momento de se calcular os custos $\mathcal{J}_l$ dos nós da árvore de segmentacção para a

decisão sobre a poda, tudo deve ocorrer utilizando-se os dicionários de estado $D_S^{m,n}$.

Após uma análise detalhada da estrutura do MMP, verificou-se que o ato de partir um bloco de entrada não se configura exatamente como um problema, bastando escolher corretamente o primeiro bloco, dentre os resultantes da divisão, a ser processado (sempre o superior ou o esquerdo), mantendo continuidade com os blocos $\hat{X}_{up}^{m,n}$ e $\hat{X}_{left}^{m,n}$ que já foram codificados.

A execução do algoritmo de *Side-match* realmente ocasiona um elevado custo computacional, cuja maior parcela é encontrada na otimização, devido ao fato de todos nós n_l da árvore de segmentação serem analisados. Como na codificação já se trabalha com a árvore de segmentação definitiva, os gastos computacionais para a geração dos dicionários de estado ocorrem somente ao se atingirem os nós-folhas, o que não é tão dispendioso.

O SM-MMP desenvolvido codifica os blocos $X_0^{m,n}$ no sentido de leitura, ou seja, da esquerda para a direita e de cima para baixo, começando do bloco superior esquerdo. Esta ordem de processamento dos blocos de entrada ajuda bastante na codificação da imagem em processamento, pois inclui, no dicionário, elementos com grande probabilidade de serem utilizados nos blocos imediatamente à direita do atual (foram realizados vários testes e esta técnica se mostrou a mais eficaz). Apenas o primeiro bloco $X_0^{m,n}$ (superior esquerdo) é codificado com o MMP padrão, utilizando o algoritmo de otimização intermediário. Este bloco é chamado de bloco fundamental e é codificado utilizando-se o dicionário oficial D em sua totalidade. É interessante ressaltar que, neste ponto do processamento, o dicionário oficial D ainda é pequeno.

No momento da divisão de um determinado bloco $X_l^{m,n}$, a primeira parte a ser processada é sempre a superior ($m > n$) ou a esquerda ($m \leq n$), objetivando-se manter continuidade com os blocos $\hat{X}_{up}^{m,n}$ e $\hat{X}_{left}^{m,n}$ que já foram codificados, permitindo a aplicação do critério de continuidade e o correto dimensionamento dos dicionários de estado $D_S^{m,n}$.

Os três vizinhos (superior, esquerdo e diagonal) do bloco $X_0^{m,n}$ que está sendo codificado são copiados para um *buffer* de quatro posições intermediário $\mathcal{B}^{4m,4n}$, onde também são escritos todos os blocos $\hat{X}_l^{m,n}$ resultantes da otimização ou da codificação, facilitando os cálculos do critério de continuidade. As aproximações do bloco atual $X_0^{m,n}$ são escritas na posição inferior direita e, ao processar suas divisões,

os algoritmos de codificação e otimização recebem as coordenadas $(\mathcal{V}_n, \mathcal{V}_m)$ dos seus vértices superiores esquerdos. O *buffer* em questão é mostrado na Figura 7.3.

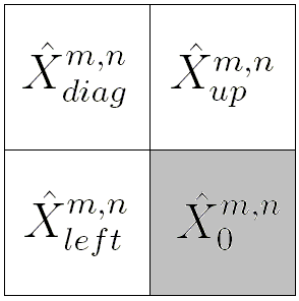


Figura 7.3: Buffer $\mathcal{B}^{4m,4n}$ para a realização dos cálculos de *Side-match*.

Todos os elementos dos dicionários $D^{m,n}$ e $D^{m,n}_R$ são dotados de sinalizadores que informam se os mesmos fazem parte do dicionário de estado $D^{m,n}_S$ ou não. Esta é a alternativa mais fácil para a implementação do *Side-match* na estrutura do MMP. Os elementos escolhidos para o dicionário de estado $D^{m,n}_S$ são aqueles que apresentam os menores valores de rugosidade (*RG - Rugosity*), dada pelo critério de continuidade descrito na fórmula (7.3) e representada na Figura 7.4;

$$\begin{aligned}
\text{Vert_rg}(X) &= \sum_{i=1}^{n} \left| \left\lfloor \frac{(u_{m-1,i} - u_{m,i}) + (x_{1,i} - x_{2,i})}{2} \right\rfloor - (u_{m,i} - x_{1,i}) \right|, \\
\text{Horiz_rg}(X) &= \sum_{j=1}^{m} \left| \left\lfloor \frac{(l_{j,n-1} - l_{j,n}) + (x_{j,1} - x_{j,2})}{2} \right\rfloor - (l_{j,n} - x_{j,1}) \right|, \\
RG\,(X) &= \text{Vert_rg}(X) + \text{Horiz_rg}(X).
\end{aligned} \tag{7.3}$$

Antes do início do processo de codificação, calcula-se a maior atividade AC presente na imagem original ($AC_{img_{max}}$), levando-se em consideração o tamanho do bloco de entrada de maior hierarquia $X^{m,n}_0$, ou seja, se o tamanho do bloco de codificação for 16×16, calcula-se a AC de todos os blocos 16×16 da imagem original e armazena-se o maior valor encontrado. A média aritmética de todas as ACs também é calculada e armazenada (AC_{avg}). Calcula-se, então, o máximo tamanho que um dicionário de estado ($l_{max}\,(D^{m,n}_S)$) pode ter, baseando-se em $AC_{img_{max}}$ e AC_{avg}. Este valor delimitará um dicionário máximo adequado a cada tipo de imagem. O cálculo

de $l_{max}\,(D_S^{m,n})$ é mostrado a seguir e a sua curva aproximada está traçada na Figura 7.5.

Passo 1: $AC_{final} = \left\lfloor \frac{AC_{img_{max}}}{4} \right\rfloor + AC_{avg}$

Passo 2: Se $AC_{final} \geq 285$, $l_{max}\,(D_S^{m,n}) = \left\lfloor \frac{(AC_{final} - 285).6000}{39} \right\rfloor + 5000$

Senão, $l_{max}\,(D_S^{m,n}) = \left\lfloor \frac{(AC_{final}).4999}{285} \right\rfloor + 1$

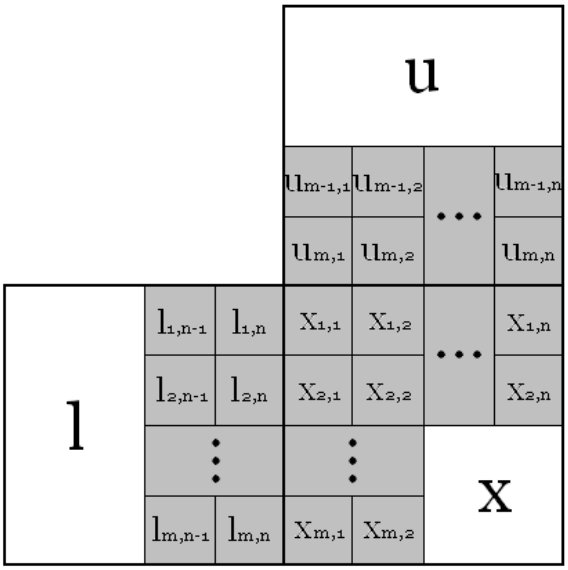


Figura 7.4: *Pixels* considerados para o cálculo da rugosidade.

A curva da Figura 7.5 é uma aproximação para a curva original para $l_{max}\,(D_S^{m,n})$, que foi levantada a partir dos tamanhos ideais necessários para o máximo desempenho na codificação de cada imagem de teste, apresentando valores maiores nas imagens que possuem maiores AC_{avg} e $AC_{img_{max}}$. O seu aspecto original é a de uma exponencial dependente das ACs das imagens. Após uma extensa análise, constatou-se que esta curva poderia ser linearizada em duas partes com base em AC_{avg} e $AC_{img_{max}}$ (AC_{final}), com ponto de interseção entre as retas próximo de $AC_{final} = 285$. Não há necessidade dos valores $l_{max}\,(D_S^{m,n})$ serem precisos, pois praticamente o mesmo resultado pode ser obtido dentro de uma larga faixa em torno do valor ideal. Por exemplo, a imagem *LENA* pode ser comprimida com a mesma qualidade utilizando-se valores de $l_{max}\,(D_S^{m,n})$ entre 4500 e 5500 elementos ($\approx \pm 10\%$).

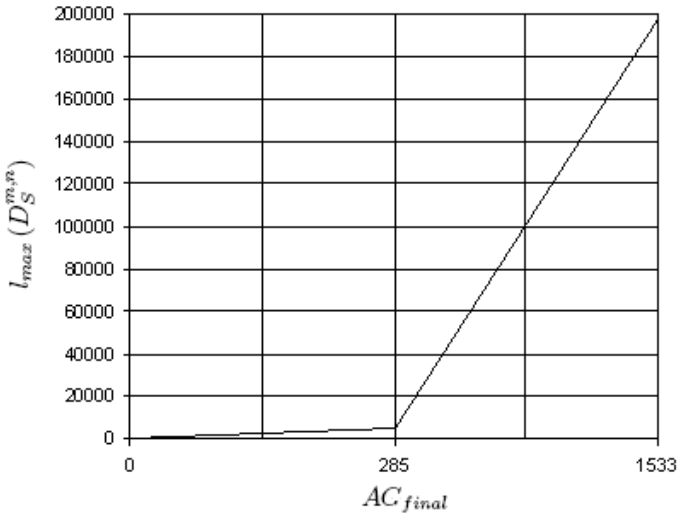


Figura 7.5: Curva aproximada para a escolha de $l_{max}\left(D_S^{m,n}\right)$.

O tamanho do dicionário de estado, para cada bloco $X_l^{m,n}$ a ser codificado, é escolhido de acordo com razão entre a média das ACs dos blocos vizinhos (superior e esquerdo), mostrados nas Figuras 7.3 e 7.4, e $AC_{img_{max}}$. É interessante observar que os blocos vizinhos podem ser blocos de entrada $\hat{X}_{up}^{m,n}$ e $\hat{X}_{left}^{m,n}$ completos já codificados, partes dos blocos de entrada já codificados ou partes já codificadas do bloco de entrada $X_0^{m,n}$ que está sendo processado; tudo depende da posição atual na árvore de segmentação do bloco $X_0^{m,n}$. Para se obter o valor final do tamanho do dicionário de estado $D_S^{m,n}$ $\left(l\left(D_S^{m,n}\right)\right)$, calculam-se as ACs de todas as linhas e colunas do bloco vizinho em questão (superior ou esquerdo) e escolhe-se o maior valor; calcula-se, então, a média aritmética dos valores máximos encontrados para os vizinhos superior e esquerdo e divide-se o mesmo por $AC_{img_{max}}$. Este resultado, então, multiplica $l_{max}\left(D_S^{m,n}\right)$. O processo de cálculo do valor final da AC de um bloco X de dimensões $m \times n$ é dado por:

$$
\begin{aligned}
AC_h\left(X\right) &= \max_{1\leq i\leq m} \sum_{j=1}^{n-1} \left|x_{i,j} - x_{i,j+1}\right|, \\
AC_v\left(X\right) &= \max_{1\leq j\leq n} \sum_{i=1}^{m-1} \left|x_{i,j} - x_{i+1,j}\right|, \\
AC_{bloco}\left(X\right) &= max\left\{AC_h\left(X\right), AC_v\left(X\right)\right\}. \qquad (7.4)
\end{aligned}
$$

O principal fator que define o rendimento do algoritmo é o critério de continuidade. Se o mesmo for mal projetado, serão escolhidos blocos para o dicionário de estado $D_S^{m,n}$ muito diferentes daquele presente na imagem original. Por exemplo, suponha que o critério de continuidade seja dado por:

$$\begin{aligned} \text{Vert\_rg}(X) &= \sum_{i=1}^{n} \left|(u_{m,i} - x_{1,i})\right|, \\ \text{Horiz\_rg}(X) &= \sum_{j=1}^{m} \left|(l_{j,n} - x_{j,1})\right|, \\ RG\,(X) &= \text{Vert\_rg}(X) + \text{Horiz\_rg}(X). \end{aligned} \tag{7.5}$$

e o mesmo seja aplicado aos blocos mostrados na Figura 7.6.

153	152	152	148	143	139	138	138
154	153	153	149	144	139	137	137
153	153	151	147	143	138	139	138
152	153	152	148	145	141	136	136

Figura 7.6: Exemplo de valores de *pixels* nas bordas de blocos de imagem vizinhos.

O bloco da esquerda já foi codificado e o da direita é o que está sendo analisado. Como resultado do critério, serão escolhidos elementos para o dicionário de estado que têm seus *pixels* de borda mais próximos de 148, 149, 147 e 148. Entretanto, é fácil perceber que existe uma transição entre os blocos, com valores de *pixels* decrescentes entre as bordas, um fato extremamente comum em imagens reais. Como o critério não está preparado para isto, os elementos escolhidos apresentarão grande distorção em relação ao bloco original, mesmo para *pixels* próximos das bordas, resultando também numa grande propagação de erro para a realização do casamento com os demais blocos.

Utilizando-se o critério apresentado neste trabalho (ver equação (7.3)), a escolha dos elementos para a codificação do bloco mostrado na Figura 7.6 seria muito mais exata, pois são consideradas transições entre os blocos, sejam estas crescentes

ou decrescentes, utilizando-se não um, mas dois *pixels* de borda. Entretanto, como o SM-MMP pode realizar a divisão dos blocos, em alguns casos a sua aplicação fica comprometida, como em blocos de 2×1. Nestes, o cálculo da distorção horizontal levará em consideração apenas 1 *pixel*, que pode ainda ser muito diferente daquele presente na borda do seu vizinho.

É interessante ainda observar que, durante a otimização, os dicionários de estado $D_S^{m,n}$ de alguns blocos são gerados através do casamento lateral com blocos $\hat{X}_l^{m,n}$ que foram previamente analisados pela otimização, mas ainda não são exatamente os escolhidos (definitivos), pois a árvore de segmentação pode ser podada mais adiante. Este aspecto é demonstrado na Figura 7.7, onde um exemplo com blocos de 4×4 é apresentado. Os nós N_1 e N_5 são nós-folhas provisórios, porém, N_6 é escolhido através da avaliação da rugosidade com relação aos mesmos. Após isso, N_5 e N_6 podem ser substituídos por N_2 (que teve seus cálculos de rugosidade avaliados em relação a N1), e N_1 e N_2 por N_0.

O casamento lateral com blocos provisórios, que foi explicado acima, é uma característica extremamente importante do SM-MMP e requer uma área de memória separada para que todas as avaliações sejam feitas. Essa área de memória facilita também a implementação da super-atualização de dicionário, comentada no próximo capítulo.

Deste modo, para que seja possível uma análise rápida e eficaz da distorção lateral de cada bloco, deve-se utilizar $\mathcal{B}^{4m,4n}$ (ver figura 7.3 e explicação que a antecede), onde se escrevem os blocos previamente analisados para que sirvam de casamento lateral para os demais.

Na codificação, também é possível que blocos em análise façam casamento lateral com partes do bloco de entrada atual $X_0^{m,n}$, porém, estas já são codificações definitivas.

Dependendo da localização do bloco de entrada $X_0^{m,n}$, nem todas as suas partes têm como fazer casamento lateral com $\hat{X}_{up}^{m,n}$ e $\hat{X}_{left}^{m,n}$. No que diz respeito aos elementos da primeira linha ou coluna de blocos $X_0^{m,n}$ da imagem, em muitos casos (depende da localização na árvore de segmentação) só há o bloco esquerdo ou o superior, respectivamente, o que é mostrado na Figura 7.8. Assim, apenas um dos valores é utilizado para o cálculo de $l\,(D_S^{m,n})$.

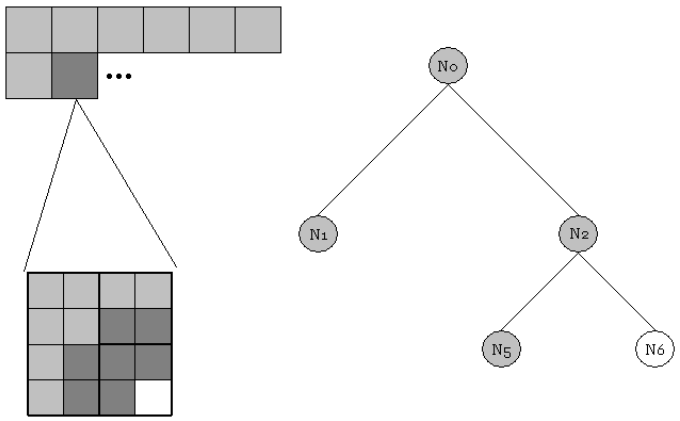


Figura 7.7: Casamento lateral durante a otimização.

