COMPRESSÃO DE IMAGENS USANDO DECOMPOSIÇÃO EM DICIONÁRIOS MULTIESCALAS ADAPTATIVOS

Marcelo Borges de Gouvêa

DISSERTAÇÃO 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. Sérgio Lima Netto, Ph.D.

RIO DE JANEIRO, RJ - BRASIL
OUTUBRO DE 2007

GOUVÊA, MARCELO BORGES DE

Compressão de Imagens Usando Decomposição em Dicionários Multiescalas Adaptativos [Rio de Janeiro] 2007

XI, 122 pp 29,7 cm (COPPE/UFRJ, M.Sc. Engenharia Elétrica, 2007)

Dissertação - Universidade Federal do Rio de Janeiro, COPPE

1. Compressão de Imagens
2. Casamento de Padrões Multiescalas
3. Codificação por Transformada
4. Otimização Taxa-distorção
5. Aproximações Sucessivas

I. COPPE/UFRJ II.Título (série)

Agradecimentos

Quero agradecer primeiramente à minha família, em especial aos meus pais, Eduardo e Suely, que deram tudo de si pela minha educação e formação. Também à minha irmã, Renata, particularmente pelos momentos engraçados.

Agradeço imensamente à minha amada e eterna namorada, e agora também esposa, Kris, pelo grande incentivo e carinho durante todo o curso e toda nossa vida juntos.

Obrigado, também, ao meu orientador, Prof. Eduardo, que soube como poucos me guiar durante todo o trabalho e cuja ajuda sempre foi muito valiosa e esclarecedora.

Finalmente, agradeço a todos, incluindo professores e membros da COPPE, que, de alguma forma, colaboraram com o desenvolvimento deste trabalho.

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

COMPRESSÃO DE IMAGENS USANDO DECOMPOSIÇÃO EM DICIONÁRIOS MULTIESCALAS ADAPTATIVOS

Marcelo Borges de Gouvêa

Outubro/2007

Orientador: Eduardo Antônio Barros da Silva

Programa: Engenharia Elétrica

Este trabalho propõe a incorporação de três conceitos aos algoritmos UMMP e MMP, que são métodos de compressão baseados em casamento aproximado de padrões recorrentes multiescalas. Primeiramente, implementamos o UMMP-RD, uma versão do UMMP com otimização taxa-distorção, obtendo um ganho de desempenho em relação ao algoritmo original. A seguir, tentamos inserir a idéia de codificação por transformada ao UMMP-RD, criando o UMMP-H, que, basicamente, escolhe, para cada bloco da imagem, o tipo de codificador que consegue um melhor resultado e o utiliza para a codificação. Finalmente, incorporamos o conceito de aproximações sucessivas ao MMP, gerando o MMP-SA, algoritmo que, apesar de pouco inferior ao MMP-RDI, pode ser bastante útil em determinadas aplicações, já que se caracteriza pela possibilidade de abrigar em um só *bitstream* diferentes aproximações da imagem original, cada uma com um grau de fidelidade e uma taxa de compressão.

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

IMAGE COMPRESSION USING DECOMPOSITION ON ADAPTATIVE MULTISCALE DICTICIONARIES

Marcelo Borges de Gouvêa

October/2007

Advisor: Eduardo Antônio Barros da Silva

Department: Electrical Engineering

This work proposes the incorporation of three concepts to the UMMP and MMP algorithms, which are compression methods based on approximate recurrent pattern matching with scales. First, we implement the UMMP-RD, a version of UMMP with rate-distortion optimization that improves the performance of the original algorithm. Later, we try to insert the idea of transform coding in UMMP-RD, creating the UMMP-H algorithm, that, basically, chooses, for each image block, which encoder obtains the best result, and then uses it to encode the block. Finally, we incorporate the concept of successive approximations to the MMP, generating the MMP-SA algorithm, which, although a little worse than MMP-RDI, can be very useful in certain applications, since it can have in only one bitstream different approximations of the original image, each one with a degree of fidelity and a compression rate.

# Sumário

**1 Introdução** **1**

1.1 Organização da dissertação . . . . . 2

**2 Algoritmo MMP** **3**

2.1 Introdução . . . . . 3

2.2 Descrição e implementação do MMP . . . . . 4

2.3 Algoritmo MMP-RDI . . . . . 10

2.4 Resultados experimentais . . . . . 12

**3 Algoritmo UMMP** **18**

3.1 Introdução . . . . . 18

3.2 O algoritmo *Matching Pursuits* . . . . . 19

3.3 Descrição do UMMP . . . . . 20

3.4 Resultados experimentais . . . . . 22

**4 Otimização taxa-distorção no UMMP** **31**

4.1 Introdução . . . . . 31

4.2 UMMP-RD1 . . . . . 32

4.3 UMMP-RD2 . . . . . 37

4.4 UMMP-RD3 . . . . . 41

4.5 Resultados experimentais . . . . . 43

**5 Algoritmo UMMP-H** **50**
5.1 Introdução . . . . . 50
5.2 Codificadores por transformada . . . . . 51
5.3 Codificador de Haar . . . . . 52
5.4 UMMP-H1 . . . . . 55
5.5 UMMP-H2 . . . . . 58
5.6 Resultados experimentais . . . . . 59

**6 Algoritmo MMP-SA** **67**
6.1 Introdução . . . . . 67
6.2 Descrição e implementação do MMP-SA . . . . . 68
6.3 MMP-SA2 . . . . . 73
6.4 MMP-SA3 . . . . . 74
6.5 MMP-SA4 . . . . . 76
6.6 Avaliação acerca da seqüência de lambdas . . . . . 78

**7 Conclusão** **94**

**A Algoritmo MMP** **97**

**B Algoritmo MMP-RDI** **100**

**C Algoritmo UMMP** **104**

**D Algoritmo do Codificador de Haar** **106**

**E Imagens originais** **110**

**Referências Bibliográficas** **120**

# Lista de Figuras


2.1 Divisão de blocos no MMP . . . . . 5
2.2 Exemplo de árvore de segmentação no MMP . . . . . 6
2.3 Bloco segmentado no MMP. . . . . 6
2.4 Taxa-distorção para *aerial* comprimida pelo MMP-RDI . . . . . 13
2.5 Taxa-distorção para *barbara* comprimida pelo MMP-RDI . . . . . 14
2.6 Taxa-distorção para *f-16* comprimida pelo MMP-RDI. . . . . 14
2.7 Taxa-distorção para *gold* comprimida pelo MMP-RDI . . . . . 15
2.8 Taxa-distorção para *lena* comprimida pelo MMP-RDI . . . . . 15
2.9 Taxa-distorção para *pp1205* comprimida pelo MMP-RDI . . . . . 16
2.10 Taxa-distorção para *pp1209* comprimida pelo MMP-RDI . . . . . 16
3.1 Exemplo de árvore de segmentação do UMMP. . . . . 21
3.2 Atualização do dicionário no UMMP. . . . . 22
3.3 Taxa-distorção para *aerial* comprimida pelo UMMP. . . . . 25
3.4 Taxa-distorção para *baboon* comprimida pelo UMMP. . . . . 25
3.5 Taxa-distorção para *barbara* comprimida pelo UMMP . . . . . 26
3.6 Taxa-distorção para *bridge* comprimida pelo UMMP . . . . . 26
3.7 Taxa-distorção para *f-16* comprimida pelo UMMP . . . . . 27
3.8 Taxa-distorção para *gold* comprimida pelo UMMP . . . . . 27
3.9 Taxa-distorção para *lena* comprimida pelo UMMP . . . . . 28
3.10 Taxa-distorção para *pp1205* comprimida pelo UMMP. . . . . 28
3.11 Taxa-distorção para *pp1209* comprimida pelo UMMP. . . . . 29
3.12 Taxa-distorção para *lena* comprimida pelo UMMP . . . . . 30
4.1 Taxa-distorção para *lena* comprimida pelo UMMP-RD1. . . . . 36
4.2 Taxa-distorção para *pp1205* comprimida pelo UMMP-RD1 . . . . . 37
4.3 Esquema de dicionários no UMMP-RD2 . . . . . 38
4.4 Divisão de cada um dos dicionários no UMMP-RD3 . . . . . 42

4.5 Atualização do dicionário no UMMP . . . . . 42
4.6 Taxa-distorção para *aerial* comprimida pelos UMMP-RDs. . . . . 44
4.7 Taxa-distorção para *baboon* comprimida pelos UMMP-RDs . . . . . 44
4.8 Taxa-distorção para *barbara* comprimida pelos UMMP-RDs. . . . . 45
4.9 Taxa-distorção para *bridge* comprimida pelos UMMP-RDs . . . . . 45
4.10 Taxa-distorção para *f-16* comprimida pelos UMMP-RDs . . . . . 46
4.11 Taxa-distorção para *gold* comprimida pelos UMMP-RDs . . . . . 46
4.12 Taxa-distorção para *lena* comprimida pelos UMMP-RDs . . . . . 47
4.13 Taxa-distorção para *pp1205* comprimida pelos UMMP-RDs . . . . . 47
4.14 Taxa-distorção para *pp1209* comprimida pelos UMMP-RDs . . . . . 48
5.1 Exemplo de divisão em *quadtree* no MGE . . . . . 54
5.2 Divisão do domínio da Transformada de Haar . . . . . 57
5.3 Taxa-distorção para *aerial* comprimida pelo UMMP-H . . . . . 60
5.4 Taxa-distorção para *baboon* comprimida pelo UMMP-H. . . . . 60
5.5 Taxa-distorção para *barbara* comprimida pelo UMMP-H . . . . . 61
5.6 Taxa-distorção para *bridge* comprimida pelo UMMP-H . . . . . 61
5.7 Taxa-distorção para *f-16* comprimida pelo UMMP-H . . . . . 62
5.8 Taxa-distorção para *gold* comprimida pelo UMMP-H . . . . . 62
5.9 Taxa-distorção para *lena* comprimida pelo UMMP-H . . . . . 63
5.10 Taxa-distorção para *pp1205* comprimida pelo UMMP-H. . . . . 63
5.11 Taxa-distorção para *pp1209* comprimida pelo UMMP-H. . . . . 64
5.12 Tipos de blocos utilizados para *lena* comprimida pelo UMMP-H1. . . . . 65
5.13 Tipos de blocos utilizados para *lena* comprimida pelo UMMP-H2. . . . . 66
6.1 Exemplo de divisão de um bloco no MMP-RDI . . . . . 70
6.2 Taxa-distorção para *lena* comprimida pelo MMP-SA . . . . . 72
6.3 Taxa-distorção para *lena* comprimida pelo MMP-SA2 . . . . . 74
6.4 Taxa-distorção para *lena* comprimida pelo MMP-SA3 . . . . . 76
6.5 Taxa-distorção para *lena* comprimida pelo MMP-SA4 . . . . . 77
6.6 Resultados do MMP-SA para os pares *{500, x}* . . . . . 79
6.7 Resultados do MMP-SA para os pares *{1000, x}* . . . . . 80
6.8 Resultados do MMP-SA para os pares *{2500, x}* . . . . . 80
6.9 Resultados do MMP-SA para os pares *{5000, x}* . . . . . 81
6.10 Resultados do MMP-SA para os pares *{10000, x}* . . . . . 81
6.11 Resultados do MMP-SA para os pares *{20000, x}* . . . . . 82

6.12 Resultados do MMP-SA para os pares *{40000, x}* . . . . . 82
6.13 Resultados do MMP-SA para os pares *{80000, x}* . . . . . 83
6.14 Resultados do MMP-SA para os pares *{150000, x}* . . . . . 83
6.15 Resultados dos testes com o MMP-SA com dois lambdas . . . . . 85
6.16 Resultados do MMP-SA para os pares *{250,100, x}*. .. . . . . 87
6.17 Resultados do MMP-SA para os pares *{1000,500, x}* . . . . . 87
6.18 Resultados do MMP-SA para os pares *{5000,1000, x}*. . . . . 88
6.19 Resultados do MMP-SA para os pares *{10000,500, x}*. . . . . 88
6.20 Resultados do MMP-SA para os pares *{20000,250, x}*. . . . . 89
6.21 Resultados do MMP-SA para os pares *{20000,5000, x}* . . . . . 89
6.22 Resultados do MMP-SA para os pares *{40000,1000, x}* . . . . . 90
6.23 Resultados do MMP-SA para os pares *{80000,1000, x}* . . . . . 90
6.24 Resultados do MMP-SA para os pares *{80000,40000, x}*. . . . . 91
6.25 Resultados dos testes com o MMP-SA com três lambdas . . . . . 93
D.1 Divisão em *quadtree* utilizada na implementação do Codificador de Haar . . 107
E.1 Imagem *aerial* original, 512x512 *pixels* e 8bpp . . . . . 111
E.2 Imagem *baboon* original, 512x512 *pixels* e 8bpp . . . . . 112
E.3 Imagem *barbara* original, 512x512 *pixels* e 8bpp. . . . . 113
E.4 Imagem *bridge* original, 512x512 *pixels* e 8bpp . . . . . 114
E.5 Imagem *f-16* original, 512x512 *pixels* e 8bpp . . . . . 115
E.6 Imagem *gold* original, 512x512 *pixels* e 8bpp . . . . . 116
E.7 Imagem *lena* original, 512x512 *pixels* e 8bpp . . . . . 117
E.8 Imagem *pp1205* original, 512x512 *pixels* e 8bpp . . . . . 118
E.9 Imagem *pp1209* original, 512x512 *pixels* e 8bpp . . . . . 119

# Lista de Tabelas

6.1 Pares de lambdas usados nos testes do MMP-SA . . . . . 79
6.2 Perdas do MMP-SA em relação ao MMP-RDI . . . . . 84
6.3 Trincas de lambdas usadas nos testes do MMP-SA . . . . . 86
6.4 Menores perdas do MMP-SA em relação ao MMP-RDI . . . . . 91
6.5 Melhor segundo lambda para trincas com os mesmos lambdas inicial e final. . 92

# Capítulo 1

# Introdução

Já há algum tempo os formatos digitais vêm ganhando cada vez mais espaço em diversas áreas de aplicação. Os exemplos são inúmeros, como as tomografias computadorizadas na medicina, as imagens de satélite usadas na meteorologia, o áudio digital na telefonia IP e o vídeo digital no entretenimento multimídia. Formatos digitais já consagrados, como o MP3 e o JPEG, substituem com grandes vantagens seus equivalentes analógicos.

Assim, apesar de o avanço da tecnologia nos fornecer capacidades de transmissão e armazenamento cada vez maiores, há uma crescente necessidade de compressão de dados. Uma simples imagem digital, por exemplo, contém um volume grande de bits de informação, tornando um algoritmo de compressão de enorme importância.

O estudo da compressão de sinais digitais foi iniciado há mais de 30 anos por Shannon, que fundou a chamada Teoria da Informação [1] e introduziu conceitos utilizados até hoje no ramo de compressão sem perdas, no qual a decodificação gera um sinal idêntico ao original.

A Teoria Taxa-Distorção [2] é a base da compressão com perdas, o outro ramo da compressão de sinais, no qual o resultado da decodificação é um sinal diferente do original. Neste caso, a qualidade requerida na reprodução do sinal é inversamente proporcional à taxa de compressão que se pode obter, ou seja, o quão mais parecido com o sinal de entrada se deseja que o sinal de saída seja, mais bits serão necessários para a sua representação.

O estado da arte em compressão de imagens, atualmente, são os codificadores por transformada, que executam a tarefa em três passos: aplicação de uma transformada ao sinal de entrada, seguida de quantização e codificação dos coeficientes obtidos. Dois dos principais codificadores pertencentes a esta classe são o JPEG [3], que utiliza a Transformada Discreta dos Cossenos (DCT) e o SPIHT [4], cuja transformada é a DWT (Transformada Discreta de Wavelet).

Neste trabalho buscamos, por meio da incorporação de três conceitos, melhorar o desempenho de dois algoritmos de compressão com perdas, propostos em [5], baseados em casamento aproximado de padrões recorrentes com escalas. O MMP (*Multi-dimensional Multiscale Parser*) utiliza versões em escala de vetores de um dicionário para representar um bloco de dados do sinal de entrada. Já o UMMP (*Universal Multiscale Matching Pursuits*) utiliza uma combinação linear destes vetores para a representação. Ambos constroem seus dicionários com concatenações de blocos previamente codificados.

## 1.1 Organização da dissertação

Além do presente capítulo de introdução, este trabalho é desenvolvido em outros seis capítulos da maneira a seguir detalhada.

No Capítulo 2, apresentamos uma descrição do algoritmo MMP originalmente proposto e de uma versão deste método com otimização taxa-distorção.

No Capítulo 3, descrevemos o algoritmo UMMP, mostrando os resultados por ele conseguidos na compressão de diferentes tipos de imagens.

A partir daí são mostradas e testadas as variações desses algoritmos propostas neste trabalho. Primeiramente, no capítulo 4, implementamos a otimização taxa-distorção no UMMP.

No Capítulo 5, introduzimos o UMMP-H, algoritmo que se utiliza da Transformada de Haar para tentar obter ganhos na compressão.

No Capítulo 6, propomos agregar ao MMP a idéia de codificação por aproximações sucessivas presente em outros tipos de codificadores.

Finalmente, concluímos, no Capítulo 7, comentando acerca dos resultados obtidos e mostrando problemas encontrados e sugestões para futuros estudos.

# Capítulo 2

# Algoritmo MMP

Neste capítulo descrevemos o MMP (Multi-dimensional Multiscale Parser), um algoritmo de compressão com perdas proposto em [5]. Ele é baseado no casamento aproximado de padrões recorrentes multiescala. Após a descrição do método originalmente proposto, mostramos também uma versão com otimização taxa-distorção, proposta em [6].

## 2.1 Introdução

Representando uma nova classe de algoritmos de compressão com perdas o MMP foi introduzido em [5]. Nele, busca-se em um dicionário um vetor que melhor aproxime um bloco do sinal de entrada. Se esta aproximação não atender a um critério de qualidade, como um valor máximo para a distorção, o bloco de entrada é particionado e cada parte é aproximada individualmente. O processo se repete até que o critério de qualidade seja atingido.

Este método é dito multiescala porque os vetores do dicionário são comprimidos e estendidos para que possam representar blocos de entrada de diferentes tamanhos, já que estes podem ser divididos ou não.

O dicionário vai sendo atualizado, ao longo da codificação, com a inserção da concatenação de blocos já codificados. Isso torna o algoritmo adaptativo, ou seja, independentemente do dicionário inicial, ele vai se adaptando às características do sinal

a ser codificado. Esta peculiaridade torna o MMP um algoritmo universal, já que ele produz bons resultados para todos os tipos de sinais, desde os mais suaves até aqueles com muitas componentes de alta freqüência.

Apesar de poder ser aplicado a qualquer tipo de sinal multidimensional, o MMP vem sendo desenvolvido voltado para a compressão de imagens. Neste campo, resultados mostram que o MMP se aproxima bastante dos melhores métodos existentes para as imagens suaves e os supera para imagens com detalhes de alta freqüência, como as compostas exclusivamente de texto e as que possuem texto mesclado com imagens mais suaves, denominadas imagens mistas.

## 2.2 Descrição e implementação do MMP

Como estamos interessados em compressão de imagens, descreveremos nesta seção apenas o algoritmo bidimensional. Para isto, consideremos que a imagem seja dividida em blocos de tamanho 8x8. Cada bloco $X_0$ de dimensão $m \times n$ é aproximado pelo elemento $S_j$ pertencente a um dicionário $D$ que melhor o representa. O dicionário é construído a partir de um dicionário inicial atualizado durante a codificação.