O critério para a inserção de elementos no dicionário de estado $D_S^{m,n}$ é simples: são escolhidos os $l\left(D_S^{m,n}\right)$ elementos que possuem as menores rugosidades (critério de continuidade RG descrito na equação (7.3)), ordenados segundo as mesmas. O primeiro elemento é o de menor rugosidade e o último o de maior. Caso dois elementos possuam a mesma rugosidade, o mais novo é colocado uma posição à frente. Além disso, se o elemento que se pretende incluir já existe em $D_S^{m,n}$, a operação é cancelada e o próximo elemento passa e ser analisado.

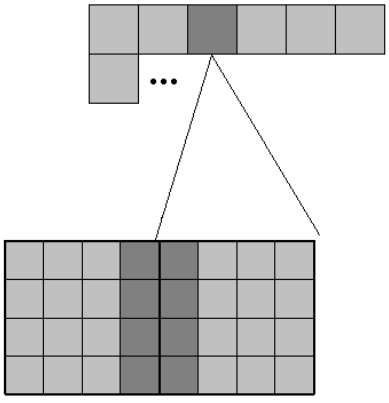


Figura 7.8: Exemplo no qual um bloco da primeira linha realiza casamento lateral apenas com o vizinho esquerdo.

É interessante observar que, durante a atualização de $D^{m,n}$ ou $D_R^{m,n}$, não existe qualquer mecanismo para verificar se um dado elemento já foi inserido (a não ser para blocos de 1×1), devido ao fato desta operação ser extremamente demorada.

No presente algoritmo, tal operação é desnecessária, pois haverá uma nova seleção para o dicionário de estado $D_S^{m,n}$ e a busca neste último pode ser implementada de forma rápida e eficiente.

Como resultado do algoritmo para a construção do dicionário de estado $D_S^{m,n}$, um sinalizador é atribuído a cada elemento de $D^{m,n}$ ou $D_R^{m,n}$. Seu objetivo é indicar se um dado elemento faz parte do dicionário de estado $D_S^{m,n}$ ou não.

A função de geração dos dicionários de estado $D_S^{m,n}$ será chamada a cada iteração da função de otimização ou codificação, para que cada elemento seja analisado ou codificado utilizando-se seu próprio dicionário de estado $D_S^{m,n}$.

Em decorrência da geração do dicionário de estado $D_S^{m,n}$, no momento da codificação ou da otimização, um novo modelo de freqüências temporário é criado, contendo apenas os elementos do dicionário de estado $D_S^{m,n}$, ou seja, as freqüências acumuladas são somatórios apenas das freqüências dos elementos escolhidos e não do dicionário oficial em sua totalidade. Com isso, os cálculos de custo $\mathcal{J}_l$ e o codificador aritmético atuam apenas nesses elementos, diminuindo significativamente a taxa $\mathcal{R}$ necessária para a codificação dos seus índices.

Por exemplo, se um dicionário de 1000 elementos (cada um com freqüência 1, com um total de freqüências acumuladas de 1000) gerar um dicionário de estado com 50 elementos, a redução na taxa para a codificação de qualquer dos índices, de acordo com a equação (2.1), será de:

$$\begin{aligned}
&\mathcal{R} = log_2\left(\frac{1}{P\left(S^{m,n}\right)}\right), \\
&log_2\left(\frac{50}{1}\right) = 5,6438\ bits,\ log_2\left(\frac{1000}{1}\right) = 9,9657\ bits, \\
&\Delta\mathcal{R} = log_2\left(\frac{50}{1}\right) - log_2\left(\frac{1000}{1}\right) = -4,3219\ bits.
\end{aligned} \tag{7.6}$$

Outro aspecto importante do algoritmo SM-MMP consiste no fato de que o dicionário de estado $D_S^{m,n}$ só não é gerado para os elementos de dimensões 1×1, pois a distorção conseqüente da restrição do dicionário oficial $D^{m,n}$ ou $D^{m,n} \cup D_R^{m,n}$ é muito alta e o algoritmo não mais seria capaz de comprimir com distorção zero (ver seção 5.1), caso isto fosse necessário.

O dicionário de estado $D_S^{m,n}$ gerado contém elementos de $D^{m,n}$ e, no caso da

função de otimização, também de $D_R^{m,n}$, com um modelo de freqüências totalmente novo e adequado ao mesmo. Uma vez que este dicionário esteja pronto, todos os cálculos de custo $\mathcal{J}_l$ devem ser realizados utilizando-se o novo modelo. O principal ganho deste algoritmo é a grande diminuição obtida no novo dicionário $D_S^{m,n}$, com um tamanho adequado ao bloco $X_l^{m,n}$ a ser codificado; blocos com vizinhos de elevada AC_{final} possuem dicionários $D_S^{m,n}$ maiores, enquanto aqueles com AC_{final} menor possuem dicionários menores.

O dicionário inicial para o SM-MMP é de 128 vetores igualmente espaçados entre 0 e 254. Esse aumento do tamanho dicionário inicial, em relação ao utilizado no RDI-MMP, é devido principalmente à restrição sofrida para a formação do dicionário de estado $D_S^{m,n}$, que permite uma grande utilização dos elementos do último nível (1×1). Logo, para melhores resultados, os elementos devem ser mais próximos dos originais. A equação de formação do dicionário é dada por:

$$
\begin{aligned}
&\text{valor\_pixel\_mínimo} = 0,\\
&\text{valor\_pixel\_máximo} = 255,\\
&\mathsf{L} = 128\\
&\mathsf{P} = \left\lfloor \frac{(\text{valor\_pixel\_máximo} - \text{valor\_pixel\_mínimo})}{\mathsf{L}-1} \right\rfloor = 2,\\
&\text{valor\_pixel\_máximo} = \mathsf{P}.(\mathsf{L}-1) = 254,
\end{aligned}
$$

$$
\begin{aligned}
&\text{Para } n = 0, 1, \ldots, \mathsf{N}_D - 1,\\
&p = 2^{\lfloor \frac{n+1}{2} \rfloor},\ q = 2^{\lfloor \frac{n}{2} \rfloor},\\
&D_o^{p,q} = \left\{ T_{1,1}^{p,q}[0], T_{1,1}^{p,q}[2], \ldots, T_{1,1}^{p,q}[252], T_{1,1}^{p,q}[254] \right\}
\end{aligned}
\tag{7.7}
$$

O algoritmo para a geração dos dicionários de estado $D_S^{m,n}$ está descrito na próxima página.

Procedimento $\{\mathcal{F}(D^{m,n}), \mathcal{F}(D_R^{m,n})\} = \text{gera\_dic\_estado}(\mathcal{V}_n, \mathcal{V}_m, \eta_o)$

Passo 1: Se $\eta_o == 0$ (escala 1×1), faz $\mathcal{F}(D^{m,n})_h = 1$, para $h = 1, 2, \ldots, l(D^{m,n})$, $\mathcal{F}(D_R^{m,n})_h = 1$, para $h = 1, 2, \ldots, l(D_R^{m,n})$, e retorna $\mathcal{F}(D^{m,n})$. O modelo de freqüências utilizado é o mesmo do dicionário oficial $D^{m,n}$ (ou $D^{m,n} \cup D_R^{m,n}$).

Senão, vai para o **Passo 2**.

Passo 2: Calcula $AC_{bloco}\left(\hat{X}_{up}^{m,n}\right)$ (se $\hat{X}_{up}^{m,n}$ existir ou se $\mathcal{V}_m > m$ em $\mathcal{B}^{4m,4n}$) e $AC_{bloco}\left(\hat{X}_{left}^{m,n}\right)$ (se $\hat{X}_{left}^{m,n}$ existir ou se $\mathcal{V}_n > n$ em $\mathcal{B}^{4m,4n}$)

Passo 3: Calcula $AC_{bloco_{avg}} = \frac{\left(AC_{bloco}\left(\hat{X}_{up}^{m,n}\right) + AC_{bloco}\left(\hat{X}_{left}^{m,n}\right)\right)}{2}$. Se existir apenas um, a média é igual ao valor deste.

Passo 4: Se $l(D^{m,n}) > l_{max}(D_S^{m,n})$, $l(D_S^{m,n}) = \frac{\left(AC_{bloco_{avg}} . l_{max}\left(D_S^{m,n}\right)\right)}{AC_{img_{max}}}$.

Senão, $l(D_S^{m,n}) = \frac{\left(AC_{bloco_{avg}} . l(D^{m,n})\right)}{AC_{img_{max}}}$.

Obs: Se a chamada for na função de otimização, $l(D^{m,n}) = l(D^{m,n}) + l(D_R^{m,n})$.

Passo 5: Se $l(D_S^{m,n}) < l_{min}(D_S^{m,n})$, $l(D_S^{m,n}) = l_{min}(D_S^{m,n})$.

Senão, vai para o **Passo 6**.

Obs: $l_{min}(D_S^{m,n})$ é o menor tamanho possível de $D_S^{m,n}$, considerado 16 neste trabalho.

Passo 6: Procura, em $D^{m,n}$, os elementos $S_i^{m,n}$ com menores $RG(S_i^{m,n})$ e os coloca em $D_S^{m,n}$, fazendo $\mathcal{F}(D^{m,n})_i = 1$. Antes do elemento $S_i^{m,n}$ ser inserido em $D_S^{m,n}$, verifica-se se já não existe um igual. Caso isso seja verdade, o elemento $S_i^{m,n}$ em questão é descartado. Os elementos de $D_S^{m,n}$ são ordenados de acordo com as suas RGs. Se $D_S^{m,n}$ for completamente preenchido e aparecer um elemento $S_i^{m,n}$ com uma RG menor que a do elemento na última posição (elemento de maior RG), retira-se este último, acrescenta-se o novo elemento e reordena-se $D_S^{m,n}$.

Obs: Se não for possível calcular Vert_rg(X) ou Horiz_rg(X), seu valor é considerado zero.

Passo 7: Caso a chamada seja na função de otimização, repete **Passo 6** para $D_R^{m,n}$ e $\mathcal{F}(D_R^{m,n})$. É importante sinalizar a origem de cada elemento de $D_S^{m,n}$,

pois caso seja encontrado um elemento $S_{i_R}^{m,n}$ de menor RG ou este seja o escolhido para representar o bloco $X_l^{m,n}$, sabe-se onde procurá-lo.

Passo 8: Cria um novo modelo de freqüências com os elementos de $D_S^{m,n}$. Os elementos, agora, serão ordenados de acordo com suas freqüências relativas e formarão um novo modelo para o codificador aritmético. Deste modo, quando o custo $J(n_l)$ de um elemento for calculado, tanto na otimização quanto na codificação, será de acordo com este novo modelo.

Passo 9: Retorna $\mathcal{F}(D^{m,n})$ e $\mathcal{F}(D_R^{m,n})$.

Como agora os cálculos de custo e as pesquisas para se encontrar um elemento aproximado estão vinculados a um dicionário de estado $D_S^{m,n}$, que deve ser criado para cada bloco de entrada $X_l^{m,n}$, os algoritmos de codificação e otimização precisam sofrer novas modificações, mostradas nas próximas páginas.

Procedimento $\left\{\hat{X}_l^{m,n}, \mathcal{T}\right\} = \text{otimiza}\left(X_l^{m,n}, \mathcal{T}_o, \eta_o, \mathcal{V}_n, \mathcal{V}_m\right)$

Passo 1: Faz $\mathcal{T} = \mathcal{T}_0$.

Passo 2: Computa $\left\{\mathcal{F}\left(D^{m,n}\right), \mathcal{F}\left(D_R^{m,n}\right)\right\} = \text{gera\_dic\_estado}\left(\mathcal{V}_n, \mathcal{V}_m, \eta_o\right)$.

Passo 3: Procura no dicionário $D^{m,n}$, onde $m = 2^{\left\lfloor \frac{\eta_o+1}{2} \right\rfloor}$ e $n = 2^{\left\lfloor \frac{\eta_o}{2} \right\rfloor}$, o elemento $S_i^{m,n}$, com $\mathcal{F}\left(D^{m,n}\right)_i == 1$, que representa $X_l^{m,n}$ com menor custo $\mathcal{J}(n_l)$, armazenando i, $\mathcal{J}(n_l)$ e dicionário de origem. O elemento $S_i^{m,n}$ é armazenado em $\hat{X}_l^{m,n}$. O custo $\mathcal{J}(n_l)$ é calculado levando-se em consideração $\sum C_h^{m,n}$, $\sum \overline{C}_h^{m,n}$, $\sum C_{h_R}^{m,n}$, para todo h tal que $\mathcal{F}\left(D^{m,n}\right)_h == 1$ e h_R tal que $\mathcal{F}\left(D^{m,n}\right)_{h_R} == 1$, $C_i^{m,n}$ e $\overline{C}_i^{m,n}$.

Passo 4: Varre o dicionário $D_R^{m,n}$ e verifica se o elemento $S_{i_R}^{m,n}$, com $\mathcal{F}\left(D^{m,n}\right)_{i_R} == 1$, de menor custo representa $X_l^{m,n}$ com custo $\mathcal{J}(n_l)$ menor que o do escolhido no **Passo 3**. Se isso ocorrer, substitui $S_i^{m,n}$ por $S_{i_R}^{m,n}$, armazenando i_R, $\mathcal{J}(n_l)$ e dicionário de origem. O custo $\mathcal{J}(n_l)$ é calculado levando-se em consideração $\sum C_h^{m,n}$, $\sum \overline{C}_h^{m,n}$, $\sum C_{h_R}^{m,n}$, para todo h tal que $\mathcal{F}\left(D^{m,n}\right)_h == 1$ e h_R tal que $\mathcal{F}\left(D^{m,n}\right)_{h_R} == 1$, e $C_{i_R}^{m,n}$.

Passo 5: Se a escala atual for $\eta_o == 0$, ou seja, 1×1 ($m == 1$ e $n == 1$), incrementa $\overline{C}_i^{m,n}$ ou $C_{i_R}^{m,n}$, dependendo da origem do elemento $S^{m,n}$ escolhido, escreve $S^{m,n}$ na sua devida posição em $\mathcal{B}^{4m,4n}$ e retorna $\hat{X}_l^{m,n}$ e $\mathcal{T}$.

Senão, vai para o **Passo 6**.

Passo 6: Acrescenta, ao custo $\mathcal{J}(n_l)$ calculado, o valor $\lambda.\mathcal{R}_{1_l}$, para representar completamente o custo do nó-folha. A taxa do *flag* '1' deve ser calculada com $\sum C_{h_F}^{m,n}$, $\sum \overline{C}_{h_F}^{m,n}$, $C_{1_F}^{m,n}$ e $\overline{C}_{1_F}^{m,n}$.

Passo 7: Calcula e armazena, separadamente, o valor $\lambda.\mathcal{R}_{0_l}$, que posteriormente complementará o custo dos nós-filhos. A taxa do *flag* '0' deve ser calculada com $\sum C_{h_F}^{m,n}$, $\sum \overline{C}_{h_F}^{m,n}$, $C_{0_F}^{m,n}$ e $\overline{C}_{0_F}^{m,n}$.

Passo 8: Incrementa o contador $\overline{C}_{0_F}^{m,n}$.

Passo 9: Se $m > n$, divide $X_l^{m,n}$ em $\begin{pmatrix} X_{2l+1}^{k,j} \\ X_{2l+2}^{k,j} \end{pmatrix}$, onde $k = \frac{m}{2}$ e $j = n$, e faz $\mathcal{V}_{1m} = \mathcal{V}_m$, $\mathcal{V}_{1n} = \mathcal{V}_n$, $\mathcal{V}_{2m} = \mathcal{V}_m + \frac{m}{2}$ e $\mathcal{V}_{2n} = \mathcal{V}_n$.

Senão, divide $X_l^{m,n}$ em $\left(X_{2l+1}^{k,j}\ X_{2l+2}^{k,j}\right)$, onde $k = m$ e $j = \frac{n}{2}$, e faz $\mathcal{V}_{1m} = \mathcal{V}_m$, $\mathcal{V}_{1n} = \mathcal{V}_n$, $\mathcal{V}_{2m} = \mathcal{V}_m$ e $\mathcal{V}_{2n} = \mathcal{V}_n + \frac{n}{2}$.

Passo 10: Computa $\left\{\hat{X}_{2l+1}^{k,j}, \mathcal{T}_1\right\} = \text{otimiza}\left(X_{2l+1}^{k,j}, \mathcal{T}, \eta_o - 1, \mathcal{V}_{1n}, \mathcal{V}_{1m}\right)$

Passo 11: Computa $\left\{\hat{X}_{2l+2}^{k,j}, \mathcal{T}_2\right\} = \text{otimiza}\left(X_{2l+2}^{k,j}, \mathcal{T}, \eta_o - 1, \mathcal{V}_{2n}, \mathcal{V}_{2m}\right)$

Passo 12: Faz $\mathcal{T} = (\mathcal{T}_1)\,\text{AND}\,(\mathcal{T}_2)$.

Passo 13: Se o $\mathcal{J}_l \leq \mathcal{J}_{2l+1} + \mathcal{J}_{2l+2} + \lambda.\mathcal{R}_{0_l}$, vai para o **Passo 14**.

Senão, vai para o **Passo 20**.

Passo 14: Decrementa os contadores $\overline{C}_{0_F}^{w,y}$ em todas as escalas $w \times y$ relacionadas aos nós das sub-árvores $S(n_{2l+1})$ e $S(n_{2l+2})$ a serem podadas.

Passo 15: Decrementa os contadores $\overline{C}_{1_F}^{w,y}$ e $\overline{C}_{h}^{w,y}$ ou $\overline{C}_{h_R}^{w,y}$ nas escalas $w \times y$ referentes aos nós-folhas das sub-árvores $S(n_{2l+1})$ e $S(n_{2l+2})$ e nas posições dos seus índices (deve ser verificada a origem de cada elemento, ou seja, se é de D ou de D_R).

Passo 16: Decrementa o contador $\overline{C}_{0_F}^{m,n}$.

Passo 17: Incrementa o contador $\overline{C}_{1_F}^{m,n}$ e o contador $\overline{C}_{i}^{m,n}$ ou $C_{i_R}^{m,n}$, dependendo da origem do elemento $S^{m,n}$ utilizado como aproximação.

Passo 18: Elimina as atualizações do dicionário D_R ocasionadas pelas sub-árvores $S(n_{2l+1})$ e $S(n_{2l+2})$ a serem podadas.

Passo 19: Indica, em $\mathcal{T}$, que as sub-árvores $S(n_{2l+1})$ e $S(n_{2l+2})$ foram podadas, escreve o elemento $S^{m,n}$ escolhido na sua devida posição em $\mathcal{B}^{4m,4n}$ e retorna $\hat{X}_l^{m,n}$ e $\mathcal{T}$.

Passo 20: Se $m > n$, faz $\hat{X}_l^{m,n} = \begin{pmatrix} \hat{X}_{2l+1}^{k,j} \\ \hat{X}_{2l+2}^{k,j} \end{pmatrix}$.

Senão, faz $\hat{X}_l^{m,n} = \begin{pmatrix} \hat{X}_{2l+1}^{k,j} & \hat{X}_{2l+2}^{k,j} \end{pmatrix}$.

Passo 21: Atualiza o dicionário D_R em todas as escalas com $\hat{X}_l^{m,n}$, ou seja:

Para $n = 0, 1, \ldots, \mathsf{N}_D - 1$,

$p = 2^{\left\lfloor \frac{n+1}{2} \right\rfloor}$, $q = 2^{\left\lfloor \frac{n}{2} \right\rfloor}$,

$D^{p,q} = D^{p,q} \cup T_{m,n}^{p,q}\left[\hat{X}_l^{m,n}\right]$.

Passo 22: Faz $\mathcal{J}_l = \mathcal{J}_{2l+1} + \mathcal{J}_{2l+2} + \lambda.\mathcal{R}_{0_l}$

Passo 23: Retorna $\hat{X}_l^{m,n}$ e $\mathcal{T}$.

Procedimento $\hat{X}_l^{m,n} = \text{codifica}\left(X_l^{m,n}, \eta_o, \mathcal{T}, \mathcal{V}_n, \mathcal{V}_m\right)$

Passo 1: Se $\mathcal{T}_{2l+1} == 0$ e $\mathcal{T}_{2l+2} == 0$, ou $\eta_o = 0$, vai para o **Passo 2**.

Senão, vai para o **Passo 6**.

Passo 2: Computa $\mathcal{F}\left(D^{m,n}\right) = \text{gera\_dic\_estado}\left(\mathcal{V}_n, \mathcal{V}_m, \eta_o\right)$.

Passo 3: Procura, no dicionário $D^{m,n}$, onde $m = 2^{\left\lfloor \frac{\eta_o+1}{2} \right\rfloor}$ e $n = 2^{\left\lfloor \frac{\eta_o}{2} \right\rfloor}$, o elemento $S_i^{m,n}$, com $\mathcal{F}\left(D^{m,n}\right)_i == 1$, que representa $X_l^{m,n}$ com menor custo $\mathcal{J}(n_l)$, armazenando-o em $\hat{X}_l^{m,n}$. O custo $\mathcal{J}(n_l)$ é calculado levando-se em consideração $\sum C_h^{m,n}$, para todo h tal que $\mathcal{F}\left(D^{m,n}\right)_h == 1$, e $C_i^{m,n}$.

Passo 4: Se a escala atual for $\eta_o == 0$, ou seja, 1×1 ($m == 1$ e $n == 1$), codifica o índice i do elemento $S_i^{m,n}$ escolhido, escreve $S_i^{m,n}$ na sua devida posição em $\mathcal{B}^{4m,4n}$ e retorna $\hat{X}_l^{m,n}$.

Senão, vai para o **Passo 5**.

Passo 5: Codifica o *flag* '1', codifica o índice i do elemento $S_i^{m,n}$ escolhido, escreve $S_i^{m,n}$ na sua devida posição em $\mathcal{B}^{4m,4n}$ e retorna $\hat{X}_l^{m,n}$.

Passo 6: Codifica o *flag* '0'.

Passo 7: Se $m > n$, divide $X_l^{m,n}$ em $\begin{pmatrix} X_{2l+1}^{k,j} \\ X_{2l+2}^{k,j} \end{pmatrix}$, onde $k = \frac{m}{2}$ e $j = n$, e faz $\mathcal{V}_{1m} = \mathcal{V}_m$, $\mathcal{V}_{1n} = \mathcal{V}_n$, $\mathcal{V}_{2m} = \mathcal{V}_m + \frac{m}{2}$ e $\mathcal{V}_{2n} = \mathcal{V}_n$.

Senão, divide $X_l^{m,n}$ em $\left(X_{2l+1}^{k,j}\ X_{2l+2}^{k,j}\right)$, onde $k = m$ e $j = \frac{n}{2}$, e faz $\mathcal{V}_{1m} = \mathcal{V}_m$, $\mathcal{V}_{1n} = \mathcal{V}_n$, $\mathcal{V}_{2m} = \mathcal{V}_m$ e $\mathcal{V}_{2n} = \mathcal{V}_n + \frac{n}{2}$.

Passo 8: Computa $\hat{X}_{2l+1}^{k,j} = \text{codifica}\left(X_{2l+1}^{k,j}, \eta_o - 1, \mathcal{T}, \mathcal{V}_{1n}, \mathcal{V}_{1m}\right)$

Passo 9: Computa $\hat{X}_{2l+2}^{k,j} = \text{codifica}\left(X_{2l+2}^{k,j}, \eta_o - 1, \mathcal{T}, \mathcal{V}_{2n}, \mathcal{V}_{2m}\right)$

Passo 10: Se $m > n$, faz $\hat{X}_l^{m,n} = \begin{pmatrix} \hat{X}_{2l+1}^{k,j} \\ \hat{X}_{2l+2}^{k,j} \end{pmatrix}$.

Senão, faz $\hat{X}_l^{m,n} = \left(\hat{X}_{2l+1}^{k,j}\ \hat{X}_{2l+2}^{k,j}\right)$.

Passo 11: Atualiza o dicionário D em todas as escalas com $\hat{X}_l^{m,n}$, ou seja:

Para $n = 0, 1, \ldots, \mathsf{N}_D - 1$,

$p = 2^{\left\lfloor \frac{n+1}{2} \right\rfloor}$, $q = 2^{\left\lfloor \frac{n}{2} \right\rfloor}$,

$$D^{p,q} = D^{p,q} \cup T^{p,q}_{m,n}\left[\hat{X}^{m,n}_{l}\right].$$

Passo 12: Retorna $\hat{X}^{m,n}_{l}$.

7.2 Resultados de simulações

As simulações apresentadas nesta seção foram obtidas através de uma implementação do SM-MMP em C, rodando em ambiente *Linux*.

As imagens comprimidas para teste foram as mesmas utilizadas no capítulo 5, ou seja, *LENA*, *BABOON*, *F-16*, *BRIDGE*, *AERIAL*, *BARBARA*, *GOLD*, *PP1209* e *PP1205*, todas de 512×512 *pixels*.

Os gráficos desta seção mostram os resultados para o SM-MMP e também para o SPIHT, o JPEG, o RD-MMP [1] e o RDI-MMP. Deste modo, uma comparação instantânea pode ser feita com um codificador baseado em DCT, outro em *Wavelets* e também com aqueles desenvolvidos anteriormente no capítulo 6 e em [1].

Todas as imagens foram inicialmente divididas em blocos de 16×16, sendo estes, então, processados em seqüência pelo algoritmo, no sentido de leitura, ou seja, da esquerda para a direita e de cima para baixo. Para cada nó n_l analisado na otimização ou na codificação, um dicionário $D_S^{m,n}$ foi gerado, com elementos escolhidos de acordo com o critério de continuidade *RG*.

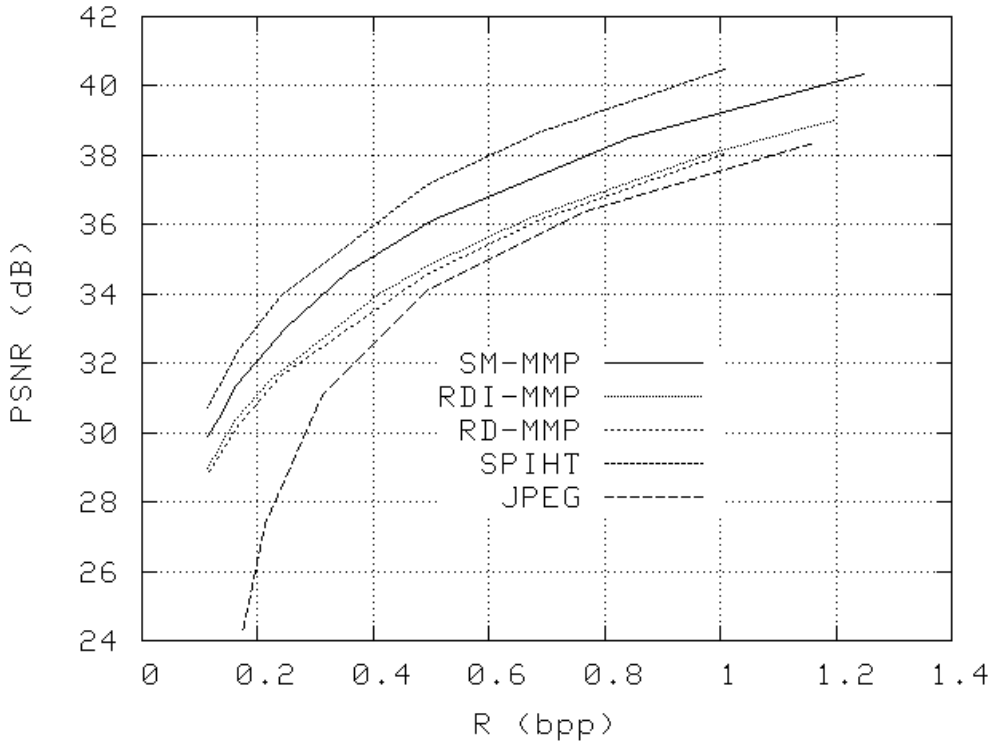


Figura 7.9: Taxa×distorção para *LENA* 512×512 comprimida com SM-MMP.

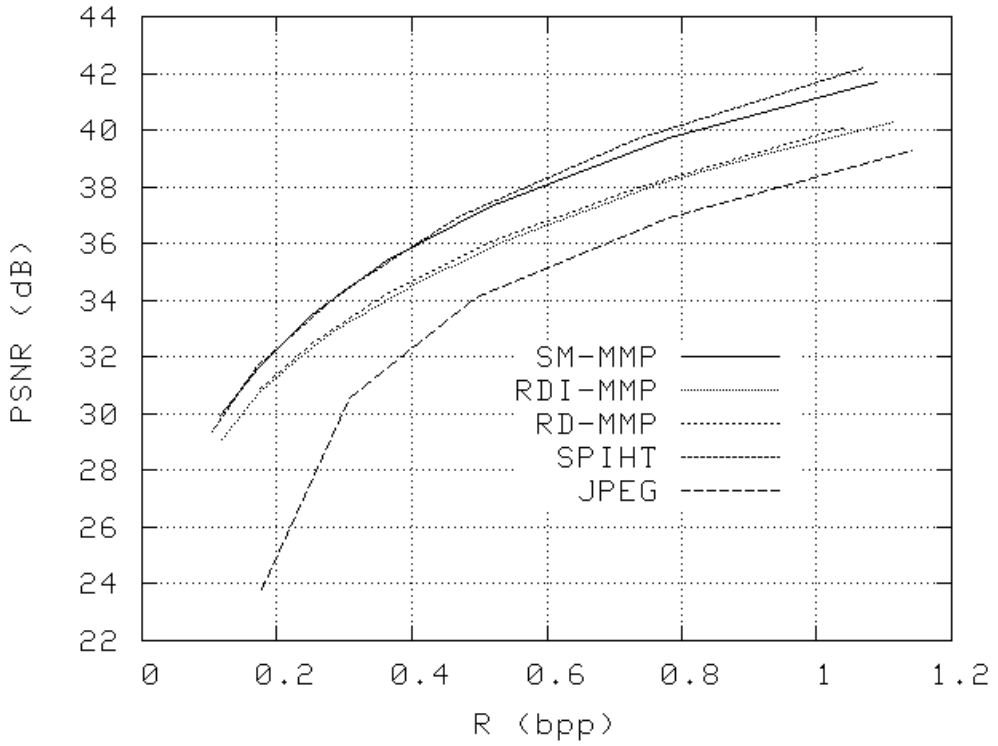


Figura 7.10: Taxa×distorção para *F-16* 512×512 comprimida com SM-MMP.

É visível que o desempenho do SM-MMP, em imagens suaves, é bem superior ao do RDI-MMP, apresentando ainda uma sensível redução no efeito de blocagem. Entretanto, em taxas menores, as imagens mais complexas, como *BARBARA* e *BABOON*, acabam apresentando *PSNRs* um pouco baixas. Esse fato pode ser explicado com o auxílio do algoritmo para o cálculo do dicionário de estado. Na imagem *BARBARA*, por exemplo, o dicionário de estado tem valor máximo de 60384 elementos, porém, o dicionário oficial só chega a valores entre de 15000 e 20000 elementos para taxas próximas de 0,5*bpp*. Logo, grande parte do problema consiste no lento crescimento apresentado pelo dicionário oficial D.

A solução para este problema reside na mudança de estratégia de atualização de dicionário, fazendo com que o mesmo cresça muito mais rapidamente. É claro que tal crescimento só surtirá efeito se ocorrer de maneira adaptativa, incluindo-se no dicionário oficial elementos com grande probabilidade de serem utilizados.

O efeito de blocagem foi extremamente reduzido, proporcionado um aumento significativo na qualidade subjetiva das imagens reconstruídas, o que pode ser percebido facilmente comparando-se as Figuras 6.12 e 7.18. As imagens *PP1205* e

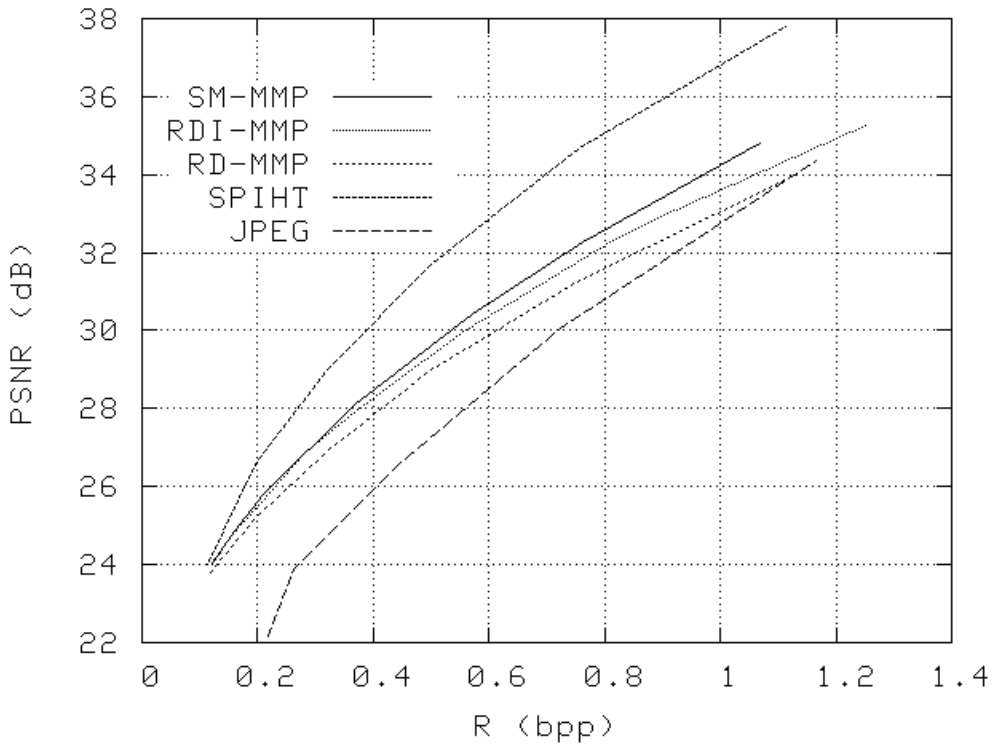


Figura 7.11: Taxa×distorção para *BARBARA* 512×512 comprimida com SM-MMP.