A distorção resultante desta aproximação é então comparada a uma distorção-alvo $\delta$, devendo atender à seguinte relação:

$$\left\| X_0 - S_j \right\|^2 = \sum_k \sum_l \left( x_{k,l} - s_{k,l} \right)^2 \leq \delta^2 .m.n \qquad (2.1)$$

Ou seja, o erro médio quadrático deve ser menor ou igual à distorção-alvo. Caso nenhum elemento do dicionário atenda a este critério de qualidade, o bloco $X_0$ é dividido em dois blocos menores, $X_1$ e $X_2$, como mostrado na Figura 2.1.

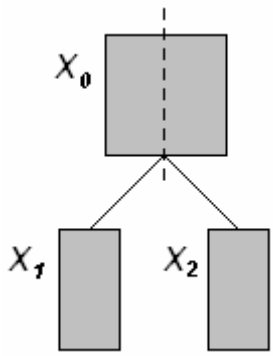


Figura 2.1: Divisão de blocos no MMP.

Para cada um dos blocos resultantes é feito o mesmo procedimento. Assim, de maneira geral, busca-se no dicionário *D* um elemento $S_k^{m,n}$ que melhor aproxime, de acordo com o erro médio quadrático, um bloco $X_j^{m,n}$, onde a dimensão dos vetores é *m x n*. Ressalta-se que $S_k^{m,n}$ é proveniente da aplicação de uma transformação de escala ao elemento $S_j$ do dicionário. Verifica-se, então, se a condição de distorção máxima é atendida, ou seja, se a Equação 2.2 é satisfeita.

$$\left\| X_j^{m,n} - S_j^{m,n} \right\|^2 \leq \delta^2 .m.n \qquad (2.2)$$

Caso contrário, o bloco $X_j^{m,n}$ é dividido, gerando dois blocos menores, que serão codificados da mesma maneira.

Devido a esta sistemática, podemos observar que o MMP pode ser utilizado para uma compressão sem perdas, bastando, para isso, que a distorção-alvo seja zero e que o dicionário contenha todos os níveis presentes na imagem original.

Quando a condição da Equação 2.2 é atendida, o vetor $S_k^{m,n}$ será a representação do bloco $X_j^{m,n}$ e o algoritmo passa para a aproximação do bloco seguinte. Ao final, poderemos representar as partições executadas por uma árvore binária, como a mostrada na Figura 2.2.

No MMP, convencionou-se que a segmentação seja feita de forma que, se *m* = *n*, a partição é vertical, ou seja os blocos-filhos serão de dimensão *m x n/2*, e se *m* > *n*, a partição é horizontal, ou seja os blocos-filhos terão a dimensão *m/2 x n*. Assim, ao particionarmos um bloco $X_j$, geramos os blocos-filhos $X_{2j+1}$ e $X_{2j+2}$.

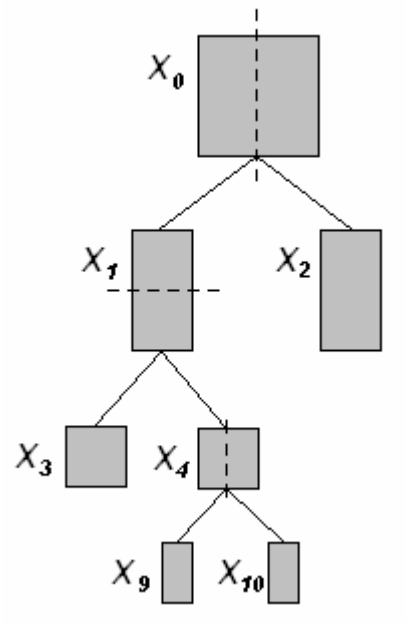


Figura 2.2: Exemplo de árvore de segmentação no MMP.

Para a codificação da imagem, utiliza-se o bit '0' para indicar a partição de um bloco (segmentação do nó da árvore) e o bit '1' para indicação de que houve sucesso na sua aproximação (nó-folha). Neste caso, após o bit '1', deve-se armazenar o índice *i* do elemento do dicionário usado para a aproximação.

No exemplo da árvore mostrada na Figura 2.2 o bloco 8x8 seria segmentado como mostrado abaixo.

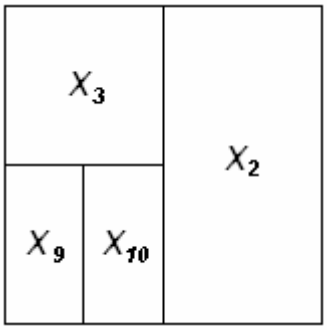


Figura 2.3: Bloco segmentado no MMP.

Assim, o bloco codificado seria composto pela concatenação dos nós-folhas da árvore binária equivalente, da seguinte forma:

$$\begin{aligned} \hat{X}_0 &= \left(\hat{X}_1 \hat{X}_2\right) \\ \hat{X}_1 &= \begin{pmatrix} \hat{X}_3 \\ \hat{X}_4 \end{pmatrix} \\ \hat{X}_4 &= \left(\hat{X}_9 \hat{X}_{10}\right) \end{aligned} \tag{2.4}$$

Outro ponto importante do MMP é a atualização de seu dicionário. Aí reside a sua característica de universalidade, já que, por ser um algoritmo adaptativo, ele é capaz de aprender as características do sinal que está sendo codificado, incorporando ao dicionário as seqüências típicas presentes no sinal. Assim, a codificação torna-se mais eficiente para qualquer classe de imagens.

Esta é a principal vantagem do MMP em relação aos codificadores por transformada, pois estes supõem que a imagem é suave, ou seja, não possui componentes relevantes de alta freqüência, enquanto que nenhuma suposição acerca da imagem é necessária ao MMP. Isto ficará evidente quando fizermos a comparação dos resultados obtidos em testes com imagens mistas e de texto.

A atualização do dicionário do MMP é feita de maneira bem simples. Quando estiverem disponíveis as aproximações dos dois blocos que possuem o mesmo nó-pai, estas são concatenadas gerando a aproximação do bloco referente a este nó-pai. E é exatamente este vetor que, após sofrer a adequada transformação de escala, será inserido no dicionário.

No exemplo da Figura 2.2, ao efetuarmos as aproximações $\hat{X}_9$ e $\hat{X}_{10}$, inserimos no dicionário a concatenação, dilatada, destes dois vetores.

Uma alternativa à existência de um dicionário cujos elementos devem ser dilatados ou contraídos para aproximação dos blocos de entrada é a implementação de diversos dicionários, um para cada escala. Neste caso, diminui-se a necessidade de processamento em contrapartida ao aumento da quantidade de memória utilizada. Na fase de atualização, basta que se adicione uma cópia, dilatada ou contraída, do novo elemento em cada dicionário.

Esta segunda opção tem sido a mais utilizada para implementação do MMP, pois, além de tornar o algoritmo mais rápido, possui outra vantagem, que é a possibilidade de que cada um dos dicionários se diferencie em relação aos outros, melhorando a adaptabilidade do algoritmo. O crescimento diferente de cada dicionário se apóia no fato de que a transformação de escala não é biunívoca, ou seja, um elemento novo em uma escala pode gerar, depois de transformado, um elemento já existente em outra escala. Assim, um dicionário é incrementado e outro não.

A transformação de escala utilizada no MMP é feita um duas etapas. Primeiro, aplica-se uma transformada unidimensional a cada linha do bloco e, a seguir, aplica-se a mesma transformada às colunas. A transformada unidimensional $T_{N_0}^{N}[\cdot]$ gera um vetor de tamanho $N$ a partir de um vetor de tamanho $N_0$, e se baseia nas seguintes equações:

- Dilatação ($N0 < N$): Considerando-se $n = 0,1,...,N-1$ como cada coordenada do vetor transformado, podemos definir que o vetor transformado é dado por:

$$S_n^s = \left\lfloor \frac{\alpha_n \left( S_{m_n^1} - S_{m_n^0} \right)}{N} \right\rfloor + S_{m_n^0} \tag{2.5}$$

onde, sendo $\lfloor a \rfloor$ o maior inteiro menor ou igual a $a$, temos:

$$\begin{aligned} m_n^0 &= \left\lfloor \frac{n\left(N_0 - 1\right)}{N} \right\rfloor \\ m_n^1 &= \begin{cases} m_n^0 + 1, & m_n^0 < N_0 - 1 \\ m_n^0, & m_n^0 = N_0 - 1 \end{cases} \\ \alpha_n &= n\left(N_0 - 1\right) - N.m_n^0 \end{aligned} \tag{2.6}$$

- Contração ($N0 > N$): Considerando-se $n = 0,1,...,N-1$ como cada coordenada do vetor transformado, podemos definir que o vetor transformado é dado por:

$$S_n^s = S_{m_{n,k=0}^0} + \frac{1}{N_0 + 1} \sum_{k=0}^{N_0} \left\lfloor \frac{\alpha_{n,k} \left( S_{m_{n,k}^1} - S_{m_{n,k}^0} \right)}{N} \right\rfloor \tag{2.7}$$

onde, temos:

$$
\begin{aligned}
m_{n,k}^{0} &= \left\lfloor \frac{n\left(N_0 - 1\right) + k}{N} \right\rfloor \\
m_{n,k}^{1} &= \begin{cases} m_{n,k}^{0} + 1, & m_{n,k}^{0} < N_0 - 1 \\ m_{n,k}^{0}, & m_{n,k}^{0} = N_0 - 1 \end{cases} \\
\alpha_{n,k} &= n\left(N_0 - 1\right) + k - N.m_{n,k}^{0}
\end{aligned}
\tag{2.8}
$$

Para o caso bidimensional, considerando $N_0 \; x \; M_0$ a dimensão do bloco original, e $N \; x \; M$ a dimensão do bloco transformado, primeiramente transformamos todas as linhas segundo a Equação 2.9, com o número de linhas variando de $i = 0,1,..., N - 1$.

$$
Y_i = T_{M_0}^{M}\left[S_i\right] \tag{2.9}
$$

O segundo passo é a transformação das colunas (Equação 2.10), que é feita usando-se a transposta de $Y_i$ obtida na Equação 2.9, com o número de colunas variando de $j = 0,1,..., M - 1$.

$$
S_j^s = \left(T_{N_0}^{N}\left[\left(Y^T\right)_j\right]\right)^T \tag{2.10}
$$

Ressalta-se que a transformação de escala foi implementada utilizando-se operações de mudança de taxa de amostragem. Essencialmente, muda-se o tamanho do vetor original por meio de uma interpolação linear e da aplicação de um filtro de média, amostrando-se o resultado à taxa necessária.

A descrição completa da implementação em pseudocódigo do algoritmo MMP encontra-se no Apêndice A.

## 2.3 Algoritmo MMP-RDI

Conforme descrito, o MMP se baseia apenas em uma distorção-alvo para a decisão acerca da divisão ou não de um bloco. Este critério local não considera, por exemplo, o aumento do número de bits decorrente de uma divisão de bloco. Com a idéia de que a decisão sobre a divisão de blocos passasse a se basear em um critério global, surgiram o MMP-RD, em [5], e o MMP-RDI, em [6]. Os dois algoritmos exibem resultados semelhantes, com o segundo sendo menos complexo. Por isso utilizamos o MMP-RDI neste trabalho.

Agora, a partição de um bloco (ou nó de uma árvore binária) fica condicionada a uma avaliação global, que é feita baseada em custo *lagrangeano* [7,8]. Assim, procura-se, antes da codificação propriamente dita, fazer uma otimização taxa-distorção da árvore binária. Para tal considera-se não só a variável distorção, como também a variável taxa para a decisão de partição de blocos.

Na prática, calcula-se o custo de um nó e os custos de seus nós-filhos. Se a soma destes for maior que aquele, os nós-filhos (ou suas sub-árvores) são podadas. O custo de um dado nó $n_l$ é obtido da seguinte forma:

$$J_{n_l} = D_{n_l} + \lambda . R_{n_l} \tag{2.11}$$

onde $D_{n_l}$ é a distorção devida à aproximação de $X_l^{m,n}$, $R_{n_l}$ é o número de bits necessário para a representação do índice do elemento do dicionário escolhido, e λ é o chamado multiplicador de *Lagrange*.

Além de utilizar um critério global para a divisão de nós, o MMP-RDI também leva em conta a dependência que existe entre o resultado da aproximação de um nó e a configuração da árvore binária à sua esquerda. Esta dependência fica explicitada se pensarmos que um elemento que deixou de ser incluído no dicionário devido a uma poda da árvore poderia ser aquele que melhor aproximaria um bloco que ainda será codificado.

Esta análise de dependência gera um grande aumento do custo computacional, já que para que fosse tomada uma decisão de poda em um determinado nó deveriam ser analisados todos os nós à sua direita na árvore. Porém, quando o dicionário atinge tamanhos razoáveis, a dependência descrita se torna menos importante, já que as novas

adições feitas ao dicionário não serão tão diferentes dos elementos nele já existentes. Assim, o alto custo computacional não mais é justificado.

Com isto em mente, foi desenvolvido em [6] o algoritmo de otimização intermediário, o MMP-RDI, cuja principal característica consiste em tornar a otimização um espelho da codificação, reproduzindo todos os seus passos. Isto quer dizer que a otimização simula todas as atualizações de dicionário, codificações de *flag* e codificações de índice executadas pelo algoritmo de codificação.

Assim, para a rotina de otimização, cria-se um dicionário rascunho $D_R$ independente do dicionário principal $D$ para receber as atualizações que ocorreriam durante a codificação. Além disso, são necessários contadores de freqüência $C_{h_R}^{m,n}$ para os índices $h_R$ do dicionário rascunho para os cálculos dos custos quando algum elemento deste dicionário for escolhido, durante a otimização, para representar um dado bloco. Finalmente, também são necessários contadores complementares $\overline{C}_h^{m,n}$ e $\overline{C}_{h_F}^{m,n}$ para o dicionário principal $D$ e para os *flags* descritores da árvore de segmentação.

Com isto, executa-se a função de otimização antes da rotina de codificação, com o objetivo de gerar a árvore de segmentação ótima, de acordo com os parâmetros utilizados, para determinado bloco a ser codificado.

Inicialmente, a rotina de otimização vai gerando a árvore até o último nível (blocos 1x1), como na codificação, simultaneamente ao cálculo do custo de cada nó. A seguir, avalia-se se a soma dos custos de representação dos nós-folhas (blocos 1x1) é maior que aquele de seu nó-pai (bloco 2x1), caso em que eles serão podados e o nó-pai será um novo nó-folha. Caso contrário, eles são mantidos, o dicionário rascunho é atualizado e o custo do nó-pai é substituído pela soma dos seus custos.

Este procedimento é repetido até que seja montada a árvore otimizada. Uma observação importante é que devem ser considerados os custos para codificação dos *flags* na decisão de poda. Assim, o custo de um nó-folha é dado por:

$$J_l = D_{n_l} + \lambda.R_{n_l} + \lambda.R_{1l} \tag{2.10}$$

onde $J_l$ é o custo lagrangeano do nó-folha, $D_{n_l}$ 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, $R_{n_l}$ é o número de bits

necessário para a representação do índice $i$ ou $i_R$ escolhido, e $R_{1l}$ é o números de bits necessário para a representação do *flag* '1', que indica que o nó $n_l$ é um nó-folha.

O custo para a representação dos nós-filhos ($n_{2l+1}$ e $n_{2l+2}$) como nós-folhas, deve considerar, além dos *flags* '1', como anteriormente, o *flag* '0' que indica a partição do nó $n_l$. Assim, temos:

$$J_{2l+1,2l+2} = J_{2l+1} + J_{2l+2} + \lambda.R_{0l} \tag{2.11}$$

onde $J_{2l+1,2l+2}$ é o custo *lagrangeano* para a representação de $n_{2l+1}$ e $n_{2l+2}$, $J_{2l+1}$ e $J_{2l+2}$ são os custos *lagrangeanos* para a representação de $n_{2l+1}$ e $n_{2l+2}$ como nós-folhas, e $R_{0l}$ é o número de bits necessário para a representação do *flag* '0'.

Outro ponto importante a ser considerado é que, quando se faz um poda na árvore de segmentação, devem-se desfazer as atualizações relativas aos nós-folhas podados, ou seja, deve-se decrementar os contadores $\overline{C}_h^{m,n}$ e $C_{h_R}^{m,n}$, retirar os elementos inseridos no dicionário rascunho $D_R$ e adequar os contadores $\overline{C}_{h_F}^{m,n}$ à nova estrutura da árvore de segmentação.

Assim, a otimização torna-se um espelho da codificação, gerando uma árvore de segmentação $T$ que será posteriormente utilizada para a codificação do bloco de entrada $X_0^{m,n}$.

A descrição completa da implementação em pseudocódigo do algoritmo MMP-RDI encontra-se no Apêndice B.

## 2.4 Resultados experimentais

A seguir são reproduzidos os resultados obtidos em [9] para os algoritmos MMP e MMP-RDI, descritos nas seções anteriores. São mostrados também, para fins de comparação, resultados obtidos por outros codificadores, o JPEG [3] e o SPIHT [4].

As imagens comprimidas foram *aerial, barbara, f-16, gold, lena, pp1205 e pp1209*, todas de tamanho 512x512 *pixels*. As cinco primeiras imagens, mostradas no Apêndice E, foram obtidas no *site http://sipi.usc.edu/services/database/Database.html*,

e as duas últimas, mostradas nas Figuras E.8 e E.9, foram digitalizadas do IEEE *Transactions on Image Processing*, volume 9, número 7, de julho de 2000, páginas 1205 e 1209, respectivamente.

Para os testes com o MMP e o MMP-RDI, as imagens foram divididas em blocos de tamanho 16x16, sendo cada um deles processado individualmente, da esquerda para a direita e de cima para baixo. Cada um dos dicionários foi limitado a 100.000 blocos, apesar de este limite não ter sido atingido em nenhum momento.

Além disso, os dicionários foram inicializados conforme definido em [9], de maneira semelhante àquela descrita no Apêndice B.

Finalmente, foram utilizados codificadores aritméticos para os *flags* descritores da árvore de segmentação e para os índices dos elementos escolhidos para as aproximações, com modelos independentes para cada um dos dicionários.

A seguir, nas Figuras 2.4 a 2.10, encontram-se os gráficos reproduzidos de [9].

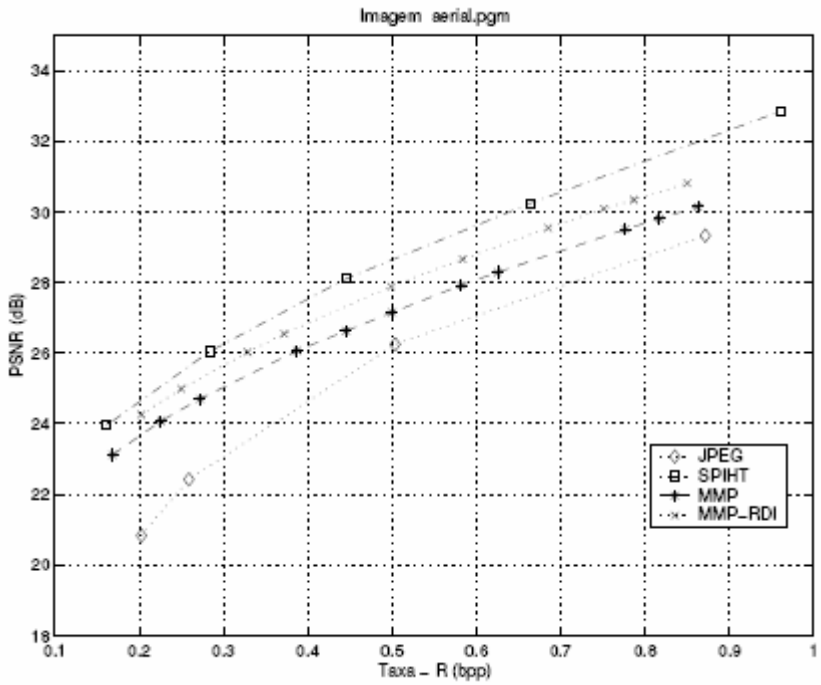


Figura 2.4: Taxa-distorção para *aerial* comprimida pelo MMP-RDI.

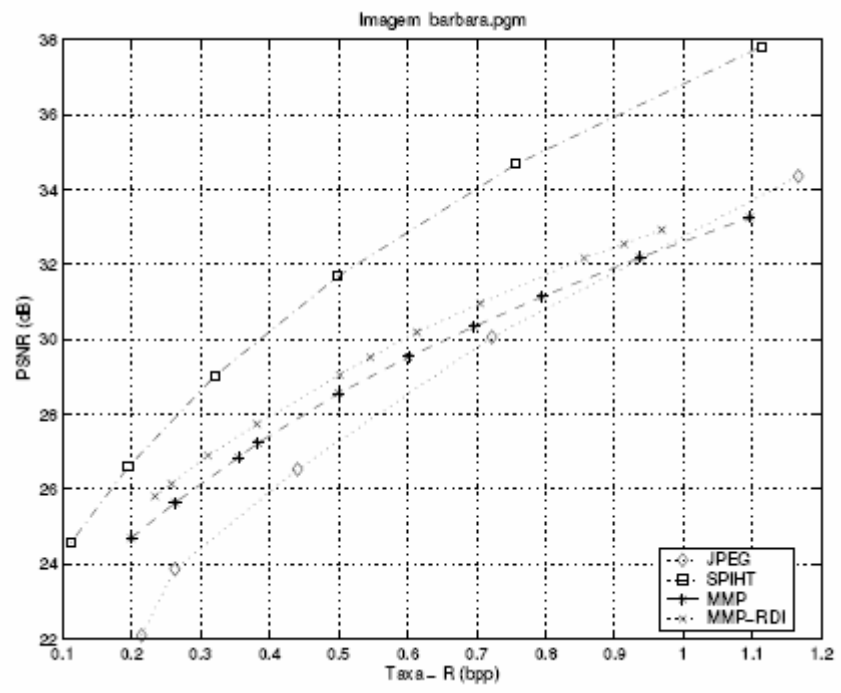


Figura 2.5: Taxa-distorção para *barbara* comprimida pelo MMP-RDI.

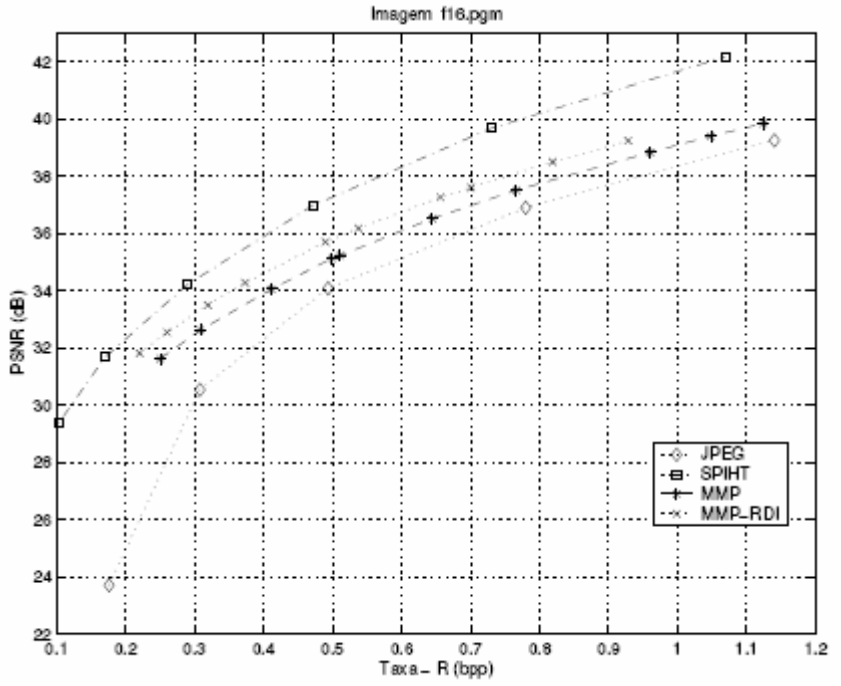


Figura 2.6: Taxa-distorção para *f-16* comprimida pelo MMP-RDI.

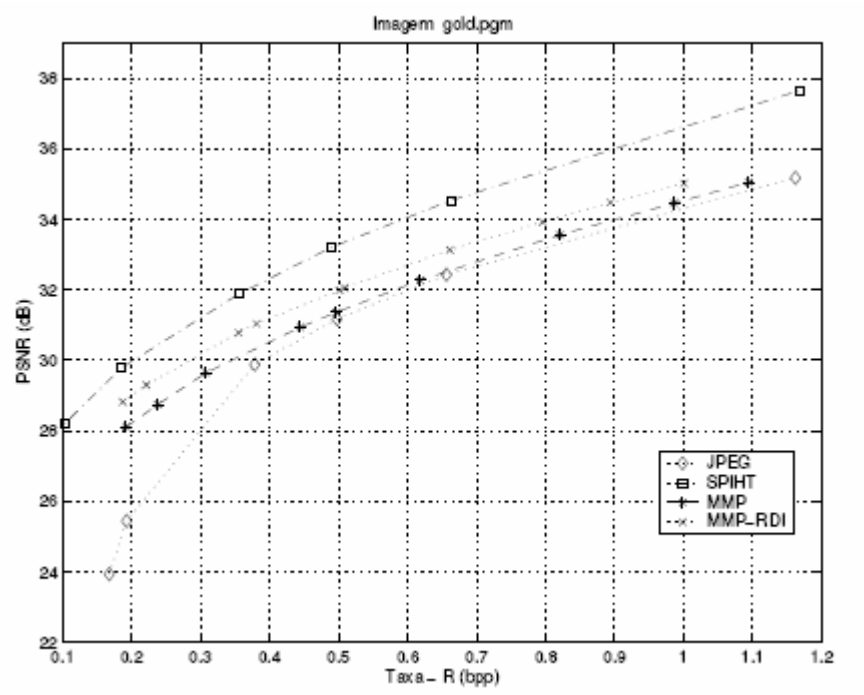


Figura 2.7: Taxa-distorção para *gold* comprimida pelo MMP-RDI.

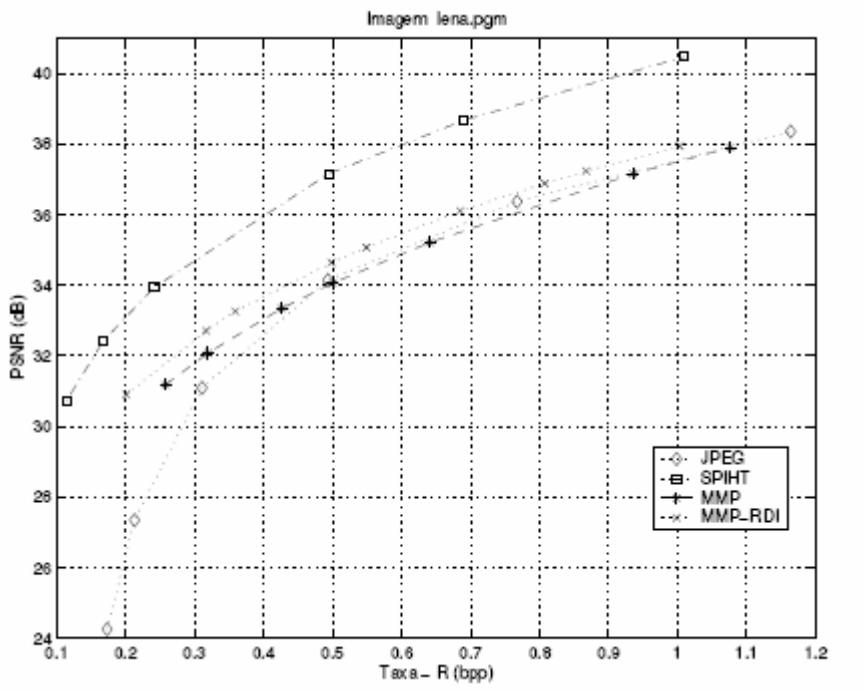


Figura 2.8: Taxa-distorção para *lena* comprimida pelo MMP-RDI.

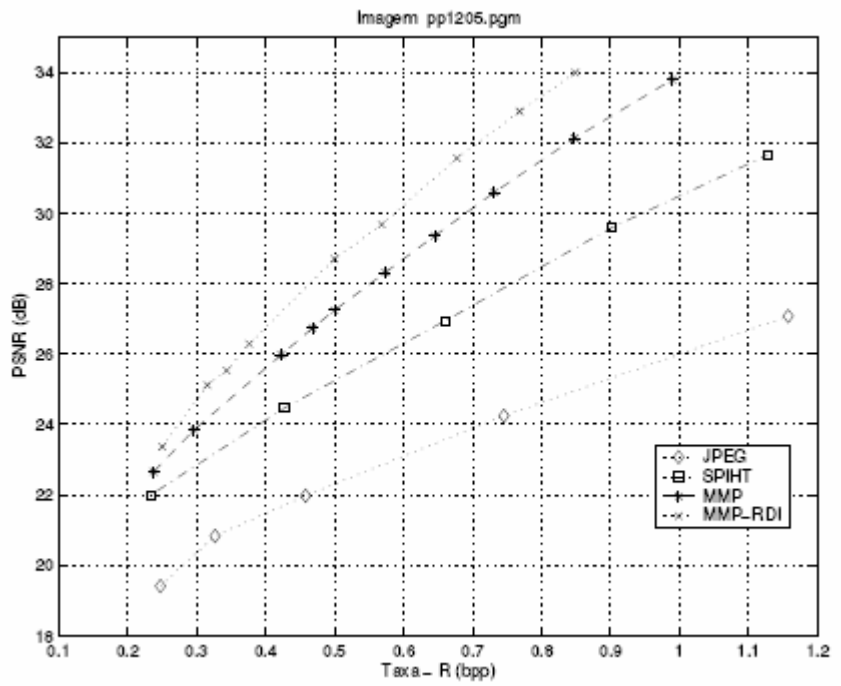


Figura 2.9: Taxa-distorção para *pp1205* comprimida pelo MMP-RDI.

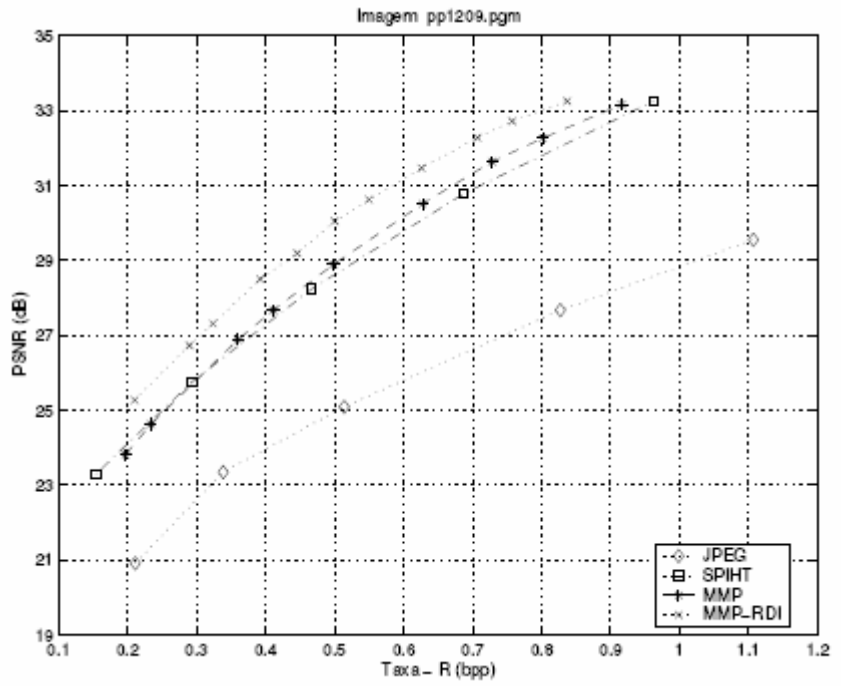


Figura 2.10: Taxa-distorção para *pp1209* comprimida pelo MMP-RDI.

Pela análise das curvas, primeiramente, podemos observar a clara melhoria de desempenho do MMP-RDI em relação ao MMP, devido à inserção da rotina de otimização taxa-distorção.

Outro ponto interessante é que o MMP-RDI não alcançou o SPIHT para as imagens suaves, mas o superou nas imagens de texto (*pp1205*) e mista (*pp1209*). Este fato evidencia a característica de universalidade presente nos algoritmos MMP, já que seu dicionário é adaptativo, ou seja, é capaz de se adaptar às características da imagem que está sendo codificada.

Já o SPIHT, assim como a grande maioria dos algoritmos de compressão de imagens existentes, supõe que a imagem a ser codificada é suave, não possuindo componentes relevantes de alta freqüência.

Assim, podemos dizer que a pequena perda de desempenho do MMP-RDI para imagens suaves é compensada pela sua característica de universalidade, tornando-o uma boa opção quando a suposição de suavidade da imagem não é apropriada.

Finalmente, observamos que o MMP-RDI foi superior ao JPEG, padrão largamente difundido, em todas as imagens analisadas.

# Capítulo 3

# Algoritmo UMMP

Neste capítulo descrevemos o UMMP (Universal Multiscale Matching Pursuits), um algoritmo de compressão com perdas, originalmente proposto em [5]. Assim como o MMP, visto no capítulo anterior, ele é baseado no casamento aproximado de padrões recorrentes multiescala.

## 3.1 Introdução

O algoritmo de compressão com perdas UMMP, introduzido em [5], é bastante similar em diversos aspectos ao MMP. Ele também se utiliza de um dicionário que é construído ao longo da codificação, além de ser multiescala. A principal diferença reside no fato de que o UMMP aproxima um bloco de entrada por uma combinação linear de elementos pertencentes ao dicionário, ao passo que, no MMP, a aproximação é feita por um único elemento do dicionário. O UMMP se baseia no algoritmo *Matching Pursuits*, descrito a seguir.

## 3.2 O algoritmo *Matching Pursuits*

O algoritmo *Matching Pursuits* [10] aproxima determinado sinal de entrada por uma combinação linear de elementos de um dicionário $D\{\phi_k\}_{k=0}^{K-1}$ arbitrário, que especifica um *frame* [11] no qual o sinal será decomposto.

O primeiro passo é encontrar o elemento $\phi_k$ do dicionário que melhor aproxima o sinal a ser codificado, ou seja, escolher o valor $k_0$ que maximiza o produto interno $<\phi_{k_0}, \mathrm{x}>$. A seguir, obtém-se o resíduo $\mathrm{R}^1\mathrm{x}$ desta aproximação:

$$\begin{aligned} \mathrm{R}^0\mathrm{x} &= \mathrm{x} \\ \mathrm{R}^1\mathrm{x} &= \mathrm{R}^0\mathrm{x} - <\phi_{k_0}, \mathrm{x}>\phi_{k_0} \end{aligned} \tag{3.1}$$

Este resíduo é, então, aproximado da mesma forma. Assim, no passo ***n*,** teremos:

$$\mathrm{R}^{\mathrm{n}+1}\mathrm{x} = \mathrm{R}^{\mathrm{n}}\mathrm{x} - <\phi_{k_n}, \mathrm{x}>\phi_{k_n} \tag{3.2}$$

E, após *N* passos, o vetor **x** pode ser aproximado como:

$$\hat{\mathrm{x}} = \sum_{i=0}^{N-1} <\phi_{k_i}, \mathrm{x}>\phi_{k_i} \tag{3.3}$$

Como um *frame* pode ser redundante, ou seja, pode possuir mais vetores que o necessário para gerar o espaço ao qual pertence o sinal de entrada, em geral, nem todos os elementos do dicionário serão usados na expansão. Assim, deve-se especificar, além dos valores dos coeficientes transformados $<\phi_{k_i}, \mathrm{x}>$, o índice $k_i$ do elemento correspondente no dicionário.

## 3.3 Descrição do UMMP

Como dito anteriormente, o algoritmo UMMP é bastante semelhante ao MMP, descrito no capítulo anterior. Ele, no entanto, se baseia no *Matching Pursuits* (MP) ao descrever o sinal de entrada como uma combinação linear de vetores pertencentes a um dicionário redundante conhecido, que forma um *frame* do espaço ao qual pertence este sinal.

O algoritmo MP pode ser interpretado como uma aproximação sucessiva baseada em casamento aproximado de padrões. O UMMP estende este conceito para um casamento aproximado com escalas ao permitir que o dicionário contenha elementos de diferentes tamanhos.

Neste sentido, o algoritmo pode dividir o resíduo obtido em cada passo e executar uma nova aproximação para cada um dos segmentos de maneira independente. A decisão de se dividir ou não o resíduo deve ser baseada em algum critério de fidelidade, como, por exemplo, um valor máximo para a distorção. Além disso, a cada iteração, de maneira semelhante ao feito no MMP, o dicionário é incrementado com a concatenação de elementos já codificados.

Exemplificando, seja $R^0 = X$ o vetor de entrada de tamanho *N*. Como no MP, é encontrado o vetor $S_{k_0}$ pertencente ao dicionário *D* que melhor aproxima $R^0$ e calcula-se o resíduo $R^1 = R^0 - < S_{k_0}, R^0 > S_{k_0}$. A seguir, divide-se o resíduo $R^{1,0} = R^1$ em dois segmentos, $R^{1,1}$ de tamanho *p* e $R^{1,2}$ de tamanho $N - p$. Ressalta-se que o segundo sobrescrito se relaciona ao índice do segmento, enquanto o primeiro é o número de aproximações já feitas. Por meio de uma transformação de escala $T_p^N[\cdot]$, obtêm-se os elementos do dicionário *D* com dimensão *p*, que são usados para aproximar $R^{1,1}$ e para calcular o novo resíduo $R^{2,1} = R^{1,1} - < T_p^N[S_{k_{1,1}}], R^{1,1} > . T_p^N[S_{k_{1,1}}]$. De forma análoga, faz-se a aproximação do resíduo $R^{1,2}$. O processo é interrompido quando se consegue uma distorção menor que um valor alvo. Assim, para uma dada distorção-alvo $d^*$, tem-se uma árvore de segmentação associada, como aquela mostrada na Figura 3.1.

Podemos afirmar que o quadrado da norma de qualquer elemento da árvore da Figura 3.1 correspondente a um nó-folha, como, por exemplo, $\left\| R^{3,3} \right\|^2$ é menor ou igual à distorção-alvo vezes o tamanho do resíduo, já que, caso contrário, o resíduo $R^{3,3}$ seria segmentado e os novos resíduos, $R^{3,7}$ e $R^{3,8}$, seriam aproximados por elementos contraídos do dicionário.

A atualização do dicionário é feita de modo bastante semelhante ao MMP. Ao obtermos a aproximação de dois resíduos ($\hat{R}^{i+1,2j+1}$ e $\hat{R}^{i+1,2j+2}$), inserimos no dicionário a sua concatenação ($\hat{R}^{i+1,j}$). A diferença em relação ao procedimento adotado no MMP é que, agora, também incluímos no dicionário o elemento $\hat{R}^{i,j}$, resultado da adição de $\hat{R}^{i+1,j}$ à aproximação $< S_{k_{i,j}}, R^{i,j} >. S_{k_{i,j}}$ do nó-pai $R^{i,j}$. O procedimento é ilustrado na Figura 3.2.