PP1209, comprimidas pelo SM-MMP a 0,5*bpp*, estão disponíveis para comparação nas Figuras 7.19 e 7.20, respectivamente.

Um fato interessante é o aumento de desempenho na codificação da imagem *PP1209*, que é claramente percebido no gráfico 7.16. Isto pode ser explicado devido à codificação mais eficaz da região das duas imagens *LENA*, proporcionada pelo casamento lateral realizado entre os blocos no SM-MMP.

O SPIHT ainda apresenta os melhores resultados, porém, o SM-MMP chega quase igualar-se ao mesmo na imagem *F-16*. O desempenho em todas as imagens suaves aumentou, diminuindo bastante a diferença com relação aos algoritmos baseados em *Wavelets*, representados aqui pelo SPIHT.

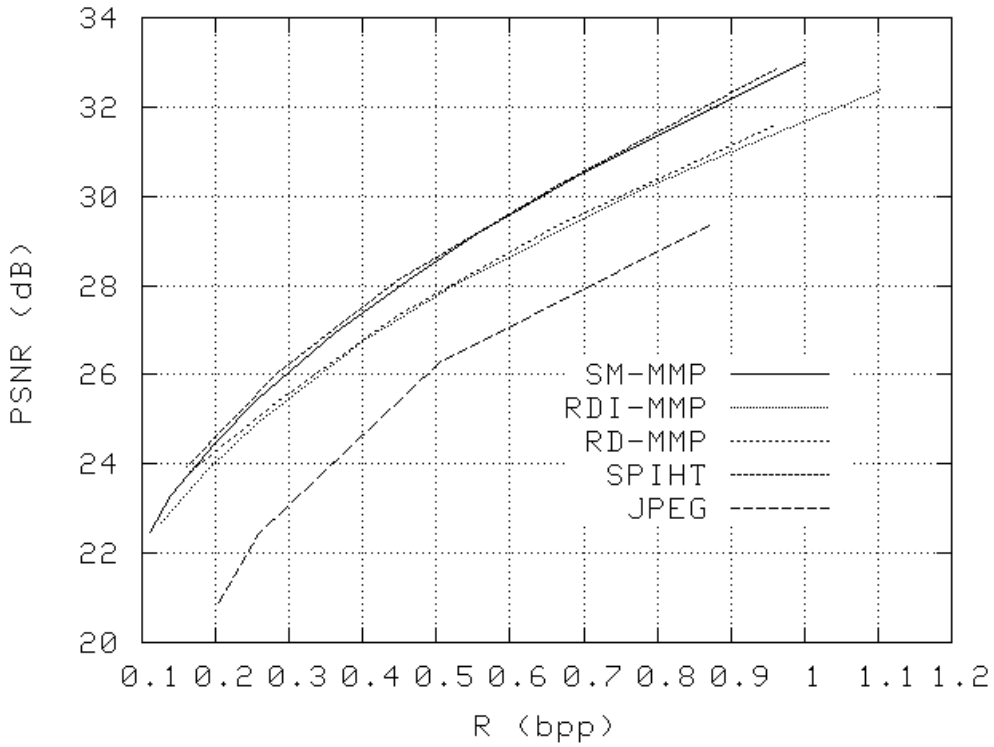


Figura 7.12: Taxa×distorção para *AERIAL* 512×512 comprimida com SM-MMP.

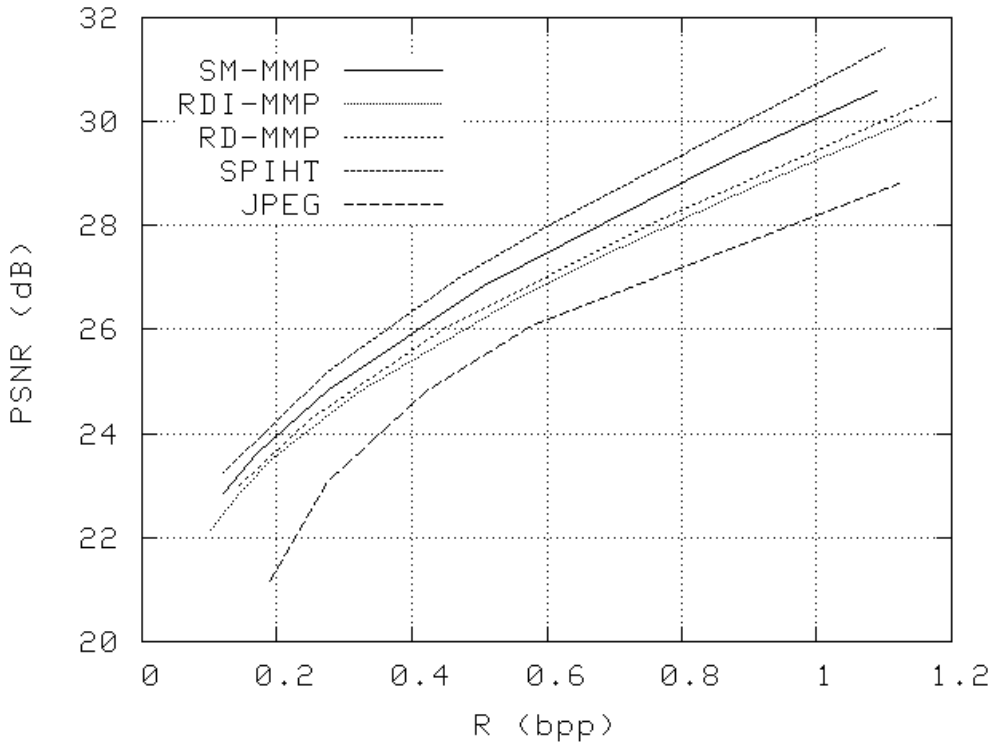


Figura 7.13: Taxa×distorção para *BRIDGE* 512×512 comprimida com SM-MMP.

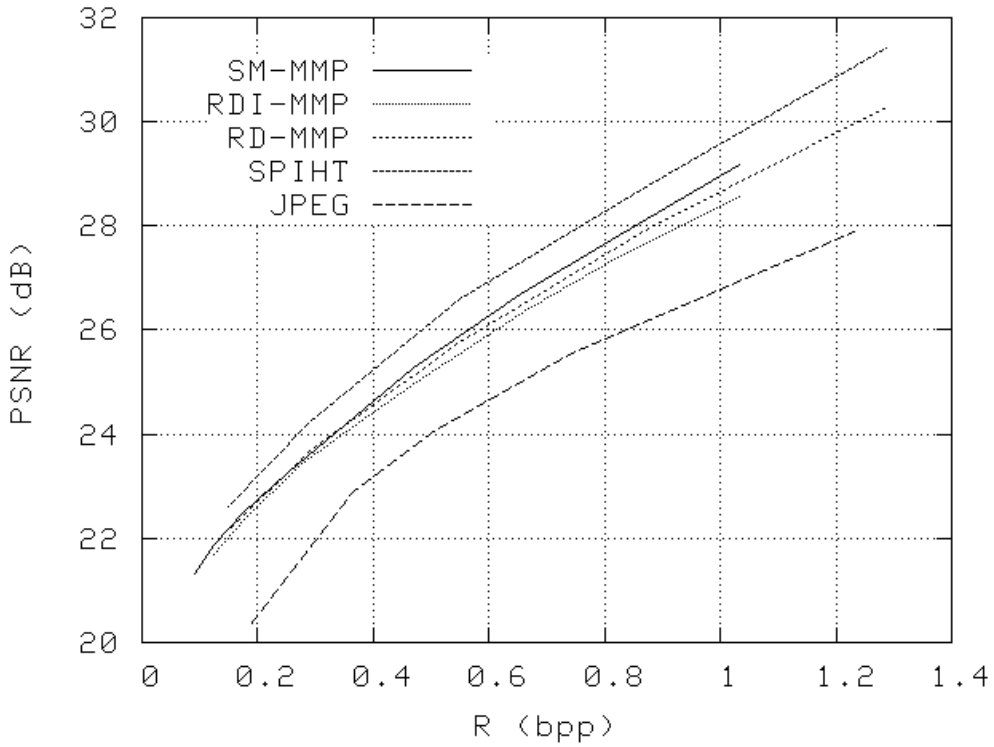


Figura 7.14: Taxa×distorção para *BABOON* 512×512 comprimida com SM-MMP.

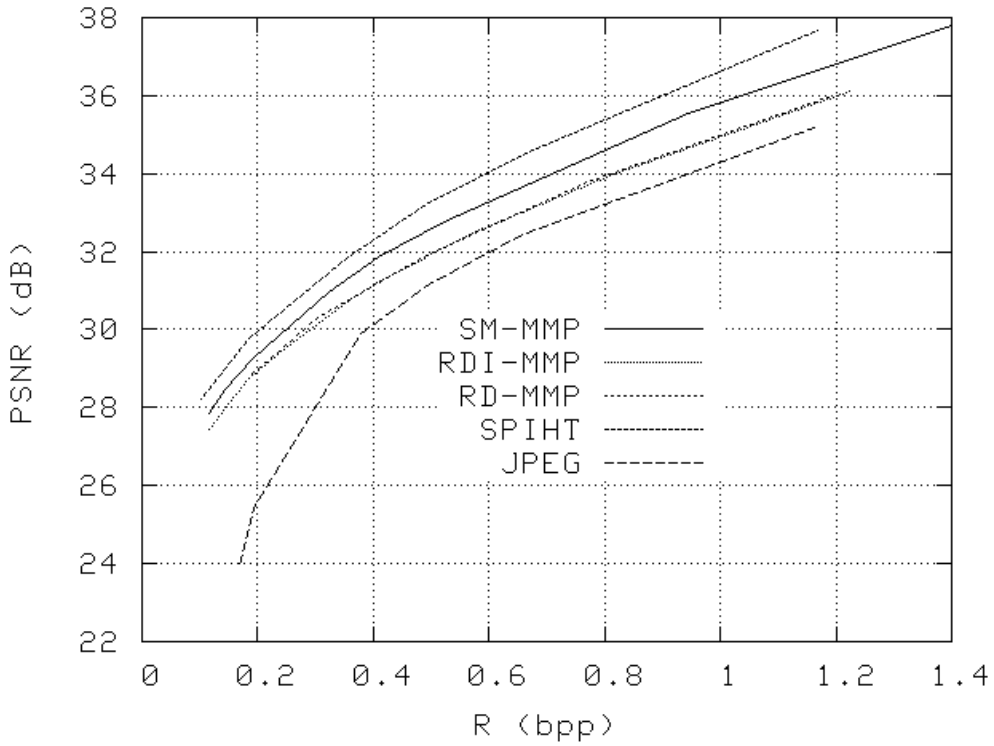


Figura 7.15: Taxa×distorção para *GOLD* 512×512 comprimida com SM-MMP.

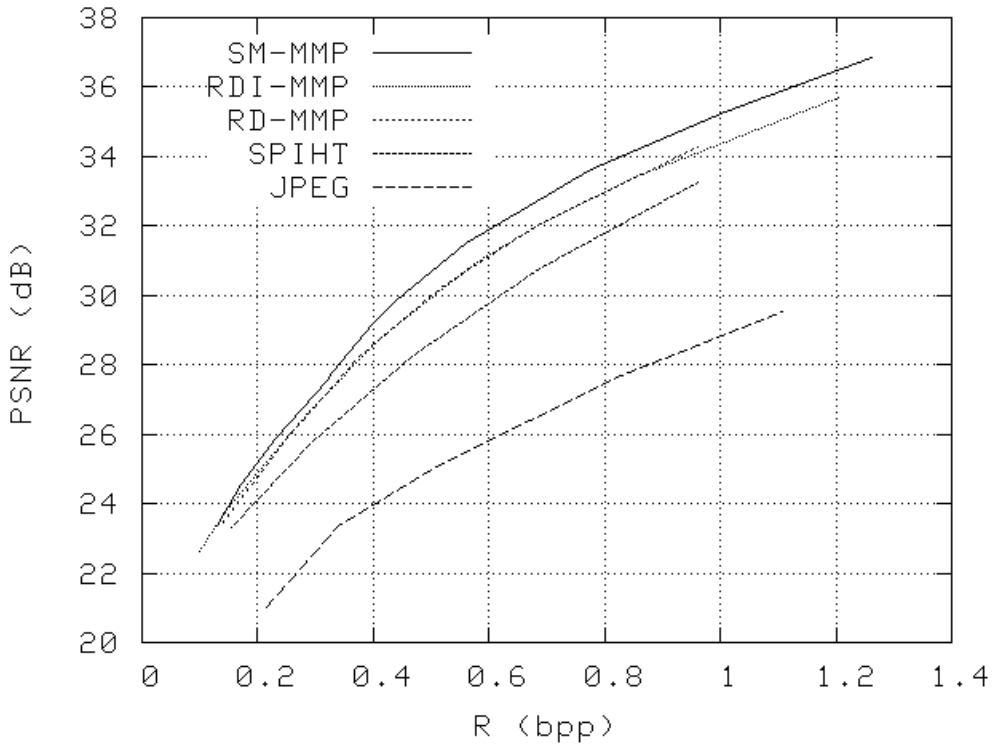


Figura 7.16: Taxa×distorção para *PP1209* 512×512 comprimida com SM-MMP.

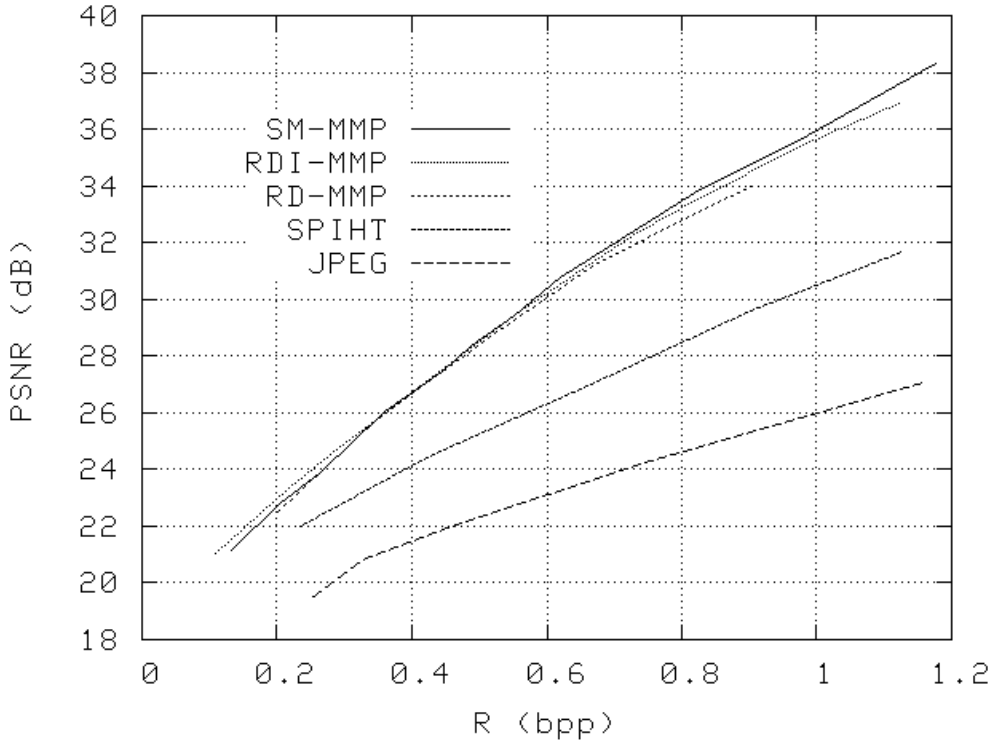
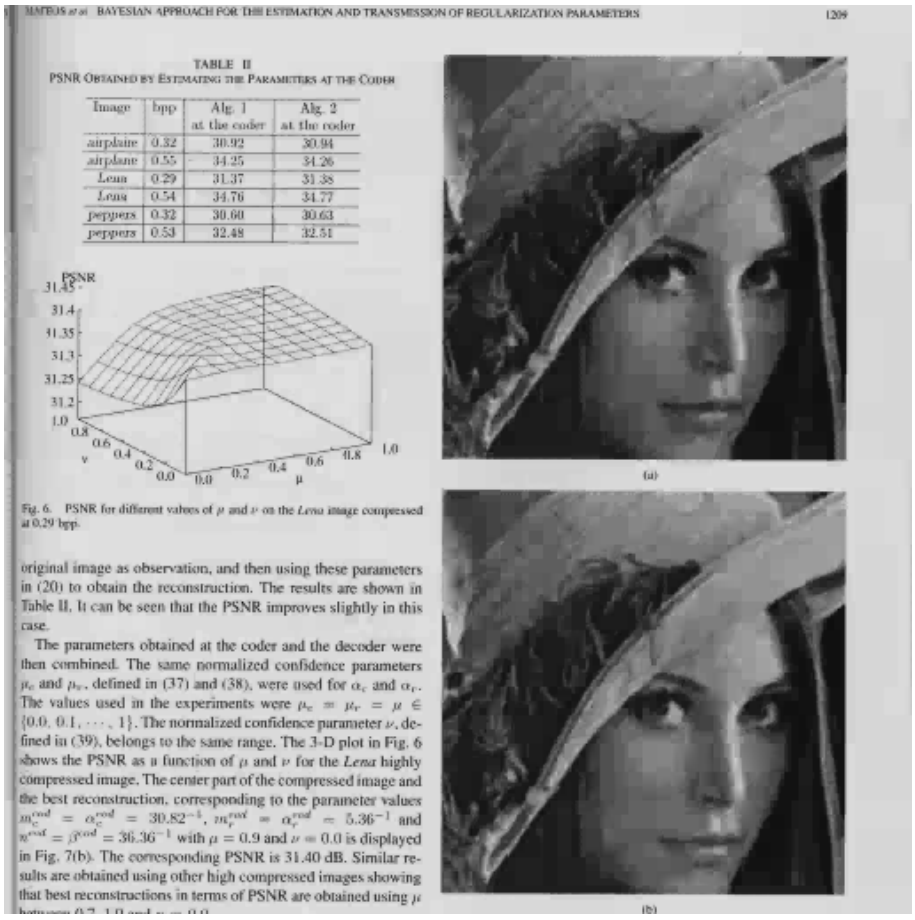


Figura 7.17: Taxa×distorção para *PP1205* 512×512 comprimida com SM-MMP.