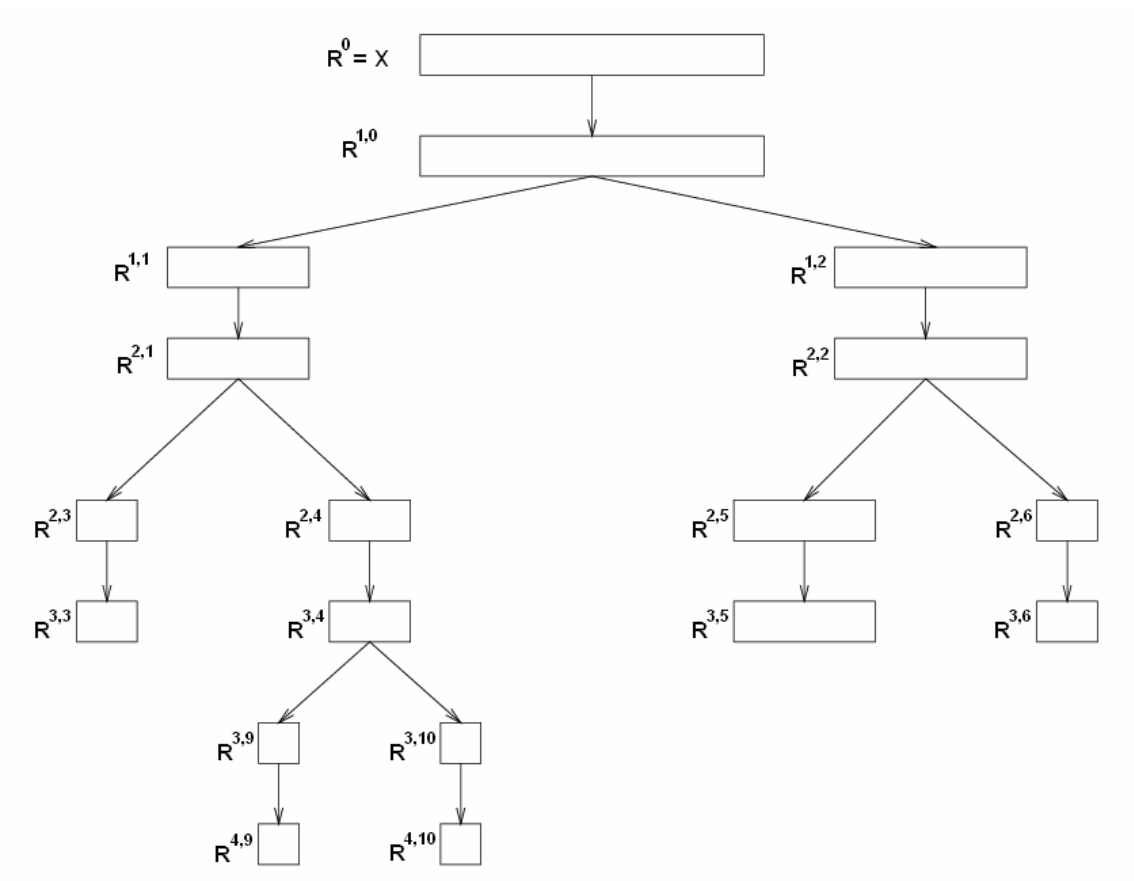


Figura 3.1: Exemplo de árvore de segmentação do UMMP.

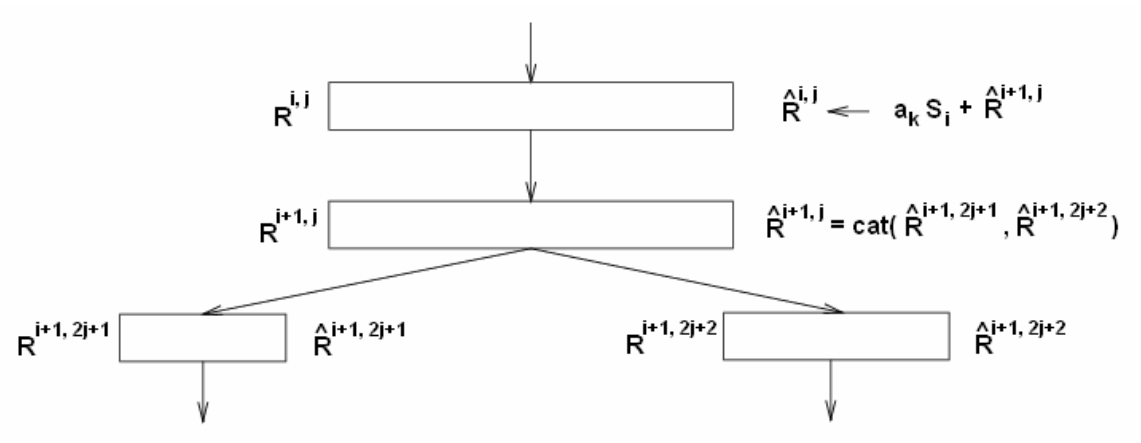


Figura 3.2: Atualização do dicionário no UMMP.

Segundo relatado em [5], simulações mostraram que há um considerável ganho de desempenho quando usamos um quantizador escalar $Q = \{1\}$ com apenas um nível para codificação do coeficiente associado a cada elemento do dicionário, ou seja, quando dispensamos a necessidade de codificação e armazenamento deste valor. Além disso, também devemos dispensar a codificação do valor de *p*, simplesmente dividindo cada resíduo ao meio, ou seja, usando sempre $p = N/2$.

A descrição completa da implementação em pseudocódigo do algoritmo UMMP encontra-se no Apêndice C.

## 3.4 Resultados experimentais

Foram feitas simulações em computador do algoritmo UMMP originalmente proposto em [5]. Como o interesse, neste trabalho, é o ramo de compressão de imagens, foi implementada uma extensão bidimensional do algoritmo mostrado no Apêndice C, o 2D-UMMP, também descrito em [5]. Os resultados obtidos são comparados com o MMP-RDI [6] e o SPIHT [4].

A extensão para o 2D-UMMP é análoga àquela descrita no caso do MMP. Por simplicidade, o termo UMMP, daqui por diante, também será utilizado para designar o algoritmo bidimensional.

A implementação feita neste trabalho contemplou os seguintes aspectos:

- tanto para o UMMP quanto para o MMP-RDI, a imagem foi dividida em blocos de tamanho 8x8, varridos da esquerda para a direita e de cima para baixo, ou seja $M = 2^K = N = 2^L = 8$;
- a segmentação de blocos *m x n* foi feita de forma que, se *m = n*, a partição é vertical, ou seja os blocos-filhos serão de dimensão *m x n/2*, e se *m > n*, a partição é horizontal, ou seja os blocos-filhos terão a dimensão *m/2 x n*;
- foram implementados K+L+1 = 3+3+1 = 7 dicionários, de dimensões 1x1, 2x1, 2x2, 4x2, 4x4, 8x4 e 8x8, de maneira análoga ao feito no MMP. O dicionário $D^{(m,n)}$ possui elementos de dimensão *m x n*;
- o dicionário inicial utilizado possui 513 elementos, sendo composto por todos os níveis de cinza possíveis da imagem e seus simétricos, ou seja, $D^{(1,1)} = \{-256, -255,..., -1, 0, 1,..., 255, 256\}$. Os dicionários iniciais nas outras escalas foram obtidos a partir de $D^{(1,1)}$ pela aplicação de transformações de escala;
- o tamanho de cada dicionário foi limitado a 8.192 elementos. Quando este limite fosse atingido, a inserção de um novo elemento seria feita com a exclusão do elemento menos utilizado dentre os 50% mais antigos;
- a transformação de escala utilizada foi a mesma descrita na seção 2.2 pelas Equações 2.5, 2.6, 2.7, 2.8, 2.9 e 2.10; e
- os índices dos elementos dos dicionários e os *flags* descritores da árvore de segmentação foram codificados por um codificador aritmético adaptativo com modelos independentes para cada uma das escalas.

Com isso, considerando uma dada distorção-alvo $d^*$, o algoritmo foi implementado da seguinte forma:

**Procedimento** $\hat{R}^{i,j} = \text{codifica}\left(R^{i,j}, d^*\right)$

**Passo 1:** Procura, no dicionário $D^{(m,n)}$, o elemento $S_{k_{i,j}}^{m,n}$ que minimiza a distorção $\left\| R^{i,j} - S_{k_{i,j}}^{m,n} \right\|$ e faz $\hat{R}^{i,j} = S_{k_{i,j}}^{m,n}$.

**Passo 2:** Codifica o índice $k_{i,j}$.

**Passo 3:** Se $m = n = 1$, retorna $\hat{R}^{i,j}$.

Senão, vai para o **Passo 4.**

**Passo 4:** Se $\left\| R^{i,j} - \hat{R}^{i,j} \right\|^2 \leq m.n.d^*$, codifica o *flag* '1' e retorna $\hat{R}^{i,j}$.

Senão, vai para o **Passo 5.**

**Passo 5:** Calcula o resíduo $R^{i+1,j} = R^{i,j} - \hat{R}^{i,j}$ e codifica o *flag* '0'.

**Passo 6:** Se $m > n$, divide $R^{i+1,j}$ em $\begin{pmatrix} R^{i+1,2j+1} \\ R^{i+1,2j+2} \end{pmatrix}$, onde $R^{i+1,2j+1,}$ e $R^{i+1,2j+2}$ são *m/2 x n*.

Senão, divide $R^{i+1,j}$ em $\begin{pmatrix} R^{i+1,2j+1} & R^{i+1,2j+2} \end{pmatrix}$, onde $R^{i+1,2j+1,}$ e $R^{i+1,2j+2}$ são *m x n/2*.

**Passo 7:** Computa $\hat{R}^{i+1,2j+1} = \text{codifica}\left(R^{i+1,2j+1}, d^*\right)$.

**Passo 8:** Computa $\hat{R}^{i+1,2j+2} = \text{codifica}\left(R^{i+1,2j+2}, d^*\right)$.

**Passo 9:** Se $m > n$, faz $\hat{R}^{i+1,j} = \begin{pmatrix} \hat{R}^{i+1,2j+1} \\ \hat{R}^{i+1,2j+2} \end{pmatrix}$.

Senão, faz $\hat{R}^{i+1,j} = \begin{pmatrix} R^{i+1,2j+1} & R^{i+1,2j+2} \end{pmatrix}$.

**Passo 10:** Atualiza $\hat{R}^{i,j} \leftarrow \hat{R}^{i,j} + \hat{R}^{i+1,j}$.

**Passo 11:** Atualiza os dicionários de todas as escalas com $\hat{R}^{i,j}$ e com $\hat{R}^{i+1,j}$, ou seja:

Para *n = 0, 1, ..., 6*,

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

$D^{(p,q)} = D^{(p,q)} \cup T_{m,n}^{p,q}[\hat{R}^{i,j}] \cup T_{m,n}^{p,q}[\hat{R}^{i+1,j}]$.

**Passo 12:** Retorna $\hat{R}^{i,j}$.

As imagens comprimidas nesta simulação foram *aerial, baboon, barbara, bridge, f-16, gold, lena, pp1205 e pp1209*, todas de tamanho 512x512 *pixels*. As sete primeiras imagens, mostradas no Apêndice E, foram obtidas no *site http://sipi.usc.edu/ services/database/Database.html*, e as duas últimas, mostradas nas Figuras E.8 e E.9, foram digitalizadas do IEEE *Transactions on Image Processing*, volume 9, número 7, de julho de 2000, páginas 1.205 e 1.209, respectivamente.

Nas Figuras 3.3 a 3.11, encontram-se os gráficos obtidos nas simulações feitas em computador.

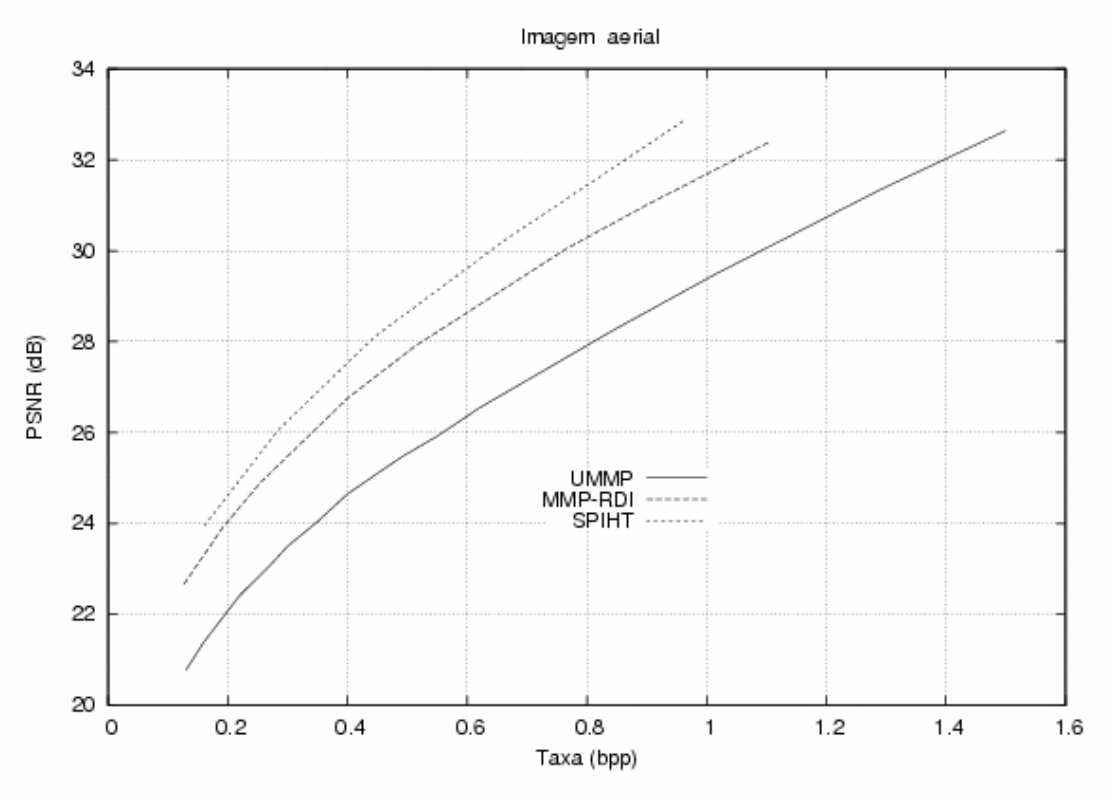


Figura 3.3: Taxa-distorção para *aerial* comprimida pelo UMMP.

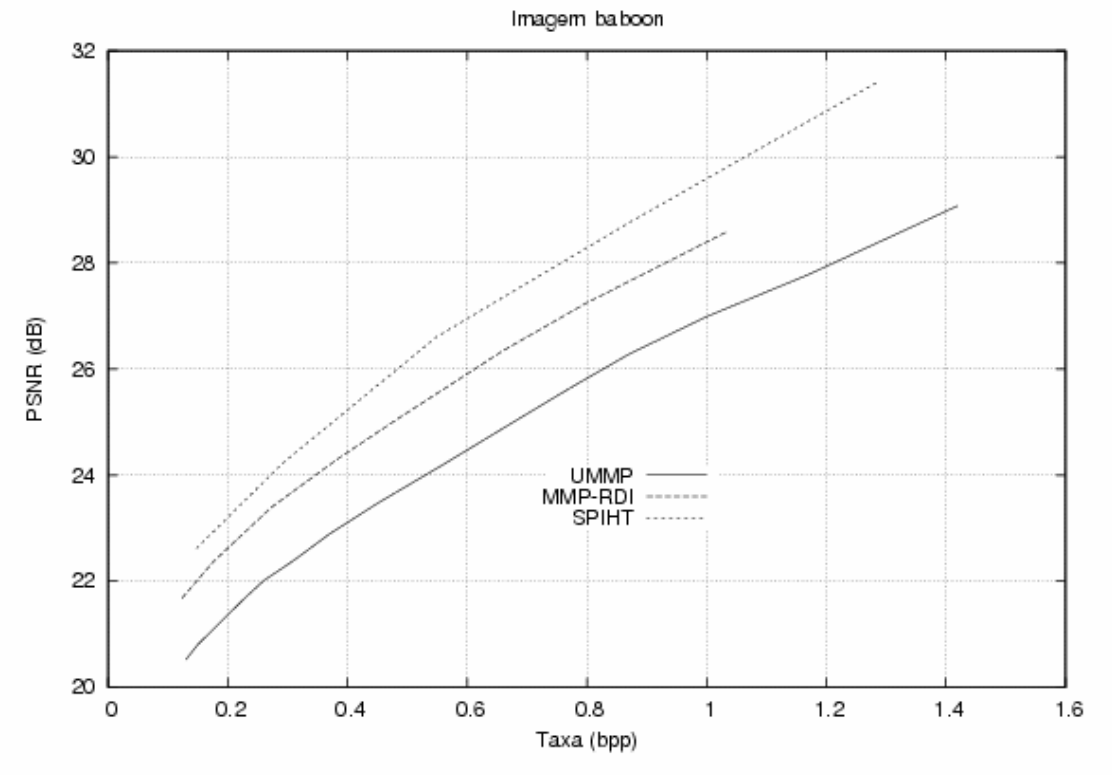


Figura 3.4: Taxa-distorção para *baboon* comprimida pelo UMMP.

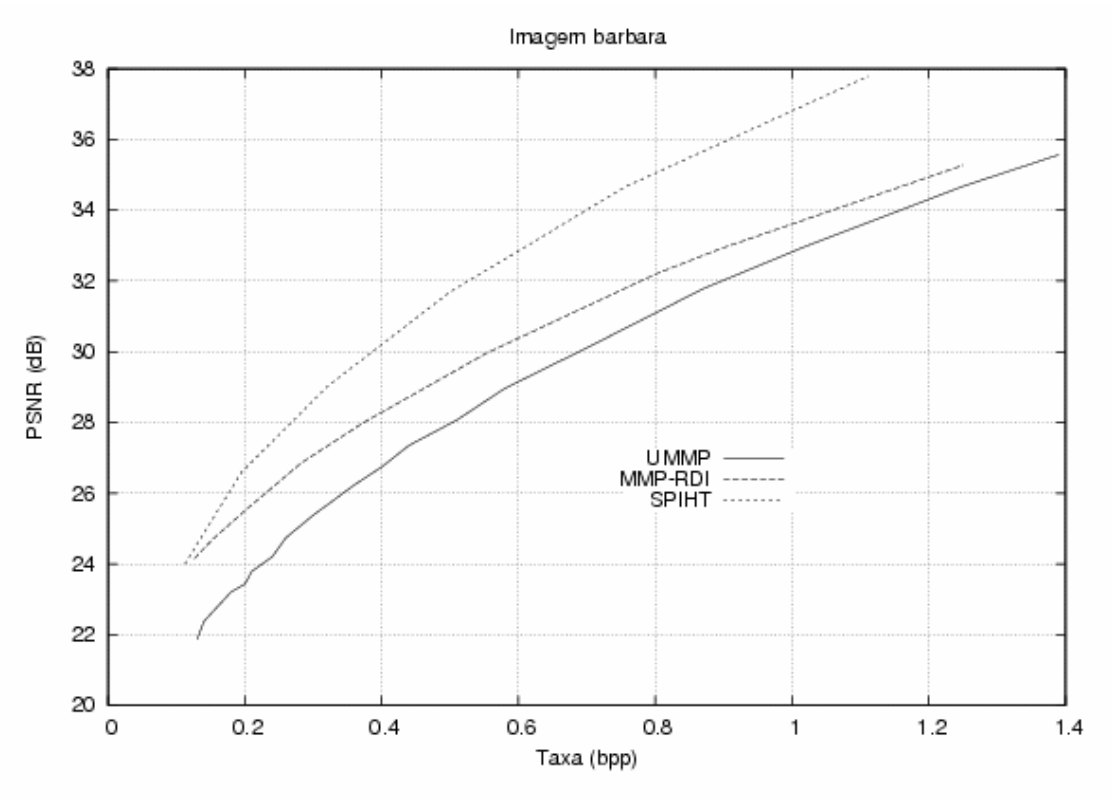


Figura 3.5: Taxa-distorção para *barbara* comprimida pelo UMMP.