Figura 7.18: *LENA* comprimida a 0,5*bpp* pelo SM-MMP. PSNR=36,13dB.

Figura 7.19: *PP1205* comprimida a 0,5*bpp* pelo SM-MMP. PSNR=28,54dB.



MATEOS *et al.*: BAYESIAN APPROACH FOR THE ESTIMATION AND TRANSMISSION OF REGULARIZATION PARAMETERS 1209

TABLE II
PSNR OBTAINED BY ESTIMATING THE PARAMETERS AT THE CODER

Image	bpp	Alg. 1 at the coder	Alg. 2 at the coder
airplane	0.32	30.92	30.94
airplane	0.55	34.25	34.26
Lena	0.29	31.37	31.38
Lena	0.54	34.76	34.77
peppers	0.32	30.60	30.63
peppers	0.53	32.48	32.51

Fig. 6. PSNR for different values of μ and ν on the *Lena* image compressed at 0.29 bpp.

original image as observation, and then using these parameters in (20) to obtain the reconstruction. The results are shown in Table II. It can be seen that the PSNR improves slightly in this case.

The parameters obtained at the coder and the decoder were then combined. The same normalized confidence parameters μ_c and μ_r, defined in (37) and (38), were used for α_c and α_r. The values used in the experiments were $\mu_c = \mu_r = \mu \in \{0.0, 0.1, \cdots, 1\}$. The normalized confidence parameter ν, defined in (39), belongs to the same range. The 3-D plot in Fig. 6 shows the PSNR as a function of μ and ν for the *Lena* highly compressed image. The center part of the compressed image and the best reconstruction, corresponding to the parameter values $m_c^{cod} = \alpha_c^{cod} = 30.82^{-1}$, $m_r^{cod} = \alpha_r^{cod} = 5.36^{-1}$ and $n^{cod} = \beta^{cod} = 36.36^{-1}$ with $\mu = 0.9$ and $\nu = 0.0$ is displayed in Fig. 7(b). The corresponding PSNR is 31.40 dB. Similar results are obtained using other high compressed images showing that best reconstructions in terms of PSNR are obtained using μ

(a)

(b)

Figura 7.20: *PP1209* comprimida a 0,5*bpp* pelo SM-MMP. PSNR=30,69dB.

Capítulo 8

A Super-atualização de dicionário

Um ponto fraco do algoritmo SM-MMP apresentado, que pode ocasionar uma redução significativa na $PSNR$ das imagens reconstruídas, reside no fato de que, apesar da técnica de construção de dicionário ser adaptativa, tudo começa com um dicionário bastante pequeno (dicionário inicial D_o), ocasionando uma codificação pobre para os primeiros blocos $X_0^{m,n}$. Além disso, a adaptabilidade é bastante lenta, pois cada nó n_l acima de um nó-folha leva à inclusão de apenas um elemento $S_i^{m,n}$ novo em cada escala, sendo este uma versão transformada da concatenação dos blocos $\hat{X}_{2l+1}^{m,n}$ e $\hat{X}_{2l+2}^{m,n}$ já codificados, não havendo qualquer modificação ou enriquecimento do elemento para facilitar a codificação de outros tipos de estruturas presentes na imagem.

Além disso, como foi demonstrado no capítulo anterior, o dicionário oficial atinge um número de elementos abaixo do que seria necessário para a codificação de alguns tipos de imagens, como *BARBARA* e *BABOON*, o que resulta em imagens reconstruídas com qualidade abaixo do que seria possível.

Este capítulo apresenta uma nova abordagem para a atualização de dicionário no SM-MMP, que proporciona uma solução para os problemas apresentados e resulta num dicionário oficial D muito mais adaptado e diversificado para a imagem em codificação.

8.1 Descrição e implementação do algoritmo

A primeira solução para o problema apresentado seria provocar um crescimento mais agressivo do dicionário D, proporcionando maior diversidade para a codificação dos blocos de entrada $X_l^{m,n}$. No algoritmo MMP padrão, esta estratégia pode significar um aumento da taxa $\mathcal{R}$ dos índices, pois a freqüência acumulada dos mesmos também aumenta e, com isso, o número de bits necessário para a sua representação. Entretanto, no caso do SM-MMP, o tamanho do dicionário oficial não importa, pois sempre serão escolhidos os $l\left(D_S^{m,n}\right)$ elementos com menores rugosidades para preencherem $D_S^{m,n}$. Deste modo, no algoritmo SM-MMP, o dicionário oficial $D^{m,n}$ pode ter qualquer tamanho, sem resultar diretamente em aumento de taxa dos índices.

Por outro lado, o aumento descontrolado e sem regras do dicionário oficial D pode levar a um esforço computacional inútil, incluindo elementos $S_i^{m,n}$ com probabilidade de utilização extremamente baixa. A solução encontrada para isto é a inclusão de elementos $S_i^{m,n}$ que sejam combinações ou deslocamentos dos blocos $\hat{X}_i^{m,n}$ que já foram codificados, ainda aumentando o tamanho do dicionário de uma maneira adaptativa e incluindo elementos com probabilidade razoável de serem utilizados. Esta técnica se justifica também pelo fato de que, numa mesma imagem, pode haver estruturas semelhantes em diferentes posições (deslocadas). Um bom exemplo desse fato seria uma imagem com uma barra inclinada, como mostrado na Figura 8.1. Num determinado bloco, a borda da barra está um pouco deslocada em relação à do seu antecessor, e a inclusão de elementos deslocados para cima, no bloco antecessor, poderia facilitar em muito sua codificação.

No algoritmo SM-MMP, mostrado na seção anterior, cada atualização de dicionário leva à inclusão de 1 elemento novo em cada escala; na nova abordagem proposta, cada atualização pode levar, agora, à inclusão de 10 elementos novos em cada escala, a partir de deslocamentos na vertical, horizontal e diagonal, na direção dos blocos $\hat{X}_i^{m,n}$ já codificados (deslocamentos em outras direções seriam incompletos ou necessitariam de repetição de *pixels*, ou seja, *stuffing*). Todos os deslocamentos são de $\frac{1}{4}$ do número de linhas, de colunas ou dos dois (diagonal). Tal técnica, mostrada na Figura 8.2, leva a aumentos consideráveis na $PSNR$ e na qualidade subjetiva das imagens codificadas.

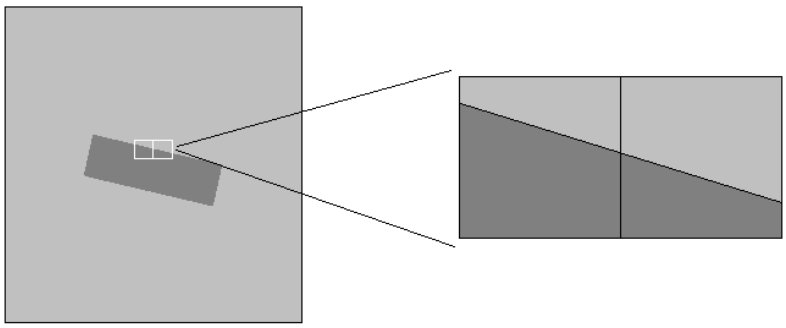


Figura 8.1: Demonstração da necessidade de inclusão de elementos deslocados.

Na Figura 8.2, são mostradas as atualizações deslocadas de $\frac{1}{2}m$ e $\frac{1}{2}n$ em relação ao bloco de maior hierarquia, que no presente trabalho é de 16×16. O bloco tracejado vermelho mostra um deslocamento para cima de $\frac{1}{2}m$, o amarelo para a esquerda de $\frac{1}{2}n$ e o azul para cima e para a esquerda de $\frac{1}{2}m$ e $\frac{1}{2}n$, respectivamente.

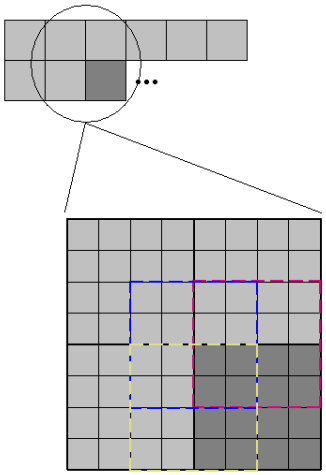


Figura 8.2: Atualizações com deslocamentos de $\frac{1}{2}m$ e $\frac{1}{2}n$ no bloco $X_0^{m,n}$.

O número total de inclusões adicionais pode chegar a 9, desde que as dimensões e a posição do bloco, na imagem original, permitam os devidos deslocamentos. Por exemplo, os elementos da primeira linha ou coluna de blocos $X_0^{m,n}$ ocasionam um número menor de inclusões, e os blocos com dimensões 2×1 e 2×2 não permitem deslocamentos de $\frac{1}{4}$ de qualquer das dimensões (apenas os de $\frac{1}{2}$).

Os outros deslocamentos possíveis, não mostrados na Figura 8.2, são de $\left\{\frac{1}{4}m; \frac{1}{4}n; \frac{1}{4}m, \frac{1}{4}n\right\}$ e $\left\{\frac{3}{4}m; \frac{3}{4}n; \frac{3}{4}m, \frac{3}{4}n\right\}$. É importante observar que a super-atualização

pode ocorrer em blocos de quaisquer dimensões e mesmo nos não quadrados. Neste ponto, é válido ressaltar que os deslocamentos podem ser realizados dentro de $\mathcal{B}^{4m,4n}$ com grande facilidade, sendo esta uma outra vantagem que ainda não tinha sido mencionada, ratificando a importância da sua implementação no algoritmo SM-MMP. Um exemplo de atualização em bloco não quadrado é mostrado na Figura 8.3, onde se observa que não seriam possíveis deslocamentos de $\frac{1}{4}$ e $\frac{3}{4}$ para a esquerda ou na diagonal, devido às dimensões do bloco.

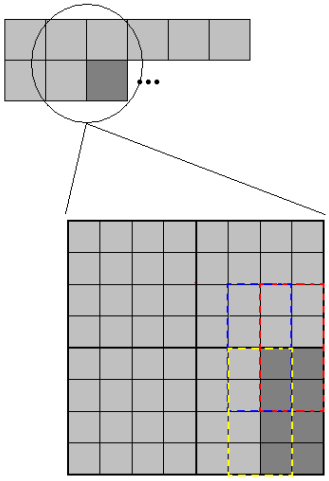


Figura 8.3: Atualizações com deslocamentos de $\frac{1}{2}m$ e $\frac{1}{2}n$ no bloco $X_l^{m,n}$ resultante de divisão.

É interessante ressaltar, ainda, que a implementação da técnica de superatualização do dicionário oficial é relativamente simples e tem como características principais:

- Crescimento mais acelerado do dicionário, melhorando a codificação para os primeiros blocos $X_0^{m,n}$;
- Inclusão de elementos $S_i^{m,n}$ com razoável probabilidade de serem utilizados;
- Possibilidade de uma codificação de menor custo $\mathcal{J}(n_l)$ para blocos que são deslocamentos dos $\hat{X}_l^{m,n}$ que já foram codificados;
- Maior adaptabilidade do dicionário à imagem que está sendo codificada.

Deve-se observar, porém, que os maiores ganhos são esperados em imagens de complexidade média , pois imagens mais suaves já obtêm bons resultados mesmo com dicionários menores, devido principalmente a pouca diversidade de estruturas.

Apesar de todas as vantagens apresentadas, este aumento mais agressivo do dicionário tem um custo computacional elevado, tanto na inclusão de novos elementos quanto na procura dos de menores RGs para inclusão em $D_S^{m,n}$.

8.2 Resultados de simulações

As simulações apresentadas nesta seção foram obtidas através de uma implementação do SM-MMP com super-atualização de dicionário (SM-MMP(SA)) em C, rodando em ambiente *Linux*.

As imagens comprimidas para teste foram as mesmas utilizadas no capítulo 5, ou seja, *LENA*, *BABOON*, *F-16*, *BRIDGE*, *AERIAL*, *BARBARA*, *GOLD*, *PP1209* e *PP1205*, todas de 512×512 *pixels*.

Os gráficos desta seção mostram os resultados para o SM-MMP(SA) e também para o SM-MMP, o SPIHT, o JPEG, o RD-MMP [1] e o RDI-MMP. Deste modo, uma comparação instantânea pode ser feita com um codificador baseado em DCT, outro em *Wavelets* e também com aqueles desenvolvidos anteriormente nos capítulos 6, 7 e em [1].

Todas as imagens foram inicialmente divididas em blocos de 16×16, sendo estes, então, processados em seqüência pelo algoritmo, no sentido de leitura, ou seja, da esquerda para a direita e de cima para baixo. Para cada nó n_l analisado na otimização ou na codificação, um dicionário $D_S^{m,n}$ foi gerado, com elementos escolhidos de acordo com o critério de continuidade RG. No momento da atualização de dicionário, foram incluídos até 10 novos elementos em cada escala, de acordo com a posição e a dimensão de cada bloco. As inclusões adicionais são $\left\{\frac{1}{4}m; \frac{1}{4}n; \frac{1}{4}m, \frac{1}{4}n\right\}$, $\left\{\frac{1}{2}m; \frac{1}{2}n; \frac{1}{2}m, \frac{1}{2}n\right\}$ e $\left\{\frac{3}{4}m; \frac{3}{4}n; \frac{3}{4}m, \frac{3}{4}n\right\}$.

Como pode ser visto nos gráficos, houve melhora em todas as imagens, até mesmo nas mais suaves. Os resultados obtidos, para algumas imagens, foram iguais ao do SPIHT (e.g. *AERIAL* e *F-16*), mantendo-se diferenças menores que 1 dB nas demais.

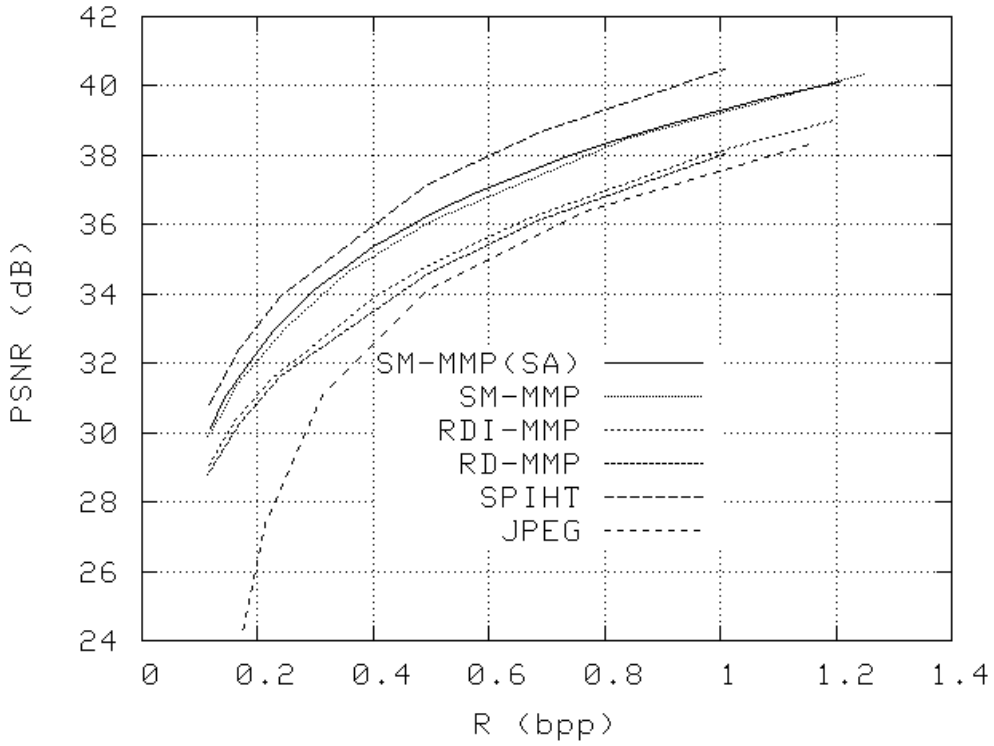


Figura 8.4: Taxa×distorção para *LENA* 512×512 comprimida com SM-MMP.

Ao se analisar os gráficos, percebe-se que a melhora proporcionada pela super-atualização de dicionário foi agressiva para algumas imagens, fazendo com que o rendimento aumentasse até mesmo em imagens que contêm texto.

Com relação ao efeito de blocagem, o mesmo tornou-se, em imagens comprimidas a taxas próximas de 0,5*bpp*, bem suave, aumentando mais ainda a qualidade subjetiva das imagens codificadas. Este fato pode ser comprovado analisando-se as Figuras 8.13 e 7.18. A imagem *AERIAL*, comprimida pelo SM-MMP(SA) e pelo SPIHT a 0,5*bpp*, está disponível para comparação nas Figuras 8.14 e 8.15, respectivamente.

Os resultados apresentados levam à necessidade de implementação do algoritmo de super-atualização em qualquer codificador baseado no SM-MMP, proporcionando maior diversidade para a codificação dos elementos e levando o tamanho do dicionário oficial a patamares adequados a um grande espectro de imagens, desde as mais suaves, como *LENA*, até as mais complexas, ressaltando-se a *PP1205* e a *PP1209*.

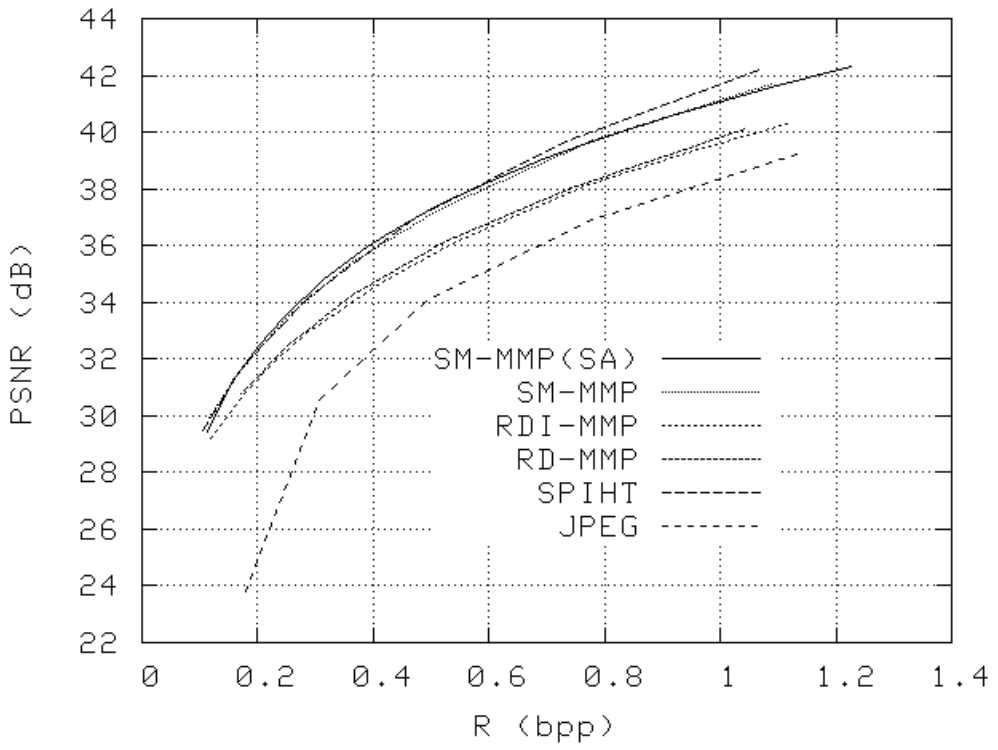


Figura 8.5: Taxa×distorção para *F-16* 512×512 comprimida com SM-MMP.

8.3 O efeito de blocagem

O efeito de blocagem é um problema existente no algoritmo MMP padrão e ocorre devido ao processamento independente sofrido pelos blocos de imagem. Como principal conseqüência, acarreta uma redução significativa na qualidade subjetiva da imagem reconstruída, não refletindo os bons resultados alcançados pelo algoritmo. Sua solução, nos antecessores do SM-MMP, estava basicamente restrita a uma filtragem na reconstrução da imagem, o que muitas vezes acabava reduzindo a $PSNR$ da mesma.

Vale a pena ressaltar que o efeito de blocagem é extremamente reduzido com a utilização do algoritmo SM-MMP, devido a escolha de elementos com *pixels* de borda similares (ou coerentes) aos dos seus vizinhos. Este fato é melhor percebido em taxas mais baixas, como exemplificado nas Figuras 8.16 e 8.17. Com isso, o rendimento desta classe de algoritmos se aproxima bastante do apresentado por algoritmos baseados em *Wavelets* (e.g. SPIHT), seja com relação à $PSNR$ ou à qualidade subjeitva das imagens reconstruídas.

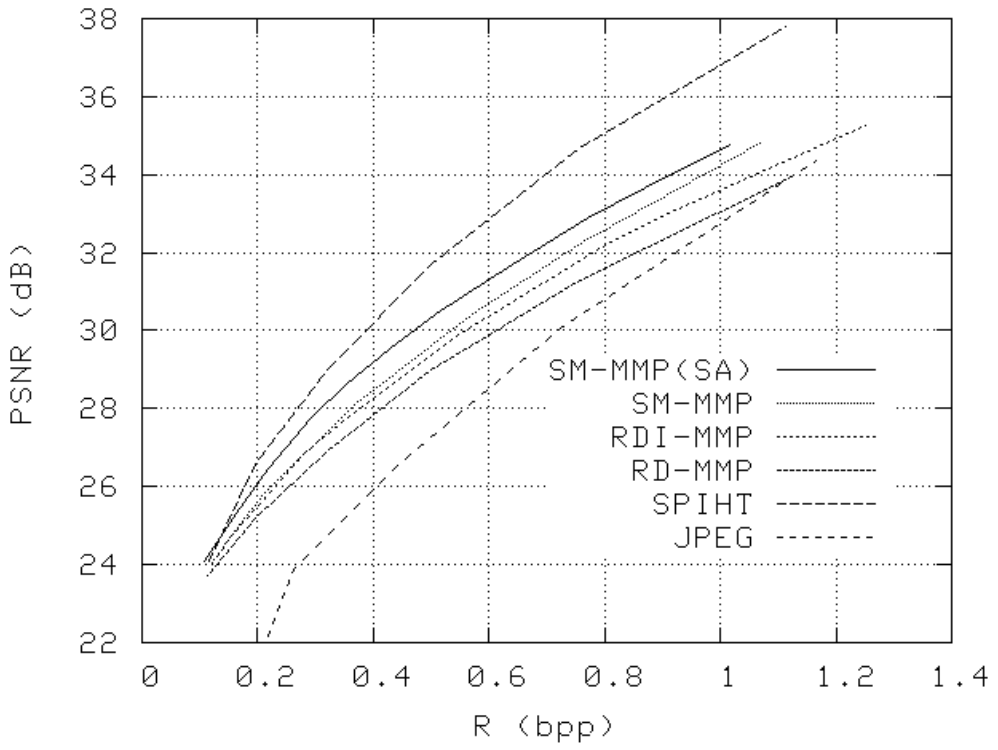


Figura 8.6: Taxa×distorção para *BARBARA* 512×512 comprimida com SM-MMP.

Novas formas de redução do efeito de blocagem podem ser estudadas e incorporadas ao algoritmo, ou o critério de continuidade utilizado pode ser aperfeiçoado, o que melhoraria ainda mais os resultados apresentados, praticamente anulando o efeito de blocagem.

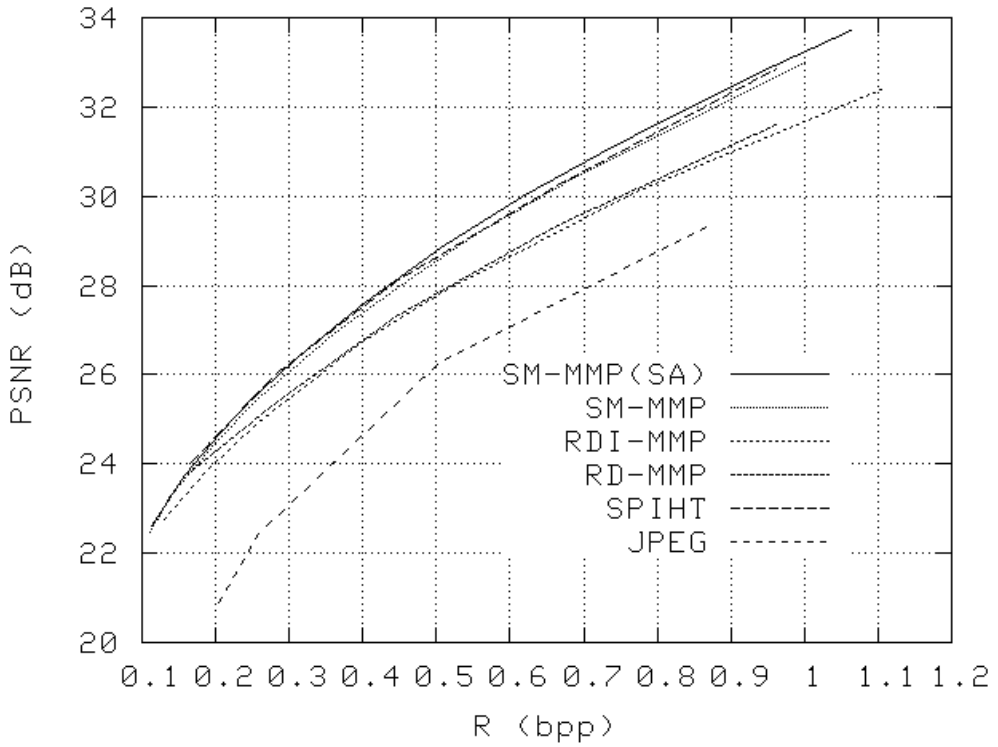


Figura 8.7: Taxa×distorção para *AERIAL* 512×512 comprimida com SM-MMP.

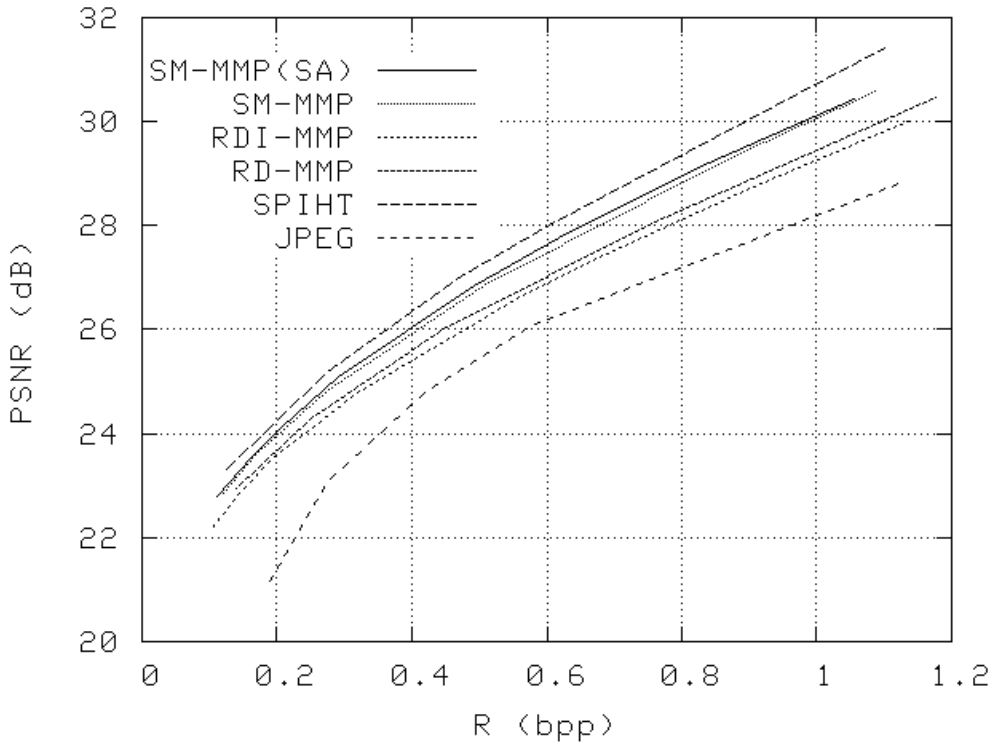


Figura 8.8: Taxa×distorção para *BRIDGE* 512×512 comprimida com SM-MMP.

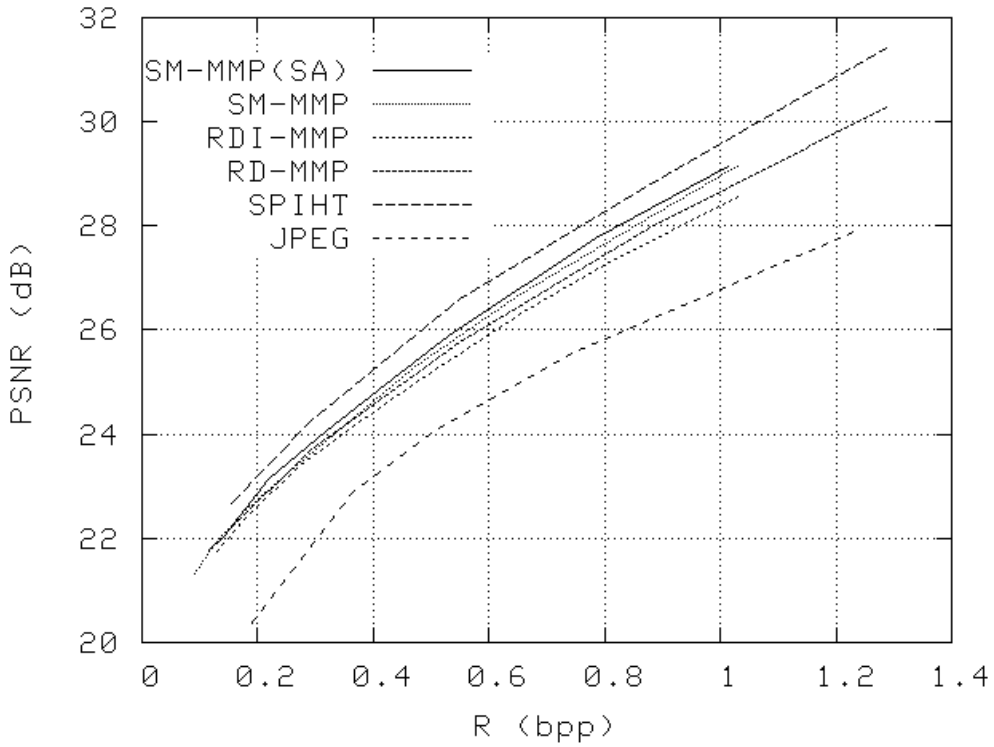


Figura 8.9: Taxa×distorção para *BABOON* 512×512 comprimida com SM-MMP.