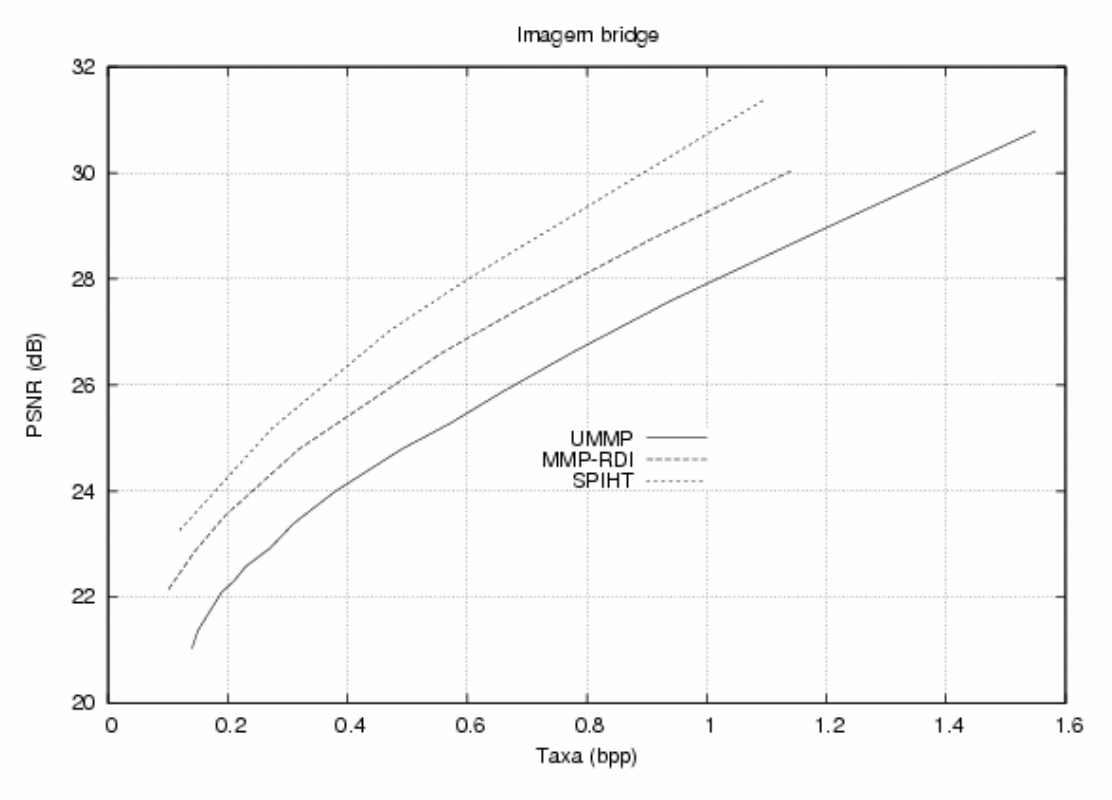


Figura 3.6: Taxa-distorção para *bridge* comprimida pelo UMMP.

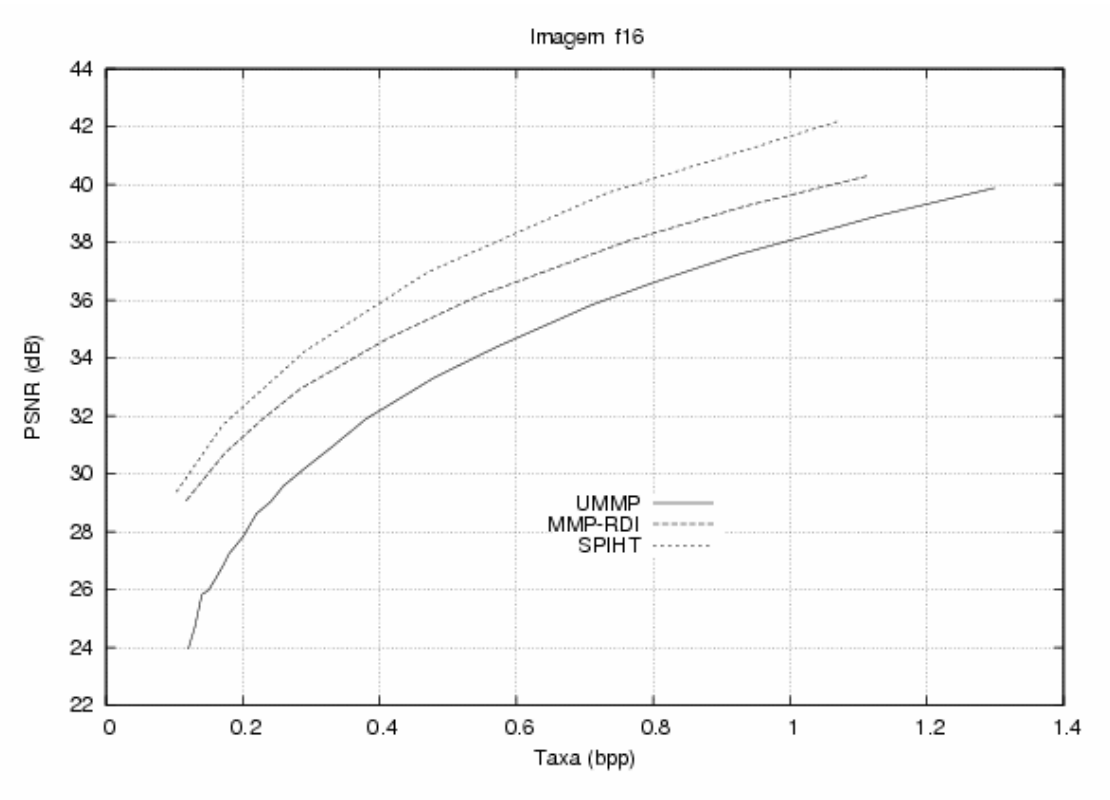


Figura 3.7: Taxa-distorção para *f-16* comprimida pelo UMMP.

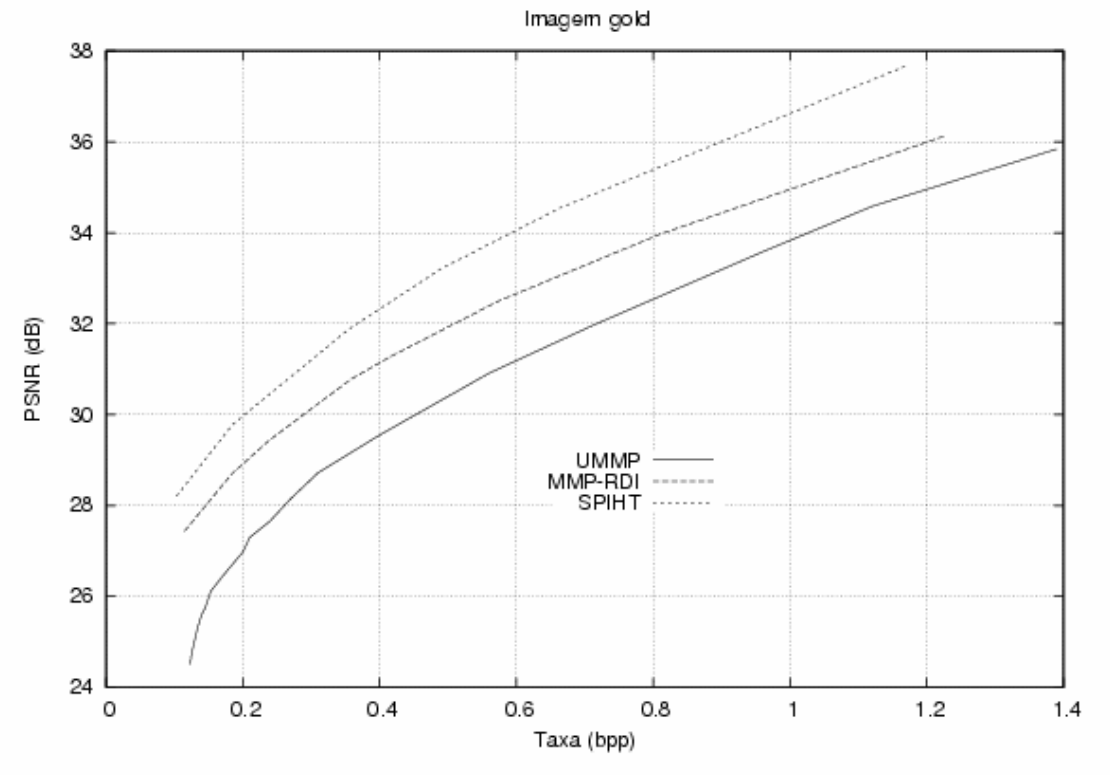


Figura 3.8: Taxa-distorção para *gold* comprimida pelo UMMP.

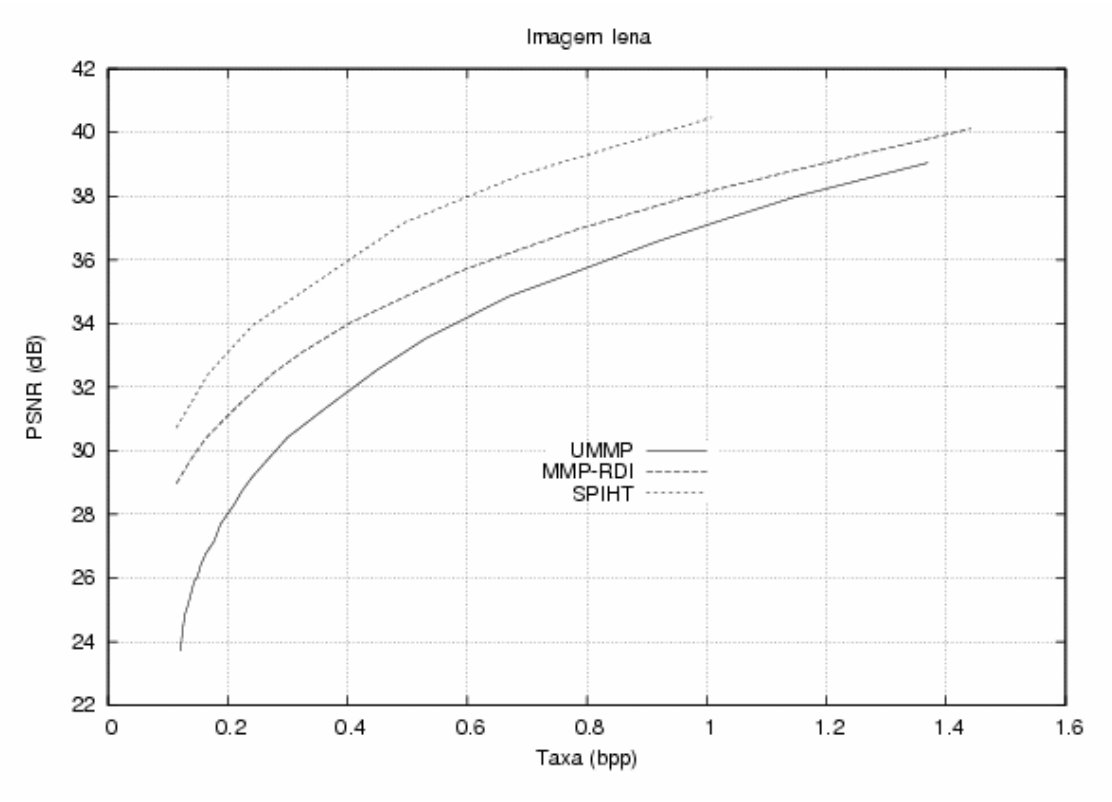


Figura 3.9: Taxa-distorção para *lena* comprimida pelo UMMP.

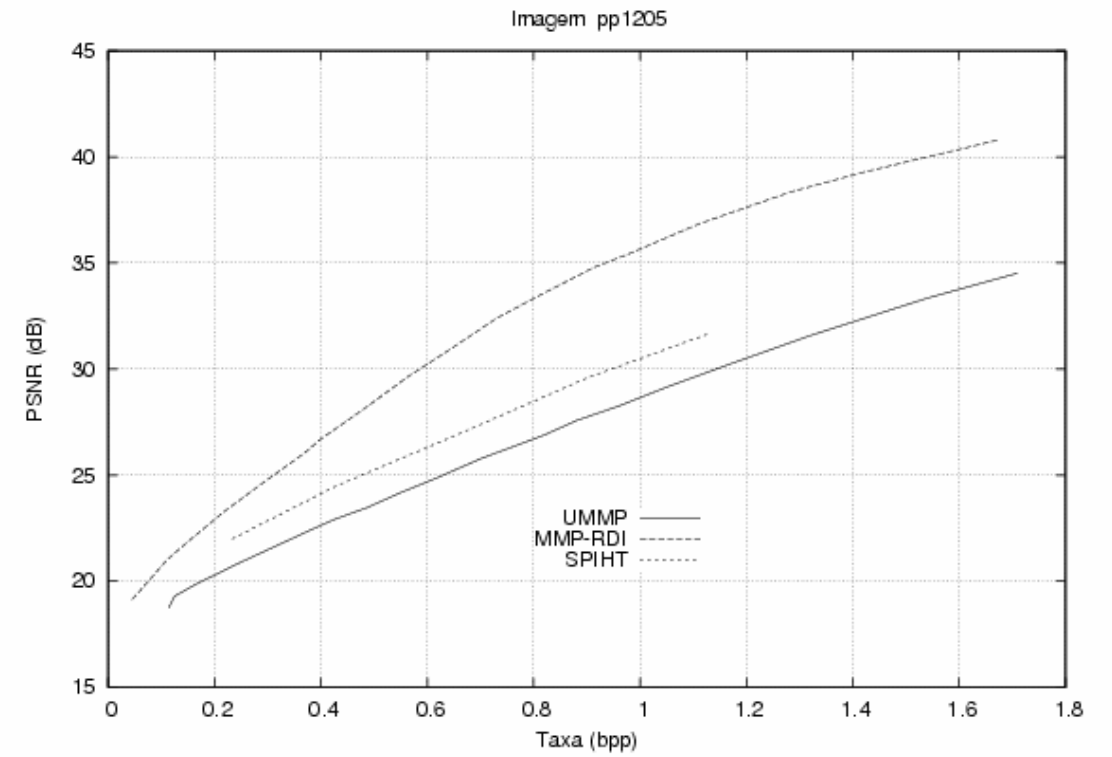


Figura 3.10: Taxa-distorção para *pp1205* comprimida pelo UMMP.

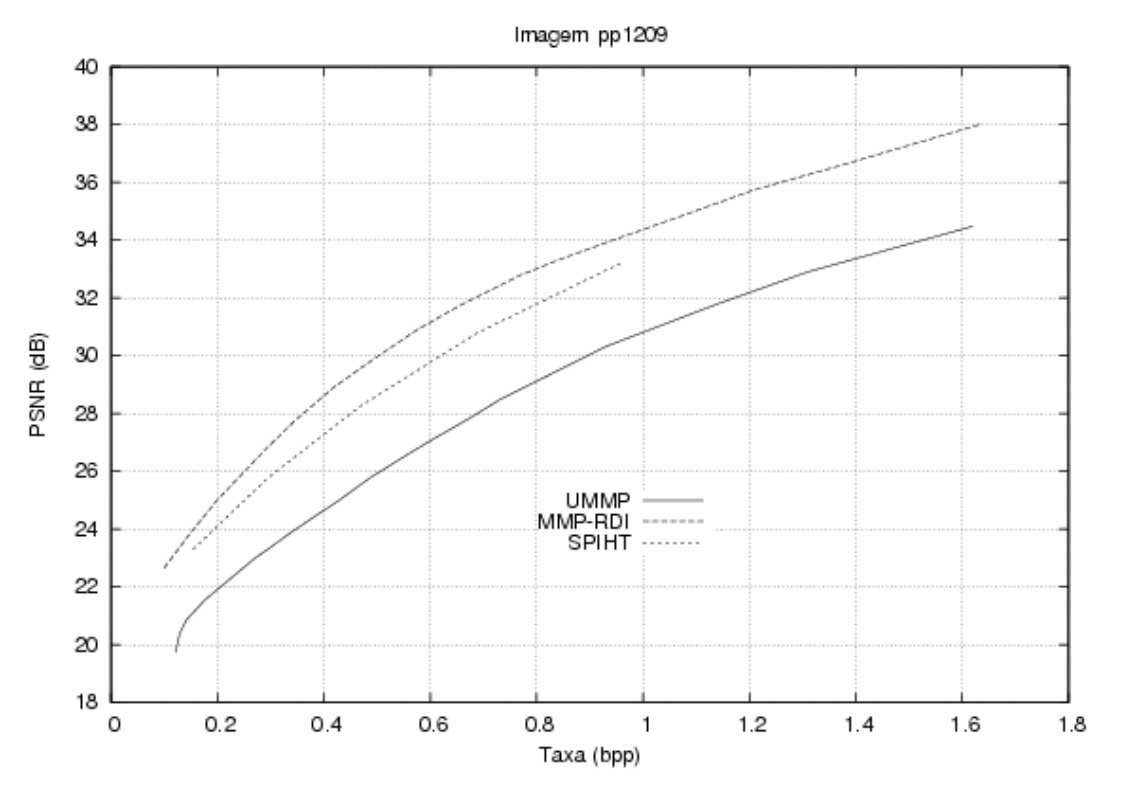


Figura 3.11: Taxa-distorção para *pp1209* comprimida pelo UMMP.

Podemos observar claramente, pelas curvas mostradas, que, para todas as imagens, o desempenho do UMMP foi bastante inferior ao do MMP-RDI, especialmente nas imagens mista e de texto. Para as imagens suaves, a perda ficou, na maioria dos casos, entre 1dB e 2dB.

Deve-se destacar, porém, que o MMP-RDI, como visto no capítulo anterior, é uma versão otimimizada, no sentido taxa-distorção, do MMP original. No trabalho original [5], pode-se observar que o UMMP não fica tão distante, em termos de desempenho, em relação ao MMP sem otimização. A Figura 3.12 reproduz as curvas obtidas em [5] para a imagem *lena*.

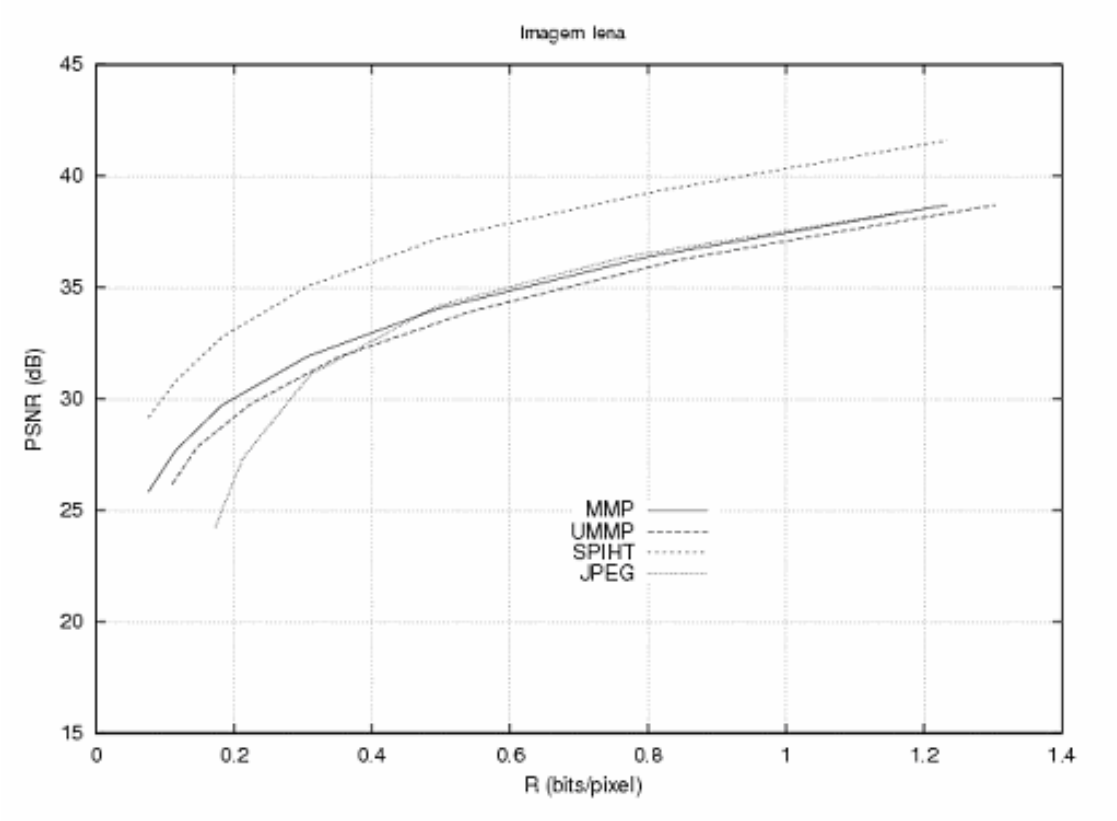


Figura 3.12: Taxa-distorção para *lena* comprimida pelo UMMP.