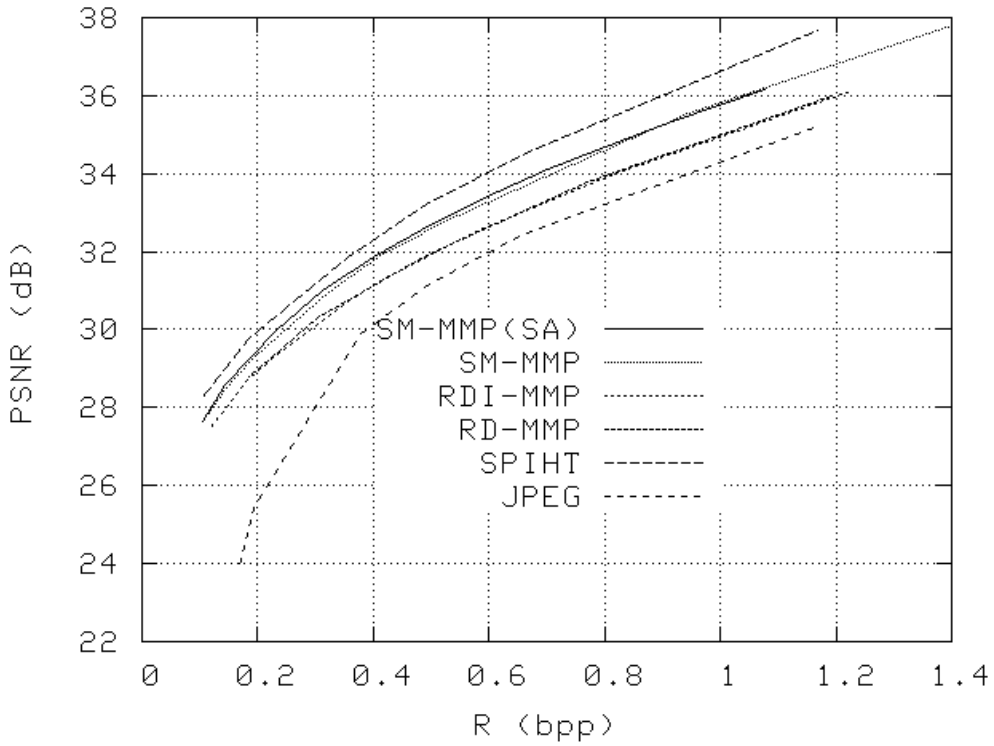


Figura 8.10: Taxa×distorção para *GOLD* 512×512 comprimida com SM-MMP.

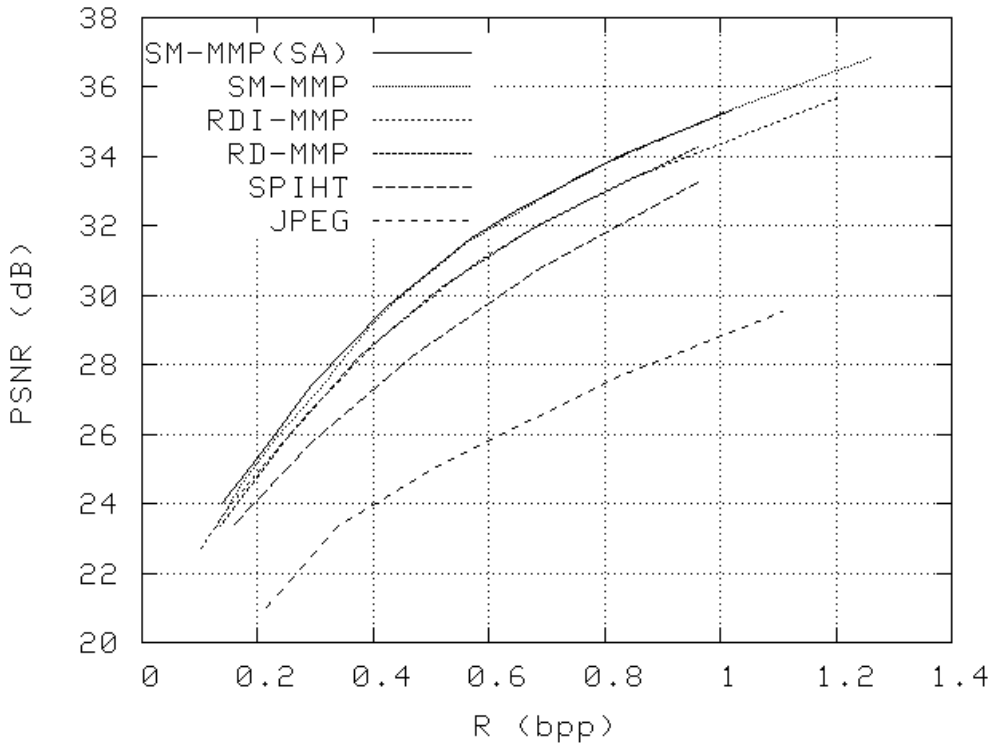


Figura 8.11: Taxa×distorção para *PP1209* 512×512 comprimida com SM-MMP.

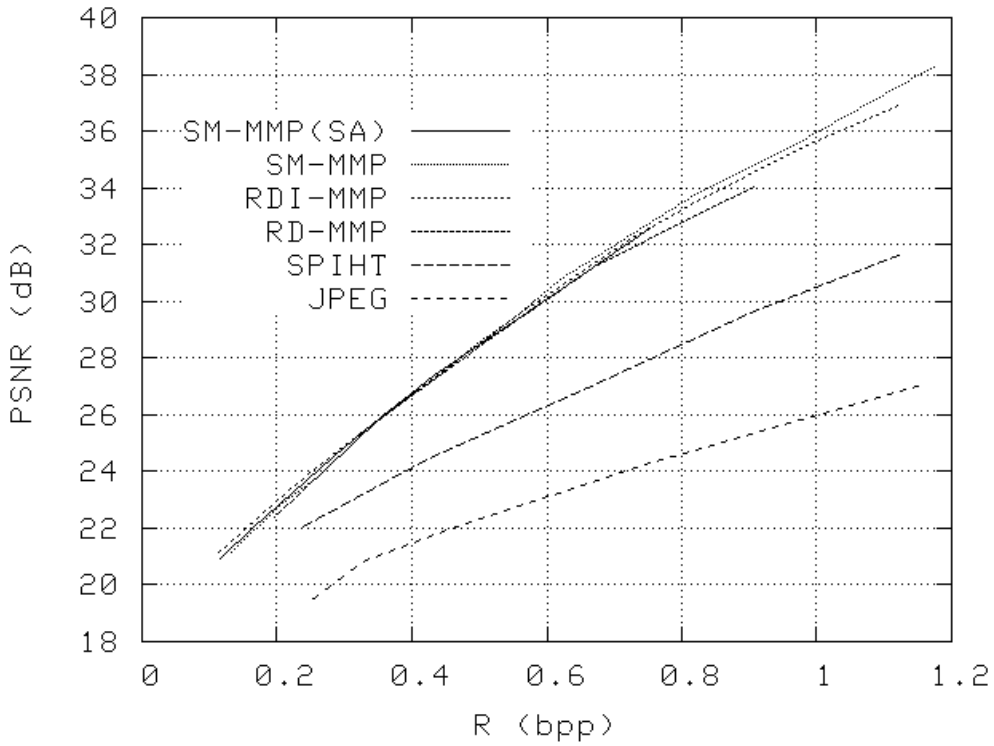


Figura 8.12: Taxa×distorção para *PP1205* 512×512 comprimida com SM-MMP.

Figura 8.13: *LENA* comprimida a 0,5*bpp* pelo SM-MMP(SA). PSNR=36,32dB.

Figura 8.14: *AERIAL* comprimida a 0,5*bpp* pelo SM-MMP(SA). PSNR=28,85dB.

Figura 8.15: *AERIAL* comprimida a 0,5*bpp* pelo SPIHT. PSNR=28,74dB.

Figura 8.16: *LENA* comprimida a 0,3*bpp* pelo RDI-MMP. PSNR=32,71dB.

Figura 8.17: *LENA* comprimida a 0,3*bpp* pelo SM-MMP(SA). PSNR=34,14dB.

Capítulo 9

Considerações finais

Foi desenvolvido um novo algoritmo para a otimização da árvore de segmentação, muito mais rápido que o seu antecessor (o modificado, ou RD-MMP) e com performance bastante similar: o algoritmo de otimização intermediário, ou RDI-MMP. Além disso, o algoritmo é adequado à estrutura do *Side-match* e apresenta modificações na transformação de escala, na regra de formação do dicionário inicial e na otimização da árvore de segmentação.

O algoritmo final desenvolvido, chamado de SM-MMP(SA), mostrou-se, com relação às imagens suaves, bastante superior aos seus antecessores, o RD-MMP e o RDI-MMP, mantendo um desempenho igual ou melhor para o resto do espectro testado. A codificação das imagens mais suaves apresentou melhoras na $PSNR$ que chegaram a 1,6 dB (imagem *F-16*), além de sensível aumento da qualidade subjetiva em todas as imagens testadas.

Devido ao processamento realizado pelo algoritmo de *Side-match* implementado e à sua forma de escolher o bloco aproximado, houve grande redução no efeito de blocagem. Isto se deve ao fato dos blocos serem escolhidos através da similaridade de seus *pixels* de borda com os de outros blocos já codificados, o que torna o processamento aplicado ao bloco atual dependente daqueles que já foram codificados. O efeito de blocagem é um problema bastante incômodo no MMP padrão e possui soluções através de pós-filtragem adaptativa, que não são, entretanto, inerentes ao algoritmo utilizado.

A escolha do tamanho do dicionário de estado, para cada bloco e em qualquer imagem, é automática e não ajustada heuristicamente de imagem para imagem,

como nos algoritmos de *Side-match* clássicos. Cada bloco tem seu tamanho de dicionário de estado adequado às suas características, calculado antes de se iniciar a codificação ou a otimização. A escolha do máximo tamanho do dicionário de estado também é automática e adequada às características de cada imagem, seja esta suave, mista ou de texto, através dos passos mostrados no capítulo 7, desenvolvidos a partir dos tamanhos de dicionário verificados para cada imagem.

Como resultado de tudo que foi implementado, obteve-se um compressor de imagens capaz de proporcionar resultados, em imagens suaves, próximos ou até mesmo levemente maiores que os obtidos com codificadores baseados em *Wavelets* e, em imagens mistas ou de texto, resultados significativamente melhores.

Os objetivos intencionados com a realização deste trabalho foram plenamente alcançados, pois se obteve um algoritmo com boa performance em imagens suaves, apresentando comportamento universal e sem necessidade de heurística individualizada na escolha de parâmetros, calculados automaticamente para cada imagem. Entretanto, melhoras podem ser realizadas, as quais têm o potencial de proporcionar resultados ainda mais significativos que os alcançados neste trabalho, tais como:

- Implementar classificadores de borda, organizando os dicionários oficial e de estado de acordo com o tipo de estrutura presente nos blocos, proporcionando uma busca mais rápida e focada em elementos mais adequados;

- Conceber novas estratégias para a atualização do dicionário oficial, incluindo vetores com deslocamentos diferentes ou rotacionados, procurando-se codificar com maior facilidade blocos possuindo outros tipos de estruturas;

- Projetar critérios de continuidade que utilizem mais blocos para o casamento lateral, possibilitando uma predição mais exata dos elementos adequados à codificação do bloco de entrada, como um *Three-sided* ou *Four-sided Side-match* [13];

- Realizar a análise de blocos maiores que os reais (e.g. dois pixels em excesso para cada dimensão), proporcionado uma superposição no decodificador, o que reduziria ainda mais o efeito de blocagem nas imagens (ver capítulo 7 de [1]);

- Os decodificadores apenas posicionam um dado vetor nas coordenadas indicadas pelos dados presentes no arquivo da imagem comprimida, não realizando

qualquer outra tarefa. Seria possível implementar-se alguma técnica que reduzisse ainda mais o efeito de blocagem através de uma estimativa auxiliar do vetor de entrada original ou técnica de suavização de bordas entre os blocos.

O algoritmo SM-MMP(SA) desenvolvido provou ser competitivo, apresentando resultados comparáveis aos dos codificadores baseados em *Wavelets* e ganhos significativos com relação ao seu antecessor, o RD-MMP, principalmente em imagens suaves. Esta superioridade foi obtida por meio da introdução de um modelo estatístico para a fonte, que atribui o valor zero para a probabilidade de ocorrência de blocos que têm seus pixels de borda muito diferentes dos de seus vizinhos superior e esquerdo.

Apesar da introdução de tal modelo estatístico, que privilegia imagens suaves, não houve queda de desempenho na codificação de imagens de gráficos ou texto. Este fato é devido à construção adaptativa do dicionário oficial e à super-atualização desenvolvida neste trabalho.

Por último, vale a pena ressaltar a importância do desenvolvimento desta classe de algoritmos de compressão a qual pertence o SM-MMP(SA), principalmente devido ao seu comportamento universal e ao potencial que apresenta para desenvolvimentos futuros.

Referências Bibliográficas

[1] M. B. de Carvalho, "Compressão de Sinais Multidimensionais usando Recorrência de Padrões Multiescalas", tese de doutorado, Departamento de Engenharia Elétrica, UFRJ, Março de 2001.

[2] M. B. de Carvalho, E. A. B. da Silva and W. A. Finamore, "Multidimensional Signal Compression using Multiscale Recurrent Patterns", Elsevier Signal Processing 82, pp. 1559-1580, 2002.

[3] A. Papoulis, "Probability, Random Variables and Stochastic Processes", McGraw-Hill Book Company, 1991.

[4] T. M. Cover and J. A. Thomas, "Elements of Information Theory", John Wiley & Sons Inc., 1991.

[5] K. Sayood, "Introduction to Data Compression", Morgan Kaufmann Publishers, 2000.

[6] R. C. Gonzalez and R. E. Woods, "Digital Image Processing", Addison-Wesley Publishing Company Inc, 1992.

[7] C. E. Shannon, "A Mathematical Theory of Communication", Bell Syst. Tech. Journal, Vol. 27, pp. 379-423, 1948.

[8] C. E. Shannon, "Coding Theorems for a Discrete Source with a Fidelity Criterion", in IRE National Convention Record, Part 4, pp. 142-163, 1959.

[9] R. E. Blahut, "Computation of Channel Capacity and Rate Distortion Functions", IEEE Transactions on Information Theory, Vol. 18, pp. 460-473, July 1972.

[10] S. Arimoto, "An Algorithm for Computing The Capacity of Arbitrary Discrete Memoryless Channels", IEEE Transactions on Information Theory, Vol. 18, pp. 14-20, January 1972.

[11] T. Kim, "Side Match and Overlap Match Vector Quantizers for Images", IEEE Transactions on Image Processing, Vol. 1, No. 2, pp. 170-185, February 1992.

[12] S. B. Yang and L. Y. Tseng, "Smooth Side-Match Classified Vector Quantizer with Variable Block Size", IEEE Transactions on Image Processing, Vol. 10, No. 5, pp. 677-685, May 2001.

[13] H. C. Wei, P. C. Tsai and J. S. Wang, "Three-Sided Side Match Finite-State Vector Quantization", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 10, No. 1, pp. 51-58, February 2000.

[14] T. S. Chen and C. C. Chang, "A New Image Coding Algorithm Using Variable-Rate Side-Match Finite-State Vector Quantization", IEEE Transactions on Image Processing, Vol. 6, No. 8, pp. 1185-1187, August 1997.

[15] S. B. Yang, "General-Tree-Structured Vector Quantizer for Image Progressive Coding Using Smooth Side-Match Method", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 13, No. 2, pp. 193-202, February 2003.

[16] Z. M. Lu, B. Yang and S. H. Sun, "Image Compression Algorithms Based on Side-Match Vector Quantizer with Gradient-Based Classifiers", IEICE Transactions on Information and Systems, Vol. E85-D, No. 9, pp. 1409-1415, September 2002.

[17] A. Said and W. A. Pearlman, "A New, Fast and Efficient Image Codec Based on Set Partitioning in Hierarchical Trees", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 6, pp. 243-250, June 1996.

[18] C. Christopoulos, A. Skodras and T. Ebrahimi, "The JPEG2000 Still Image Coding System: An Overview", IEEE Transactions on Consumer Electronics, Vol. 46, No. 2, pp. 1103-1127, November 2000.

[19] M. M. Denn, "Optimization by Variational Methods", McGraw-Hill Book Company, 1969.

[20] W. B. Pennebaker and J. L. Mitchell, “JPEG Still Image Data Compression Standard”, Van Nostrand Reinhold, 1994.

[21] J. L. Mitchell, W. B. Pennebaker, C. E. Fogg and D. J. LeGall, “MPEG Video Compression Standard”, Kluwer Academic Publishers, 2001.

[22] I. H. Witten, R. M. Neal and J. G. Cleary, “Arithmetic Coding for Data Compression”, Communications of the ACM, Vol. 30, No. 6, pp. 520-540, June 1987.

Apêndice A

Imagens Originais

Neste apêndice, são mostradas as imagens de teste originais *LENA*, *BABOON*, *F-16*, *BRIDGE*, *AERIAL*, *BARBARA*, *GOLD*, *PP1209* e *PP1205*, todas de 512×512 *pixels*. As sete primeiras imagens, mostradas de A.1 a A.7, foram obtidas no *site* http://sipi.usc.edu/services/database/Database.html, e as duas últimas, mostradas de A.8 a A.9, foram digitalizadas do IEEE *Transactions on Image Processing*, volume 9, número 7, de julho de 2000. As páginas escolhidas foram as de número 1209 e 1205, que dão nome às imagens

Figura A.1: *LENA* Original, 512×512, 8*bpp*.

Figura A.2: *F-16* Original, 512×512, 8*bpp*.

Figura A.3: *BARBARA* Original, 512×512, 8*bpp*.

Figura A.4: *AERIAL* Original, 512×512, 8*bpp*.

Figura A.5: *BRIDGE* Original, 512×512, 8*bpp*.

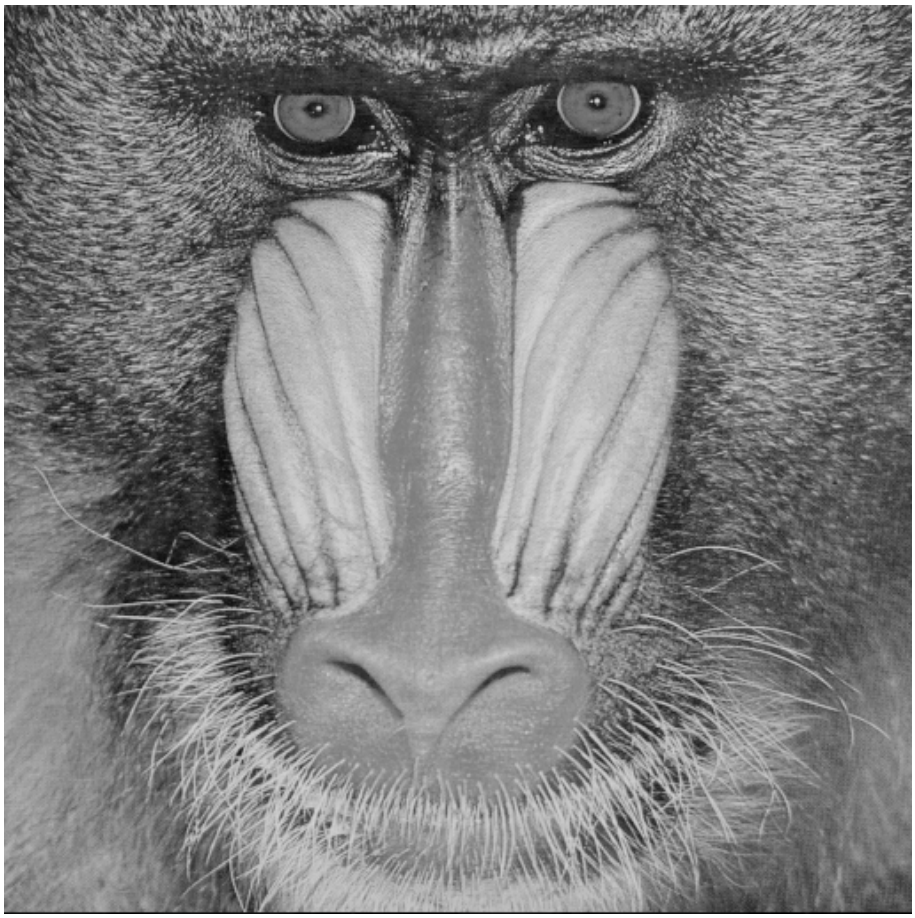
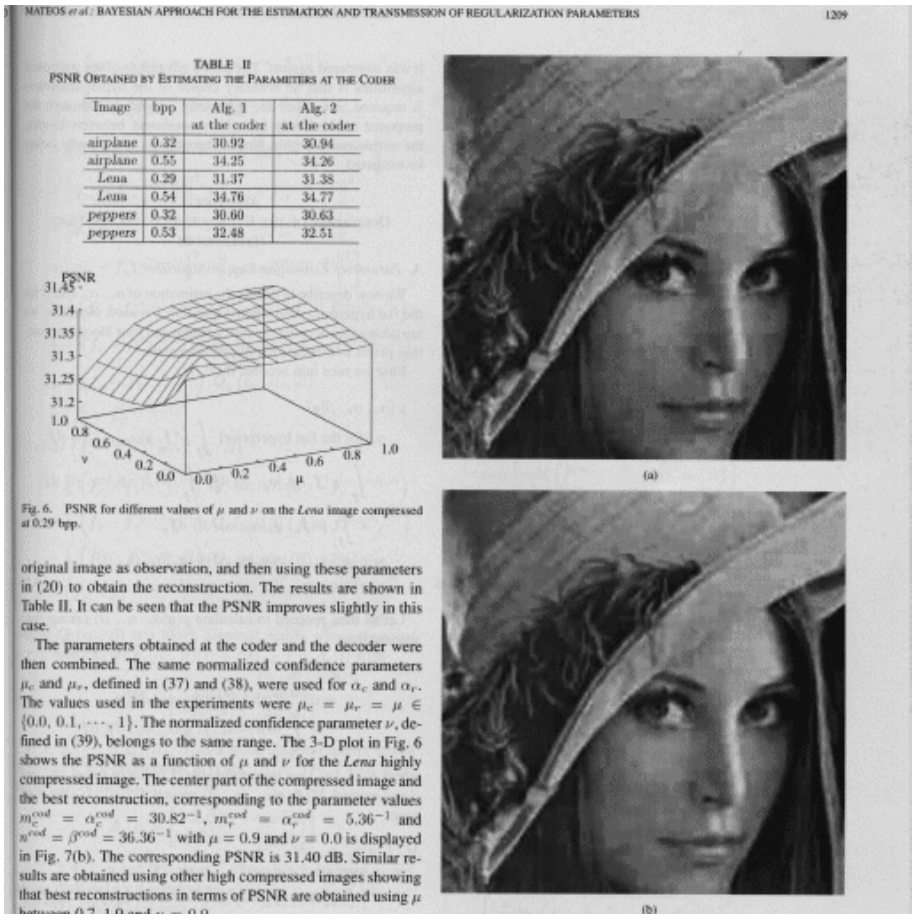


Figura A.6: *BABOON* Original, 512×512, 8*bpp*.

Figura A.7: *GOLD* Original, 512×512, 8*bpp*.



MATEOS *et al.*: BAYESIAN APPROACH FOR THE ESTIMATION AND TRANSMISSION OF REGULARIZATION PARAMETERS 1209

TABLE II
PSNR OBTAINED BY ESTIMATING THE PARAMETERS AT THE CODER

Image	bpp	Alg. 1 at the coder	Alg. 2 at the coder
airplane	0.32	30.92	30.94
airplane	0.55	34.25	34.26
Lena	0.29	31.37	31.38
Lena	0.54	34.76	34.77
peppers	0.32	30.60	30.63
peppers	0.53	32.48	32.51

Fig. 6. PSNR for different values of μ and ν on the *Lena* image compressed at 0.29 bpp.

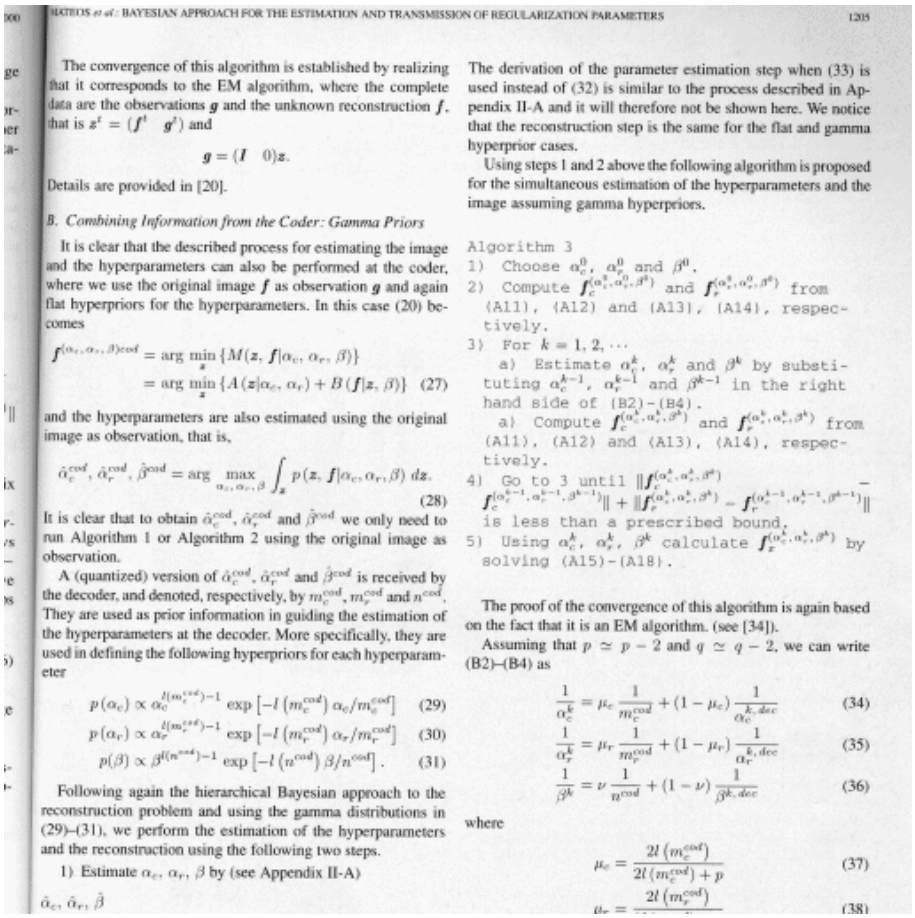
original image as observation, and then using these parameters in (20) to obtain the reconstruction. The results are shown in Table II. It can be seen that the PSNR improves slightly in this case.

The parameters obtained at the coder and the decoder were then combined. The same normalized confidence parameters μ_c and μ_r, defined in (37) and (38), were used for α_c and α_r. The values used in the experiments were $\mu_c = \mu_r = \mu \in \{0.0, 0.1, \cdots, 1\}$. The normalized confidence parameter ν, defined in (39), belongs to the same range. The 3-D plot in Fig. 6 shows the PSNR as a function of μ and ν for the *Lena* highly compressed image. The center part of the compressed image and the best reconstruction, corresponding to the parameter values $m_c^{cod} = \alpha_c^{cod} = 30.82^{-1}$, $m_r^{cod} = \alpha_r^{cod} = 5.36^{-1}$ and $n^{cod} = \beta^{cod} = 36.36^{-1}$ with $\mu = 0.9$ and $\nu = 0.0$ is displayed in Fig. 7(b). The corresponding PSNR is 31.40 dB. Similar results are obtained using other high compressed images showing that best reconstructions in terms of PSNR are obtained using μ

(a)

(b)

Figura A.8: *PP1209* Original, 512×512, 8*bpp*.



MATEOS *et al.*: BAYESIAN APPROACH FOR THE ESTIMATION AND TRANSMISSION OF REGULARIZATION PARAMETERS 1205

The convergence of this algorithm is established by realizing that it corresponds to the EM algorithm, where the complete data are the observations $\boldsymbol{g}$ and the unknown reconstruction $\boldsymbol{f}$, that is $\boldsymbol{z}^t = (\boldsymbol{f}^t \quad \boldsymbol{g}^t)$ and

$$\boldsymbol{g} = (\boldsymbol{I} \quad 0)\boldsymbol{z}.$$

Details are provided in [20].

B. Combining Information from the Coder: Gamma Priors

It is clear that the described process for estimating the image and the hyperparameters can also be performed at the coder, where we use the original image $\boldsymbol{f}$ as observation $\boldsymbol{g}$ and again flat hyperpriors for the hyperparameters. In this case (20) becomes

$$\begin{aligned} \boldsymbol{f}^{(\alpha_c,\alpha_r,\beta)\text{cod}} &= \arg\min_{\boldsymbol{z}} \{M(\boldsymbol{z}, \boldsymbol{f}|\alpha_c, \alpha_r, \beta)\} \\ &= \arg\min_{\boldsymbol{z}} \{A(\boldsymbol{z}|\alpha_c, \alpha_r) + B(\boldsymbol{f}|\boldsymbol{z}, \beta)\} \quad (27) \end{aligned}$$

and the hyperparameters are also estimated using the original image as observation, that is,

$$\hat{\alpha}_c^{\text{cod}}, \hat{\alpha}_r^{\text{cod}}, \hat{\beta}^{\text{cod}} = \arg\max_{\alpha_c,\alpha_r,\beta} \int_{\boldsymbol{z}} p(\boldsymbol{z}, \boldsymbol{f}|\alpha_c, \alpha_r, \beta)\, d\boldsymbol{z}. \quad (28)$$

It is clear that to obtain $\hat{\alpha}_c^{\text{cod}}$, $\hat{\alpha}_r^{\text{cod}}$ and $\hat{\beta}^{\text{cod}}$ we only need to run Algorithm 1 or Algorithm 2 using the original image as observation.

A (quantized) version of $\hat{\alpha}_c^{\text{cod}}$, $\hat{\alpha}_r^{\text{cod}}$ and $\hat{\beta}^{\text{cod}}$ is received by the decoder, and denoted, respectively, by m_c^{cod}, m_r^{cod} and n^{cod}. They are used as prior information in guiding the estimation of the hyperparameters at the decoder. More specifically, they are used in defining the following hyperpriors for each hyperparameter

$$p(\alpha_c) \propto \alpha_c^{l(m_c^{\text{cod}})-1} \exp\left[-l\left(m_c^{\text{cod}}\right)\alpha_c/m_c^{\text{cod}}\right] \quad (29)$$

$$p(\alpha_r) \propto \alpha_r^{l(m_r^{\text{cod}})-1} \exp\left[-l\left(m_r^{\text{cod}}\right)\alpha_r/m_r^{\text{cod}}\right] \quad (30)$$

$$p(\beta) \propto \beta^{l(n^{\text{cod}})-1} \exp\left[-l\left(n^{\text{cod}}\right)\beta/n^{\text{cod}}\right]. \quad (31)$$

Following again the hierarchical Bayesian approach to the reconstruction problem and using the gamma distributions in (29)–(31), we perform the estimation of the hyperparameters and the reconstruction using the following two steps.

1) Estimate α_c, α_r, β by (see Appendix II-A)

$\hat{\alpha}_c$, $\hat{\alpha}_r$, $\hat{\beta}$

The derivation of the parameter estimation step when (33) is used instead of (32) is similar to the process described in Appendix II-A and it will therefore not be shown here. We notice that the reconstruction step is the same for the flat and gamma hyperprior cases.

Using steps 1 and 2 above the following algorithm is proposed for the simultaneous estimation of the hyperparameters and the image assuming gamma hyperpriors.

Algorithm 3

1) Choose α_c^0, α_r^0 and β^0.
2) Compute $\boldsymbol{f}_c^{(\alpha_c^0,\alpha_r^0,\beta^0)}$ and $\boldsymbol{f}_r^{(\alpha_c^0,\alpha_r^0,\beta^0)}$ from (A11), (A12) and (A13), (A14), respectively.
3) For $k = 1, 2, \cdots$
   a) Estimate α_c^k, α_r^k and β^k by substituting α_c^{k-1}, α_r^{k-1} and β^{k-1} in the right hand side of (B2)–(B4).
   a) Compute $\boldsymbol{f}_c^{(\alpha_c^k,\alpha_r^k,\beta^k)}$ and $\boldsymbol{f}_r^{(\alpha_c^k,\alpha_r^k,\beta^k)}$ from (A11), (A12) and (A13), (A14), respectively.
4) Go to 3 until $\|\boldsymbol{f}_r^{(\alpha_c^k,\alpha_r^k,\beta^k)} - \boldsymbol{f}_c^{(\alpha_c^{k-1},\alpha_r^{k-1},\beta^{k-1})}\| + \|\boldsymbol{f}_r^{(\alpha_c^k,\alpha_r^k,\beta^k)} - \boldsymbol{f}_r^{(\alpha_c^{k-1},\alpha_r^{k-1},\beta^{k-1})}\|$ is less than a prescribed bound.
5) Using α_c^k, α_r^k, β^k calculate $\boldsymbol{f}_x^{(\alpha_c^k,\alpha_r^k,\beta^k)}$ by solving (A15)–(A18).

The proof of the convergence of this algorithm is again based on the fact that it is an EM algorithm. (see [34]).

Assuming that $p \simeq p-2$ and $q \simeq q-2$, we can write (B2)–(B4) as

$$\frac{1}{\alpha_c^k} = \mu_c \frac{1}{m_c^{\text{cod}}} + (1-\mu_c)\frac{1}{\alpha_c^{k,dec}} \quad (34)$$

$$\frac{1}{\alpha_r^k} = \mu_r \frac{1}{m_r^{\text{cod}}} + (1-\mu_r)\frac{1}{\alpha_r^{k,dec}} \quad (35)$$

$$\frac{1}{\beta^k} = \nu \frac{1}{n^{\text{cod}}} + (1-\nu)\frac{1}{\beta^{k,dec}} \quad (36)$$

where

$$\mu_c = \frac{2l\left(m_c^{\text{cod}}\right)}{2l\left(m_c^{\text{cod}}\right)+p} \quad (37)$$

$$\mu_r = \frac{2l\left(m_r^{\text{cod}}\right)}{\quad} \quad (38)$$

Figura A.9: *PP1205* Original, 512×512, 8*bpp*.