No próximo capítulo, descrevemos diversas tentativas de melhoria de desempenho do UMMP. Incorporamos ao algoritmo original o conceito de otimização taxa-distorção, introduzindo o algoritmo UMMP-RD, de modo semelhante ao desenvolvido para o MMP.

# Capítulo 4

# Otimização taxa-distorção no UMMP

A partir deste capítulo, descreveremos as três idéias principais que nortearam o desenvolvimento deste trabalho. Primeiramente, buscamos a implementação da otimização taxa-distorção no algoritmo UMMP, proposto em [5] e descrito no capítulo anterior. A abordagem é similar àquela descrita no Capítulo 2 para o caso do MMP.

## 4.1 Introdução

O algoritmo UMMP original, descrito em [5], se baseia em um critério local para decisão acerca da partição ou não de blocos de resíduos. Mais especificamente, como visto no capítulo anterior, se a norma ao quadrado de determinado resíduo é menor que dada distorção-alvo vezes a dimensão do bloco, ele não é dividido e a codificação se encerra. Percebe-se que este critério, além de ser local, não contempla o efeito do aumento do número de bits necessário à codificação que uma partição de blocos gera.

Assim, como feito em [5, 6] para o MMP, impõe-se necessária a tentativa de melhoria do desempenho do UMMP por intermédio da incorporação ao seu algoritmo de uma otimização taxa-distorção da árvore de segmentação.

Isto é feito a seguir ao propormos o algoritmo UMMP-RD1, primeira versão do UMMP otimizado. Mais à frente neste capítulo, introduzimos o UMMP-RD2 e o UMMP-RD3, que incorporam à primeira versão a idéia da segmentação dos dicionários no espaço de códigos, como feito ao MMP em [12].

## 4.2 UMMP-RD1

De forma bastante semelhante ao caso do MMP, visto no Capítulo 2, a idéia é a utilização do custo *lagrangeano* [7, 8] para a decisão de partição de um bloco ao invés da simples comparação do resíduo a uma distorção-alvo. Assim, calculam-se os custos de um nó da árvore de segmentação e de seus dois nós-filhos. Se o custo do nó-pai for menor que a soma dos custos dos nós-filhos, estes, e suas respectivas sub-árvores, são podados. Caso contrário, eles são mantidos e o resultado da soma passa a ser o novo custo do nó-pai. O custo de um nó $n_l$ é assim calculado:

$$J_{n_l} = D_{n_l} + \lambda . R_{n_l} \tag{4.1}$$

onde $D_{n_l}$ é a distorção devida à aproximação do resíduo, $R_{n_l}$ é o número de bits necessário para a representação do índice do elemento do dicionário escolhido, e λ é o multiplicador de *Lagrange*.

Na implementação feita neste trabalho, optou-se por manter o dicionário, e também os modelos para seus índices e para as *flags* de segmentação, fixos durante a fase de otimização. Isto se deveu ao considerável aumento de complexidade que qualquer tentativa de espelhar a codificação durante a otimização, como no MMP-RD [5] e no MMP-RDI [6], traz ao algoritmo, aliado ao fato de que a perda de desempenho devida à simplificação é reduzida. Tal perda é pequena porque, neste caso, a dependência, descrita no Capítulo 2, existente entre a decisão de poda de um nó da árvore e a codificação dos nós à sua direita perde importância mais rapidamente ainda em comparação ao MMP. Isto ocorre porque, como a cada aproximação de dois nós da árvore dois elementos são inseridos no dicionário, contra apenas um no MMP, o dicionário, no UMMP, atinge um tamanho grande de forma mais veloz. Assim, o dicionário passa a ter uma grande diversidade de elementos rapidamente.

A rotina de otimização, a ser executada antes da codificação, gera a árvore de segmentação até o último nível, simultaneamente ao cálculo do custo de cada nó. Na seqüência, compara a soma dos custos dos nós-folhas (1x1) com o custo do nó-pai (2x1), executando a poda se este for maior que a soma daqueles. Este mecanismo é repetido na mesma seqüência da codificação até que toda a árvore seja percorrida.

Para melhor refletir o que ocorre durante a codificação, os custos para se codificar os *flags* descritores da árvore de segmentação também foram considerados. Assim, o custo total para representação de um dado nó como nó-folha é dado por:

$$J_l = D_{n_l} + \lambda.R_{n_l} + \lambda.R_{1l} \tag{4.2}$$

onde $J_l$ é o custo *lagrangeano* do nó, $D_{n_l}$ é a distorção devida à aproximação de $X_l^{m,n}$ por $S_i^{m,n}$, $R_{n_l}$ é o número de bits necessário para a representação do índice *i* escolhido, e $R_{1l}$ é o número de bits necessário para a codificação do *flag* '1', que indica que o nó $n_l$ é um nó-folha.

Já o custo de representação dos nós-filhos ($n_{2l+1}$ e $n_{2l+2}$) como nós-folhas, deve considerar, além dos *flags* '1', como anteriormente, o *flag* '0' que indica a partição do nó $n_l$. Assim, temos a seguinte relação:

$$J_{2l+1,2l+2} = J_{2l+1} + J_{2l+2} + \lambda.R_{0l} \tag{4.3}$$

onde $J_{2l+1,2l+2}$ é o custo *lagrangeano* total para a representação de $n_{2l+1}$ e $n_{2l+2}$, $J_{2l+1}$ e $J_{2l+2}$ são os custos *lagrangeanos* para a representação de $n_{2l+1}$ e $n_{2l+2}$ como nós-folhas, obtidos por meio da Equação 4.2, e $R_{0l}$ é o número de bits necessário para a codificação do *flag* '0'.

O algoritmo de otimização foi implementado da seguinte forma:

**Procedimento** $\left\{\hat{X}^{i,j}, T\right\} = \text{otimiza}\left(X^{i,j}, T_0, \eta_0\right)$

**Passo 1:** Faz $T = T_0$.

**Passo 2:** Procura, no dicionário $D^{m,n}$, onde $m = 2^{\left\lfloor \frac{\eta_0+1}{2} \right\rfloor}$ e $n = 2^{\left\lfloor \frac{\eta_0}{2} \right\rfloor}$, o elemento $S_{k_j}^{m,n}$ que representa $X^{i,j}$ com menor custo $J_j$, armazenando $k_j$ e $J_j$. O elemento $S_{k_j}^{m,n}$ é armazenado em $\hat{X}^{i,j}$.

**Passo 3:** Se a escala atual for $\eta_0 == 0$, ou seja, 1x1 ($m == 1$ e $n == 1$), retorna $\hat{X}^{i,j}$ e *T*. Senão, vai para o **Passo 4.**

**Passo 4:** Acrescenta, ao custo $J_j$ calculado, o valor $\lambda.R_{1j}$, para representar o custo total do nó-folha, onde $R_{1j}$ é a taxa necessária para codificação do *flag* '1'.

**Passo 5:** Calcula e armazena, separadamente, o valor $\lambda.R_{0j}$, que posteriormente complementará o custo dos nós-filhos, onde $R_{0j}$ é a taxa necessária para codificação do *flag* '0'.

**Passo 6:** Calcula o resíduo $X^{i+1,j} = X^{i,j} - \hat{X}^{i,j}$.

**Passo 7:** Se $m > n$, divide $X^{i+1,j}$ em $\begin{pmatrix} X^{i+1,2j+1} \\ X^{i+1,2j+2} \end{pmatrix}$, onde $X^{i+1,2j+1,}$ e $X^{i+1,2j+2}$ são *m/2 x n*.

Senão, divide $X^{i+1,j}$ em $\begin{pmatrix} X^{i+1,2j+1} & X^{i+1,2j+2} \end{pmatrix}$, onde $X^{i+1,2j+1,}$ e $X^{i+1,2j+2}$ são *m x n/2*.

**Passo 8:** Computa $\left\{\hat{X}^{i+1,2j+1}, T_1\right\} = \text{otimiza}\left(X^{i+1,2j+1}, T, \eta_0 - 1\right)$.

**Passo 9:** Computa $\left\{\hat{X}^{i+1,2j+2}, T_2\right\} = \text{otimiza}\left(X^{i+1,2j+2}, T, \eta_0 - 1\right)$.

**Passo 10:** Atualiza a árvore de segmentação *T* de acordo com $T_1$ e $T_2$.

**Passo 11:** Se $J_j \leq J_{2j+1} + J_{2j+2} + \lambda.R_{0j}$, vai para o **Passo 12**.

Senão, vai para o **Passo 13**.

**Passo 12:** Indica, em *T*, que as sub-árvores $S(n_{2j+1})$ e $S(n_{2j+2})$ foram podadas e retorna $\hat{X}^{i,j}$ e *T*.

**Passo 13:** Se $m > n$, faz $\hat{X}^{i+1,j} = \begin{pmatrix} \hat{X}^{i+1,2j+1} \\ \hat{X}^{i+1,2j+2} \end{pmatrix}$.

Senão, faz $\hat{X}^{i+1,j} = \begin{pmatrix} \hat{X}^{i+1,2j+1} & \hat{X}^{i+1,2j+2} \end{pmatrix}$.

**Passo 14:** Atualiza $\hat{X}^{i,j} \leftarrow \hat{X}^{i,j} + \hat{X}^{i+1,j}$.

**Passo 15:** Faz $J_j = J_{2j+1} + J_{2j+2} + \lambda.R_{0j}$.

**Passo 16:** Retorna $\hat{X}^{i,j}$ e *T*.

A rotina de codificação é bastante semelhante àquela descrita no Capítulo 3 para o caso do UMMP sem otimização, bastando-se alterar a condição de partição do bloco, que, agora, é baseada na árvore de segmentação *T* gerada na rotina de otimização. Neste caso, o algoritmo foi implementado da seguinte forma:

**Procedimento** $\hat{X}^{i,j} = \text{codifica}\left(X^{i,j}, \eta_0, T\right)$

**Passo 1:** Procura, no dicionário $D^{m,n}$, onde $m = 2^{\left\lfloor \frac{\eta_0+1}{2} \right\rfloor}$ e $n = 2^{\left\lfloor \frac{\eta_0}{2} \right\rfloor}$, o elemento $S_{k_j}^{m,n}$ que representa $X^{i,j}$ com menor custo, fazendo $\hat{X}^{i,j} = S_{k_j}^{m,n}$.

**Passo 2:** Codifica o índice $k_{i,j}$.

**Passo 3:** Se a escala atual for $\eta_0 == 0$, ou seja, 1x1 ($m == 1$ e $n == 1$), retorna $\hat{X}^{i,j}$.

Senão, vai para o **Passo 4.**

**Passo 4:** Se $T_j == 1$, codifica o *flag* '1' e retorna $\hat{X}^{i,j}$.

Senão, vai para o **Passo 5.**

**Passo 5:** Calcula o resíduo $X^{i+1,j} = X^{i,j} - \hat{X}^{i,j}$ e codifica o *flag* '0'.

**Passo 6:** Se $m > n$, divide $X^{i+1,j}$ em $\begin{pmatrix} X^{i+1,2j+1} \\ X^{i+1,2j+2} \end{pmatrix}$, onde $X^{i+1,2j+1,}$ e $X^{i+1,2j+2}$ são *m/2 x n*.

Senão, divide $X^{i+1,j}$ em $\begin{pmatrix} X^{i+1,2j+1} & X^{i+1,2j+2} \end{pmatrix}$, onde $X^{i+1,2j+1,}$ e $X^{i+1,2j+2}$ são *m x n/2*.

**Passo 7:** Computa $\hat{X}^{i+1,2j+1} = \text{codifica}\left(X^{i+1,2j+1}, \eta_0, T\right)$.

**Passo 8:** Computa $\hat{X}^{i+1,2j+2} = \text{codifica}\left(X^{i+1,2j+2}, \eta_0, T\right)$.

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

Senão, faz $\hat{X}^{i+1,j} = \begin{pmatrix} \hat{X}^{i+1,2j+1} & \hat{X}^{i+1,2j+2} \end{pmatrix}$.

**Passo 10:** Atualiza $\hat{X}^{i,j} \leftarrow \hat{X}^{i,j} + \hat{X}^{i+1,j}$.

**Passo 11:** Atualiza os dicionários de todas as escalas com $\hat{X}^{i,j}$ e com $\hat{X}^{i+1,j}$, ou seja:

Para *n = 0, 1, ..., 6,*

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

$D^{(p,q)} = D^{(p,q)} \cup T_{m,n}^{p,q}[\hat{X}^{i,j}] \cup T_{m,n}^{p,q}[\hat{X}^{i+1,j}]$.

**Passo 12:** Retorna $\hat{X}^{i,j}$.

A implementação do UMMP-RD1 feita neste trabalho contemplou os mesmos aspectos citados na seção 3.4 para o caso do UMMP, com exceção do tamanho dos dicionários, que foi estendido para 81.920 elementos, ressaltando-se que, em nenhum caso testado, este limite foi atingido. Foram codificadas as mesmas imagens do capítulo anterior. As Figuras 4.1 e 4.2 ilustram o resultado obtido para as imagens *lena* e *pp1205*, respectivamente. Os resultados para as outras imagens serão mostrados mais à frente, na Seção 4.5, de forma consolidada com as outras versões implementadas.

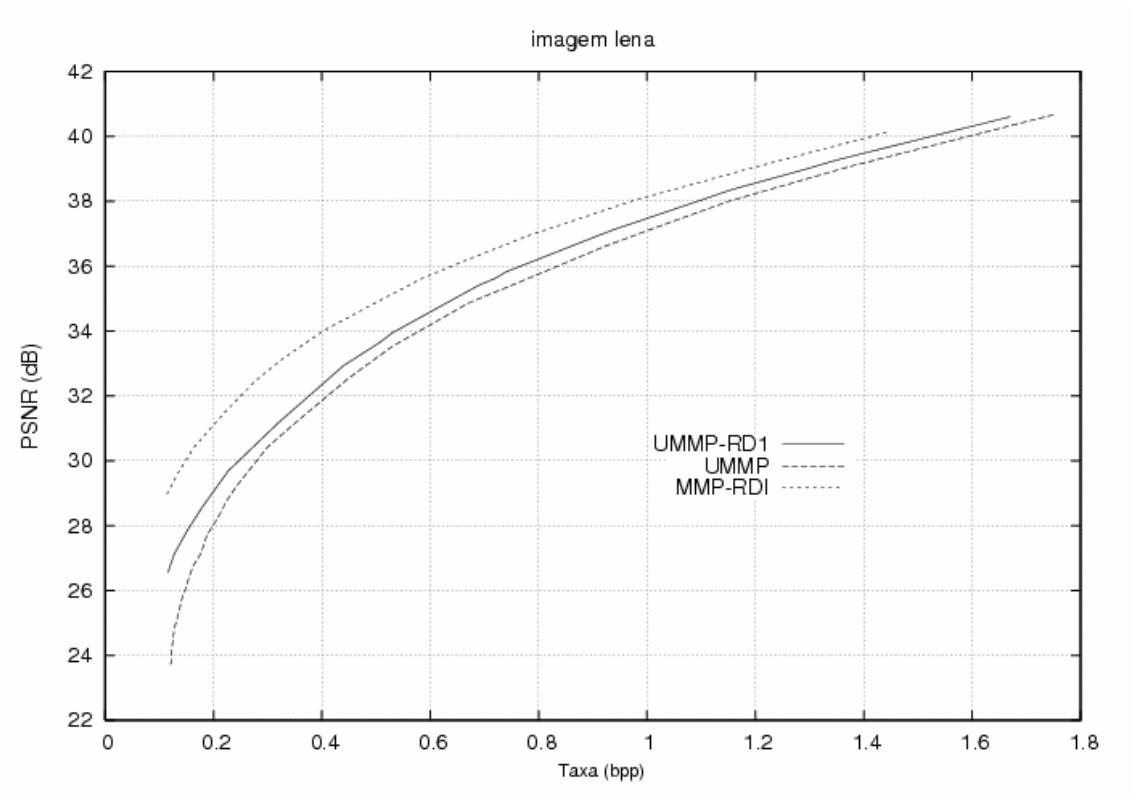


Figura 4.1: Taxa-distorção para *lena* comprimida pelo UMMP-RD1.

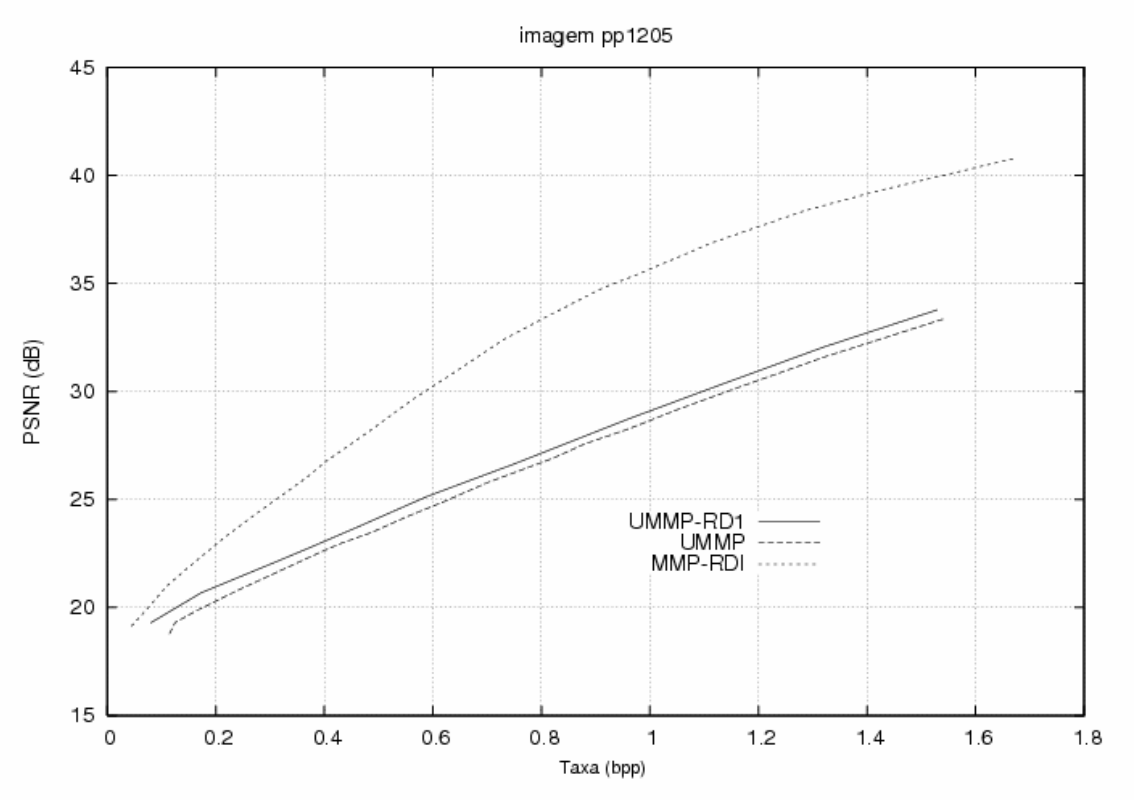


Figura 4.2: Taxa-distorção para *pp1205* comprimida pelo UMMP-RD1.

## 4.3 UMMP-RD2

Nesta seção, descrevemos o UMMP-RD2, uma tentativa de incorporar ao UMMP otimizado a idéia de segmentação dos dicionários no espaço de códigos, como feito ao MMP-RDI em [12]. A expectativa de ganho de desempenho se apóia na melhoria dos resultados obtidos em [12] para o MMP-RDI.

A idéia de segmentação no espaço de códigos advém da observação de que cada elemento inserido em um dicionário é proveniente de uma determinada escala. Por exemplo, ao obtermos uma aproximação de um bloco 4x4 da imagem de entrada, atualizamos todos os dicionários com versões dilatadas ou contraídas desta aproximação. Assim, pode ser vantajoso utilizarmos esta informação para melhorar o desempenho do algoritmo.

Com isso, os dicionários são subdivididos de forma que cada subdicionário contenha os elementos provenientes de uma determinada escala, como mostrado na Figura 4.3. No exemplo citado, as versões dilatadas ou contraídas do bloco 4x4 serão inseridas nos subdicionários 4x4 de cada um dos dicionários.

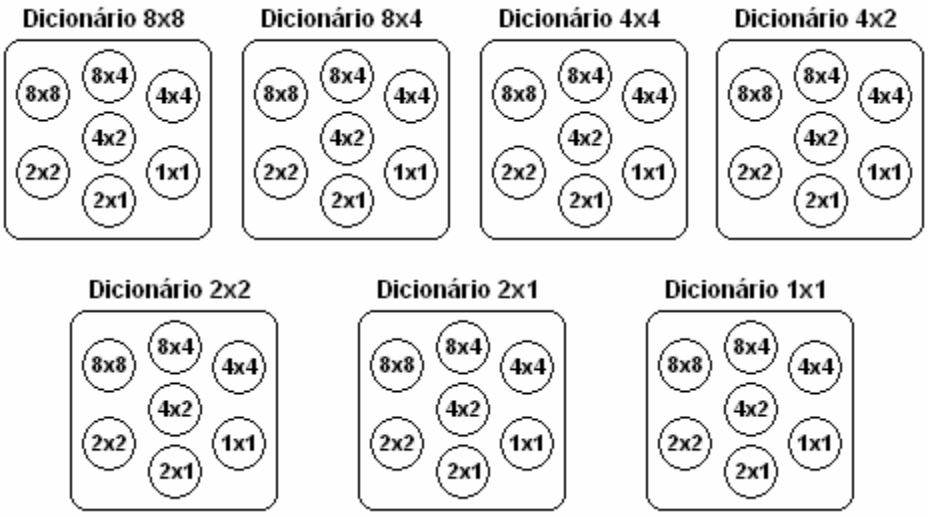


Figura 4.3: Esquema de dicionários no UMMP-RD2.

Com essa estrutura para os dicionários, a codificação do elemento a ser usado para aproximar um bloco de entrada será feita em duas etapas. Primeiramente, devemos indicar a qual subdicionário o elemento pertence, e, na seqüência, seu índice dentro do subdicionário.

Na fase de otimização, evidentemente, esta nova forma de codificação para um elemento deve ser considerada. Assim, o custo total de representação de um dado nó como nó-folha, obtido pela Equação 4.2, deve considerar o número total de bits necessários à codificação, dado por:

$$R_{n_l} = R_{k_s} + R_{k_e} \tag{4.4}$$

onde $R_{k_s}$ é o número de bits necessário para a codificação do índice $k_s$ do subdicionário escolhido, e $R_{k_e}$ é o número de bits necessário para a codificação índice $k_e$ do elemento dentro do seu subdicionário.

Os algoritmos de otimização e de codificação foram implementados de forma bastante semelhante ao UMMP-RD1:

**Procedimento** $\left\{\hat{X}^{i,j}, T\right\} = \text{otimiza}\left(X^{i,j}, T_0, \eta_0\right)$

**Passo 1:** Faz $T = T_0$.

**Passo 2:** Procura, no dicionário $D^{m,n}$, onde $m = 2^{\left\lfloor \frac{\eta_0+1}{2} \right\rfloor}$ e $n = 2^{\left\lfloor \frac{\eta_0}{2} \right\rfloor}$, o elemento $S^{m,n}_{k_s,k_e}$ que representa $X^{i,j}$ com menor custo $J_j$, armazenando $k_s$, $k_e$ e $J_j$. O elemento $S^{m,n}_{k_s,k_e}$ é armazenado em $\hat{X}^{i,j}$.

**Passo 3:** Se a escala atual for $\eta_0 == 0$, ou seja, 1x1 ($m == 1$ e $n == 1$), retorna $\hat{X}^{i,j}$ e *T*. Senão, vai para o **Passo 4.**

**Passo 4:** Acrescenta, ao custo $J_j$ calculado, o valor $\lambda.R_{1j}$, para representar o custo total do nó-folha, onde $R_{1j}$ é a taxa necessária para codificação do *flag* '1'.

**Passo 5:** Calcula e armazena, separadamente, o valor $\lambda.R_{0j}$, que posteriormente complementará o custo dos nós-filhos, onde $R_{0j}$ é a taxa necessária para codificação do *flag* '0'.

**Passo 6:** Calcula o resíduo $X^{i+1,j} = X^{i,j} - \hat{X}^{i,j}$.

**Passo 7:** Se $m > n$, divide $X^{i+1,j}$ em $\begin{pmatrix} X^{i+1,2j+1} \\ X^{i+1,2j+2} \end{pmatrix}$, onde $X^{i+1,2j+1,}$ e $X^{i+1,2j+2}$ são *m/2 x n*.

Senão, divide $X^{i+1,j}$ em $\left(X^{i+1,2j+1} \quad X^{i+1,2j+2}\right)$, onde $X^{i+1,2j+1,}$ e $X^{i+1,2j+2}$ são *m x n/2*.

**Passo 8:** Computa $\left\{\hat{X}^{i+1,2j+1}, T_1\right\} = \text{otimiza}\left(X^{i+1,2j+1}, T, \eta_0 - 1\right)$.

**Passo 9:** Computa $\left\{\hat{X}^{i+1,2j+2}, T_2\right\} = \text{otimiza}\left(X^{i+1,2j+2}, T, \eta_0 - 1\right)$.

**Passo 10:** Atualiza a árvore de segmentação *T* de acordo com $T_1$ e $T_2$.

**Passo 11:** Se $J_j \leq J_{2j+1} + J_{2j+2} + \lambda.R_{0j}$, vai para o **Passo 12**.

Senão, vai para o **Passo 13**.

**Passo 12:** Indica, em *T*, que as sub-árvores $S\left(n_{2j+1}\right)$ e $S\left(n_{2j+2}\right)$ foram podadas e retorna $\hat{X}^{i,j}$ e *T*.

**Passo 13:** Se $m > n$, faz $\hat{X}^{i+1,j} = \begin{pmatrix} \hat{X}^{i+1,2j+1} \\ \hat{X}^{i+1,2j+2} \end{pmatrix}$.

Senão, faz $\hat{X}^{i+1,j} = \left(\hat{X}^{i+1,2j+1} \quad \hat{X}^{i+1,2j+2}\right)$.

**Passo 14:** Atualiza $\hat{X}^{i,j} \leftarrow \hat{X}^{i,j} + \hat{X}^{i+1,j}$.

**Passo 15:** Faz $J_j = J_{2j+1} + J_{2j+2} + \lambda.R_{0j}$.

**Passo 16:** Retorna $\hat{X}^{i,j}$ e *T*.

**Procedimento** $\hat{X}^{i,j} = \text{codifica}\left(X^{i,j}, \eta_0, T\right)$

**Passo 1:** Procura, no dicionário $D^{m,n}$, onde $m = 2^{\left\lfloor \frac{\eta_0+1}{2} \right\rfloor}$ e $n = 2^{\left\lfloor \frac{\eta_0}{2} \right\rfloor}$, o elemento $S^{m,n}_{k_s,k_e}$ que representa $X^{i,j}$ com menor custo, fazendo $\hat{X}^{i,j} = S^{m,n}_{k_s,k_e}$.

**Passo 2:** Codifica o índice $k_s$ do subdicionário escolhido.

**Passo 3:** Codifica o índice $k_e$ do elemento dentro do subdicionário.

**Passo 4:** Se a escala atual for $\eta_0 == 0$, ou seja, 1x1 ($m == 1$ e $n == 1$), retorna $\hat{X}^{i,j}$.

Senão, vai para o **Passo 5.**

**Passo 5:** Se $T_j == 1$, codifica o *flag* '1' e retorna $\hat{X}^{i,j}$.

Senão, vai para o **Passo 6.**

**Passo 6:** Calcula o resíduo $X^{i+1,j} = X^{i,j} - \hat{X}^{i,j}$ e codifica o *flag* '0'.

**Passo 7:** Se $m > n$, divide $X^{i+1,j}$ em $\begin{pmatrix} X^{i+1,2j+1} \\ X^{i+1,2j+2} \end{pmatrix}$, onde $X^{i+1,2j+1,}$ e $X^{i+1,2j+2}$ são *m/2 x n*.

Senão, divide $X^{i+1,j}$ em $\begin{pmatrix} X^{i+1,2j+1} & X^{i+1,2j+2} \end{pmatrix}$, onde $X^{i+1,2j+1,}$ e $X^{i+1,2j+2}$ são *m x n/2*.

**Passo 8:** Computa $\hat{X}^{i+1,2j+1} = \text{codifica}\left(X^{i+1,2j+1}, \eta_0, T\right)$.

**Passo 9:** Computa $\hat{X}^{i+1,2j+2} = \text{codifica}\left(X^{i+1,2j+2}, \eta_0, T\right)$.

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

Senão, faz $\hat{X}^{i+1,j} = \begin{pmatrix} \hat{X}^{i+1,2j+1} & \hat{X}^{i+1,2j+2} \end{pmatrix}$.

**Passo 11:** Atualiza $\hat{X}^{i,j} \leftarrow \hat{X}^{i,j} + \hat{X}^{i+1,j}$.

**Passo 12:** Atualiza os dicionários de todas as escalas com $\hat{X}^{i,j}$ e com $\hat{X}^{i+1,j}$, ou seja:

Para $n = 0, 1, ..., 6$,

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

$$D^{(p,q)} = D^{(p,q)} \cup T_{m,n}^{p,q}[\hat{X}^{i,j}] \cup T_{m,n}^{p,q}[\hat{X}^{i+1,j}].$$

**Passo 13:** Retorna $\hat{X}^{i,j}$.

A implementação do UMMP-RD2 utilizou os mesmos parâmetros do UMMP-RD1, mantendo-se 81.920 como o número máximo de elementos de cada dicionário, independentemente do tamanho de cada um de seus subdicionários. Novamente, em nenhum caso testado, este limite foi atingido. Para a codificação dos índices que indicam o subdicionário escolhido também foi utilizado um codificador aritmético com modelos distintos para cada escala. O dicionário inicial foi alocado, para todas as escalas, no subdicionário 1x1, ou seja, os elementos presentes no dicionário inicial possuem 1x1 como a sua escala de origem.

Os resultados obtidos nos testes feitos com esta versão serão mostrados mais à frente, na Seção 4.5, de forma consolidada com as outras versões implementadas.

## 4.4 UMMP-RD3

A terceira versão do UMMP otimizado implementada neste trabalho, o UMMP-RD3, é simplesmente uma extensão da segunda. A idéia da segmentação dos dicionários no espaço de códigos foi introduzida no MMP em [12] e utilizada na seção anterior como tentativa de melhorar o desempenho do UMMP. Porém, uma diferença entre os algoritmos MMP e UMMP é que, neste, o dicionário é atualizado duas vezes a cada aproximação de dois nós da árvore, enquanto que, naquele, o dicionário é atualizado somente uma vez.

Com isso, surgiu a idéia de dividir cada um dos dicionários também de acordo com o tipo de atualização, e não só de acordo com a escala de origem da atualização, como no UMMP-RD2. Assim, cada um dos sete dicionários existentes (de dimensões 8x8, 8x4, 4x4, 4x2, 2x2, 2x1 e 1x1) passa a ser dividido como ilustrado na Figura 4.4.

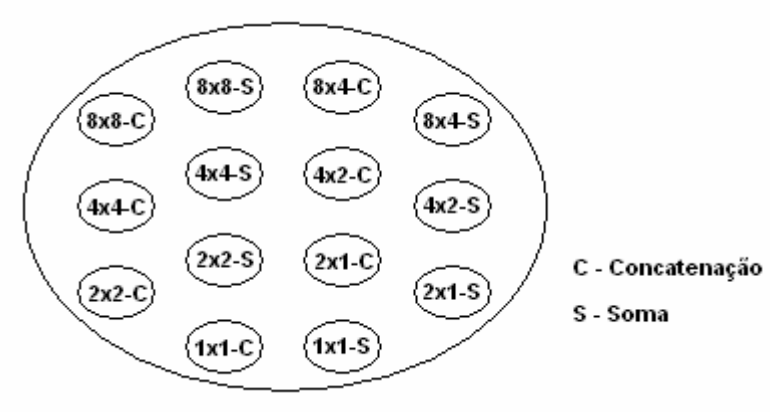


Figura 4.4: Divisão de cada um dos dicionários no UMMP-RD3.

Os subdicionários do tipo *C* contêm elementos provenientes da atualização feita com um bloco gerado pela concatenação de dois resíduos aproximados, ou seja, com $\hat{R}^{i+1,j}$, mostrado na Figura 4.5, que reproduz a Figura 3.2. Já os subdicionários do tipo *S* são formados por elementos originários da atualização feita com o bloco gerado pela soma de $\hat{R}^{i+1,j}$ à aproximação do nó-pai $R^{i,j}$, ou seja, com $\hat{R}^{i,j}$.

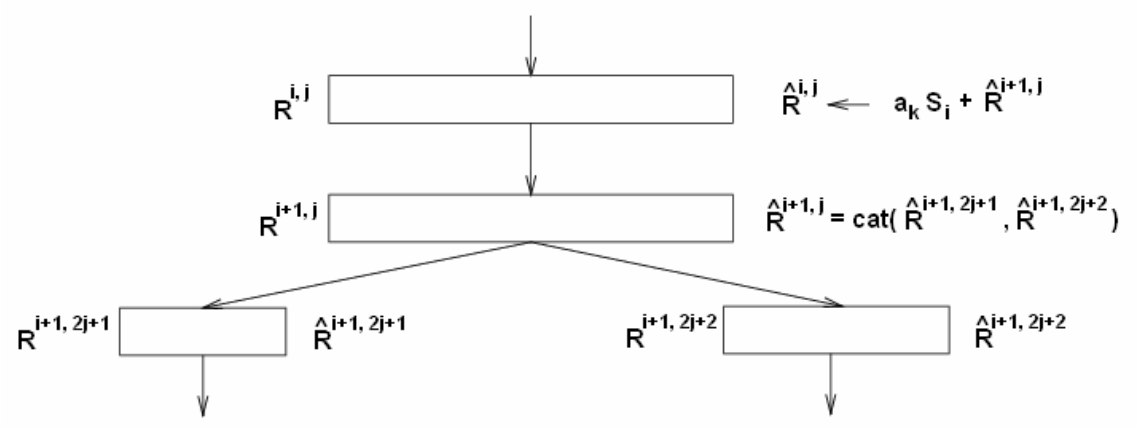


Figura 4.5: Atualização do dicionário no UMMP.

Os algoritmos para a otimização da árvore de segmentação e para a codificação propriamente dita são rigorosamente os mesmos mostrados na seção 4.3 para o UMMP-RD2, já que a única mudança implementada foi o aumento do número de subdicionários de sete para catorze em cada dicionário. Os elementos do dicionário inicial foram alocados no subdicionário 1x1-C de cada dicionário.

Os resultados obtidos nos testes realizados com o UMMP-RD3 serão mostrados na próxima seção, de forma consolidada com as outras versões implementadas.

## 4.5 Resultados experimentais

Nesta seção, mostramos uma síntese dos resultados obtidos nas simulações feitas com os algoritmos UMMP-RD1, UMMP-RD2 e UMMP-RD3. Novamente foram testadas as imagens *aerial, baboon, barbara, bridge, f-16, gold, lena, pp1205 e pp1209*, todas de tamanho 512x512 *pixels* e mostradas no Apêndice E. Os resultados obtidos são comparados com o UMMP (Capítulo 3), o MMP-RDI [6] e o SPIHT [4].

A implementação feita neste trabalho contemplou os seguintes aspectos:

- a imagem foi divida em blocos de tamanho 8x8, varridos da esquerda para a direita e de cima para baixo, ou seja $M = 2^K = N = 2^L = 8$;
- a segmentação de blocos $m \times n$ foi feita de forma que, se $m = n$, a partição é vertical, ou seja os blocos-filhos serão de dimensão $m \times n/2$, e se $m > n$, a partição é horizontal, ou seja os blocos-filhos terão a dimensão $m/2 \times n$;
- foram implementados K+L+1 = 3+3+1 = 7 dicionários, de dimensões 1x1, 2x1, 2x2, 4x2, 4x4, 8x4 e 8x8, divididos em sete subdicionários no UMMP-RD2 e em catorze subdicionários no UMMP-RD3. O dicionário $D^{(m,n)}$ possui elementos de dimensão $m \times n$;
- o dicionário inicial utilizado possui 513 elementos, sendo composto por todos os níveis de cinza possíveis da imagem e seus simétricos, ou seja, $D^{(1,1)} = \{-256, -255,..., -1, 0, 1,..., 255, 256\}$. Os dicionários iniciais nas outras escalas foram obtidos a partir de $D^{(1,1)}$ pela aplicação de transformações de escala;
- o tamanho de cada dicionário foi limitado a 81.920 elementos. Quando este limite fosse atingido (o que não ocorreu em nenhum caso), a inserção de um novo elemento seria feita com a exclusão do elemento menos utilizado dentre os 50% mais antigos;
- a transformação de escala utilizada foi a mesma descrita na seção 2.2 pelas Equações 2.5, 2.6, 2.7, 2.8, 2.9 e 2.10; e
- os índices dos elementos dos subdicionários, os índices que indicam a qual subdicionário o elemento pertence, e os *flags* descritores da árvore de segmentação foram codificados por um codificador aritmético adaptativo com modelos independentes para cada uma das escalas.

As Figuras 4.6 a 4.14 mostram os gráficos obtidos nas simulações feitas em computador.

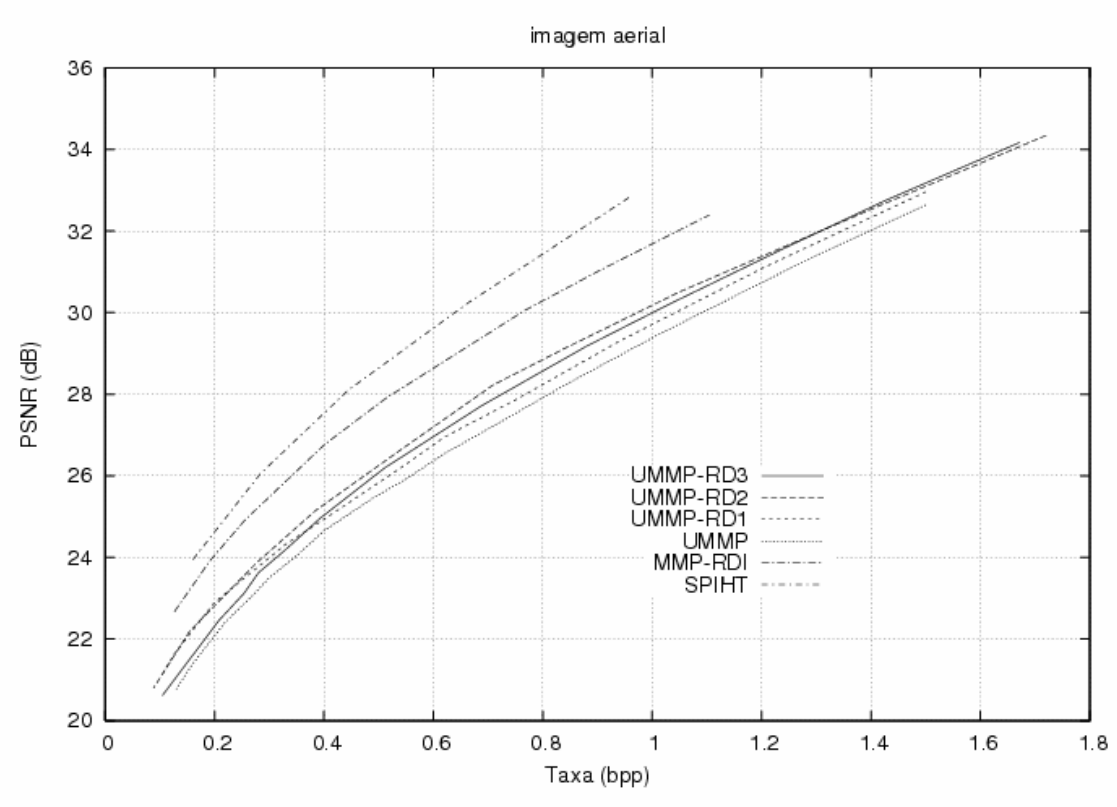


Figura 4.6: Taxa-distorção para *aerial* comprimida pelos UMMP-RDs.

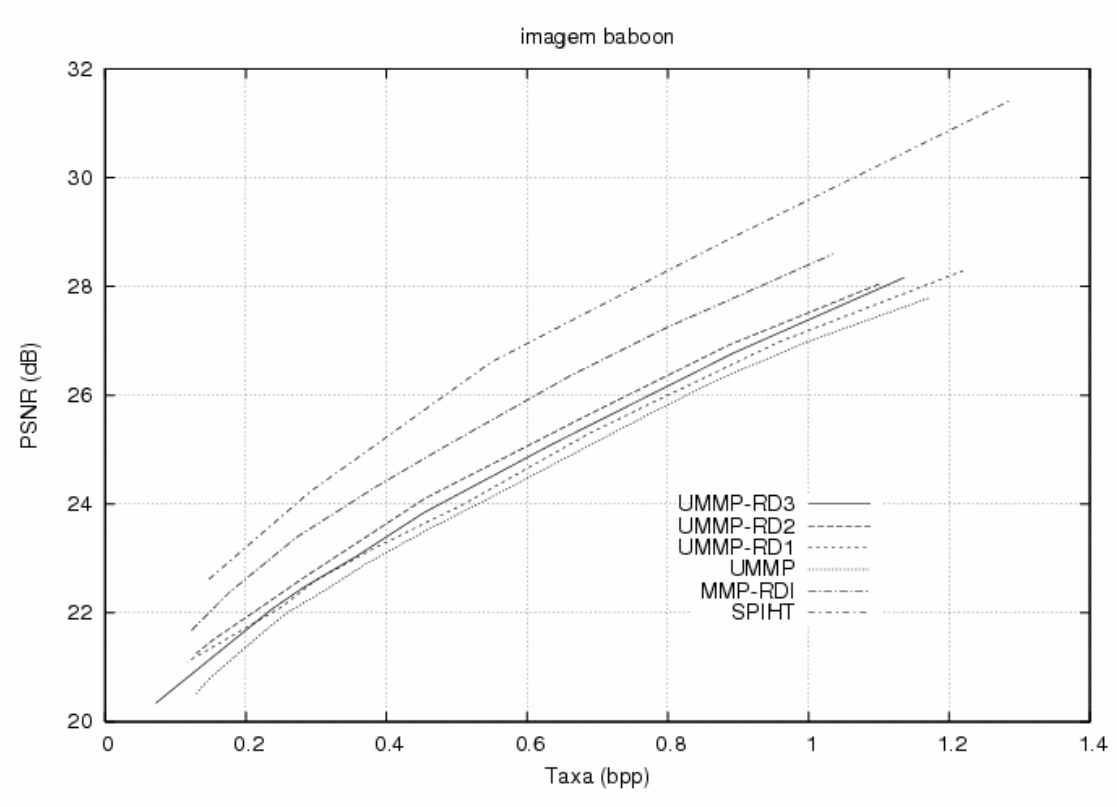


Figura 4.7: Taxa-distorção para *baboon* comprimida pelos UMMP-RDs.

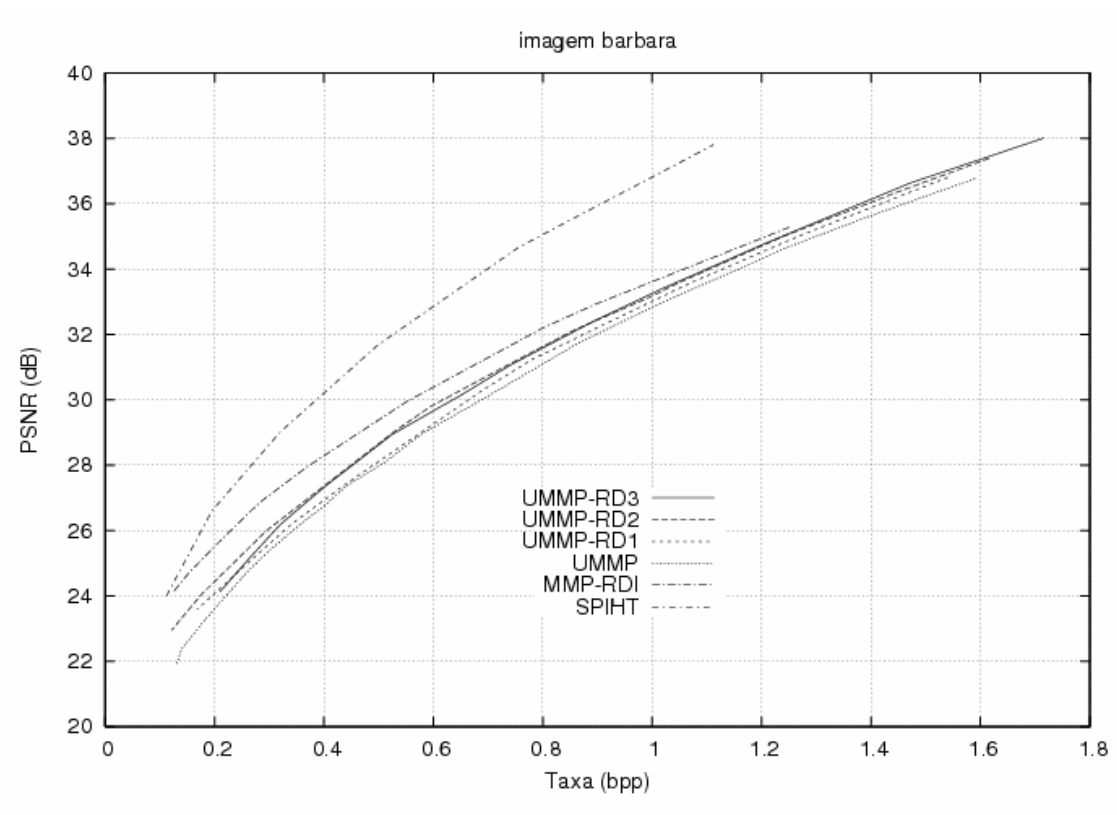


Figura 4.8: Taxa-distorção para *barbara* comprimida pelos UMMP-RDs.

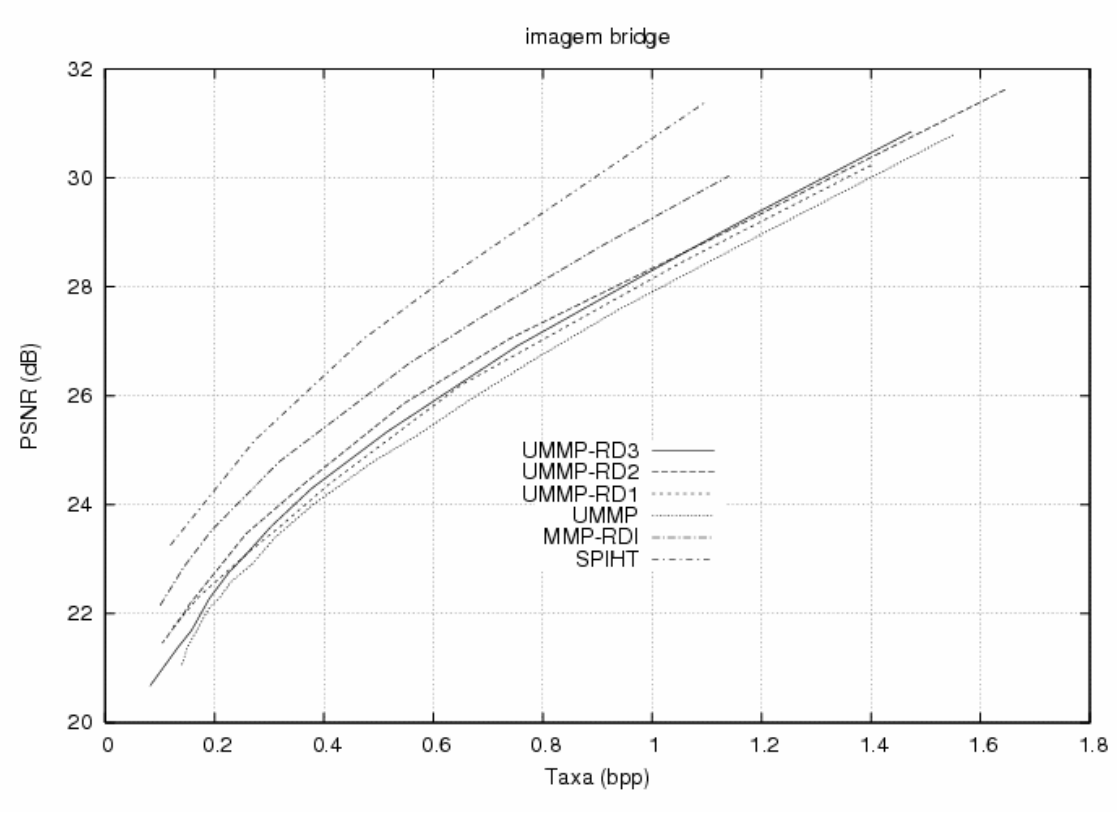


Figura 4.9: Taxa-distorção para *bridge* comprimida pelos UMMP-RDs.

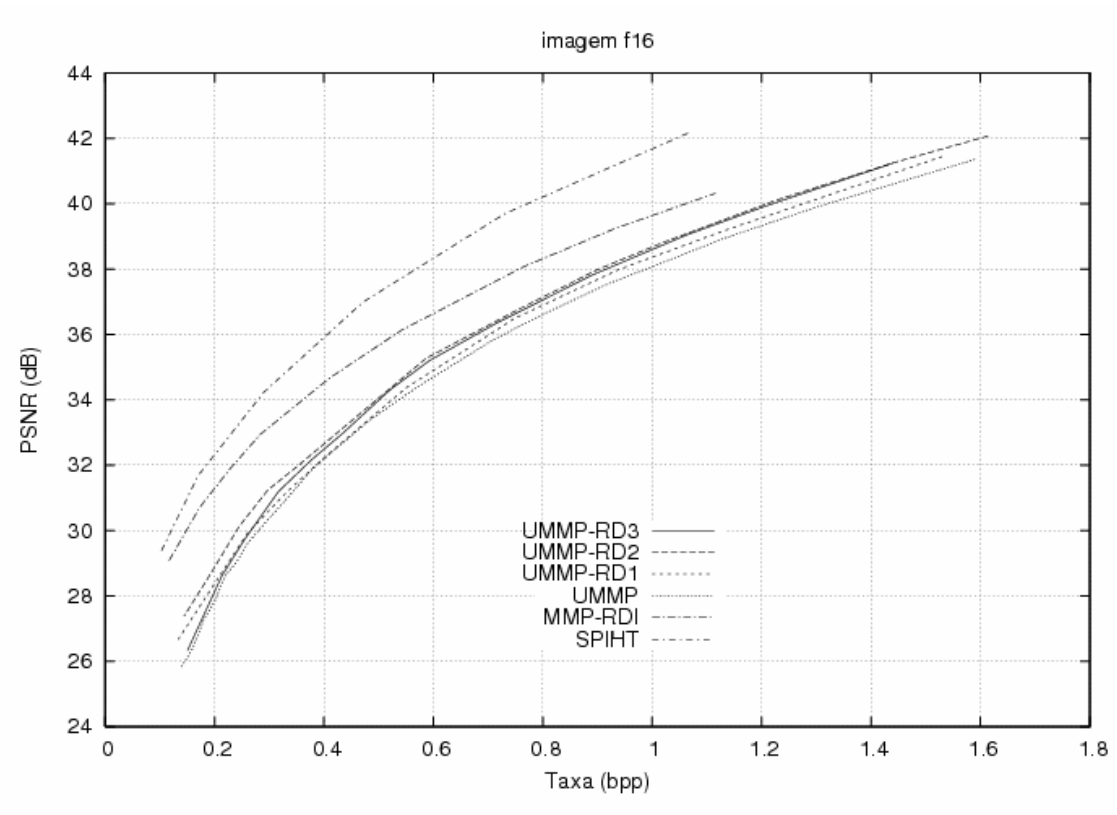


Figura 4.10: Taxa-distorção para *f-16* comprimida pelos UMMP-RDs.

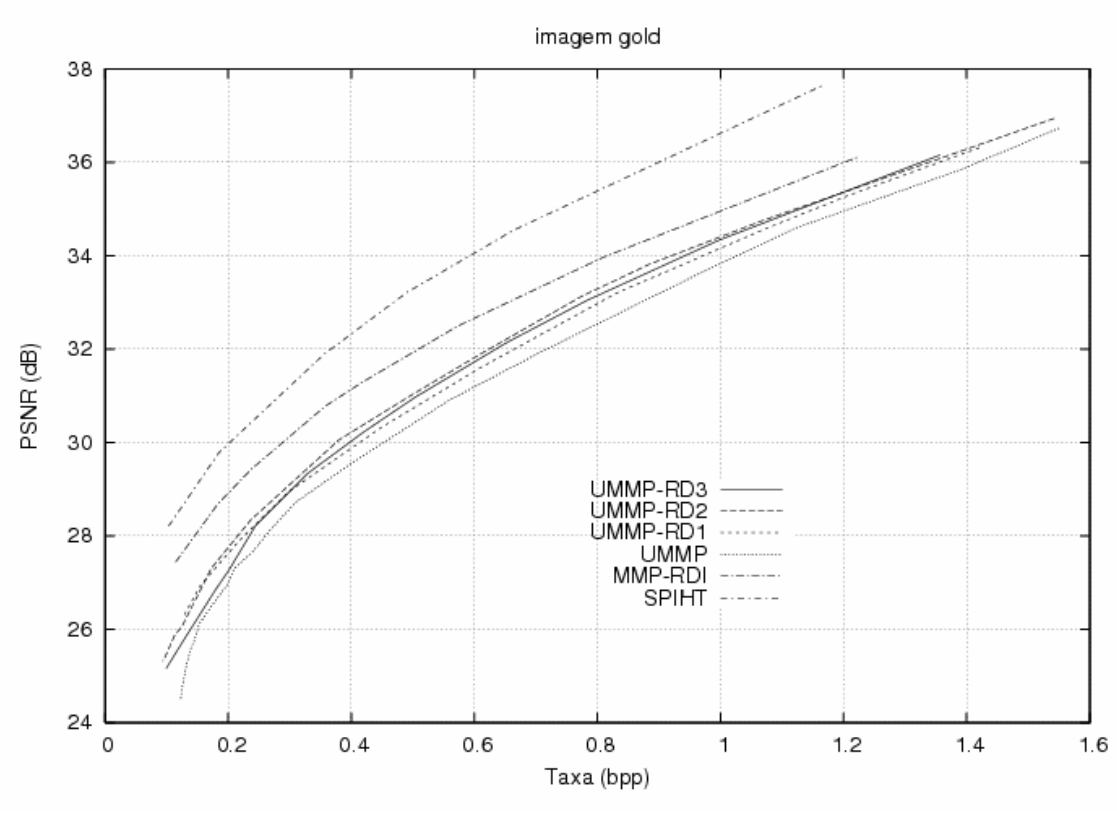


Figura 4.11: Taxa-distorção para *gold* comprimida pelos UMMP-RDs.

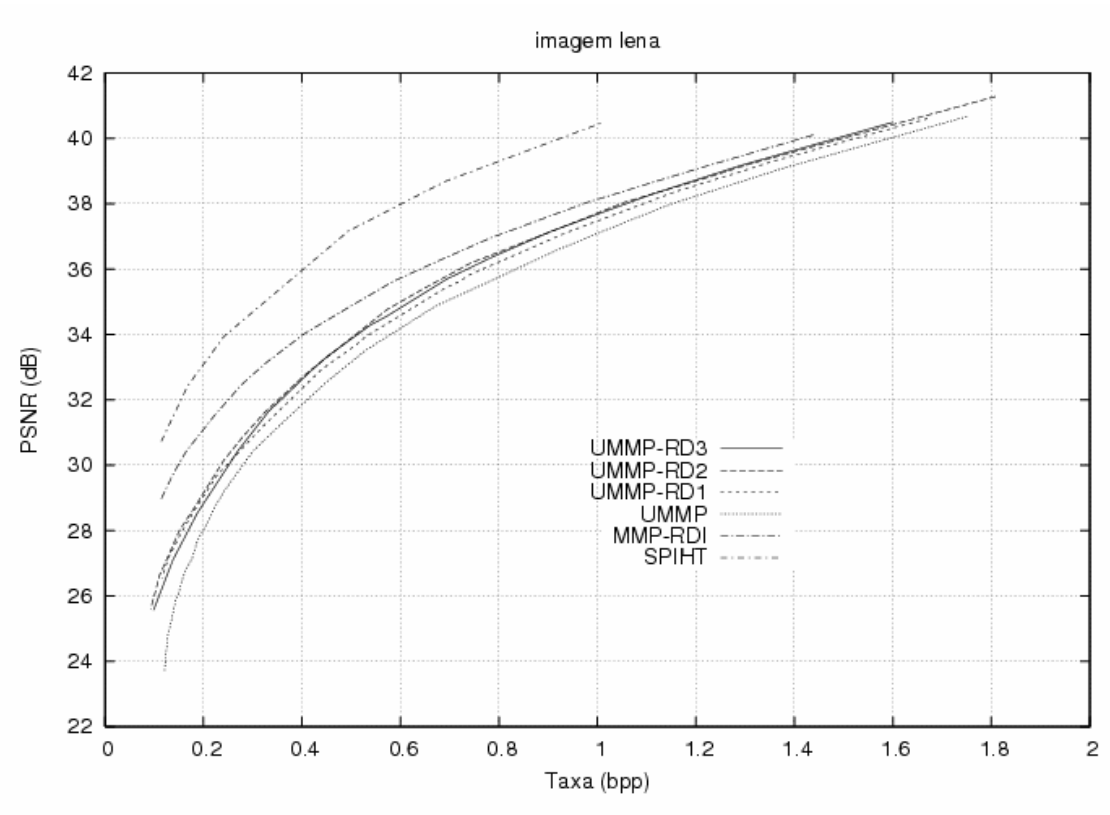


Figura 4.12: Taxa-distorção para *lena* comprimida pelos UMMP-RDs.

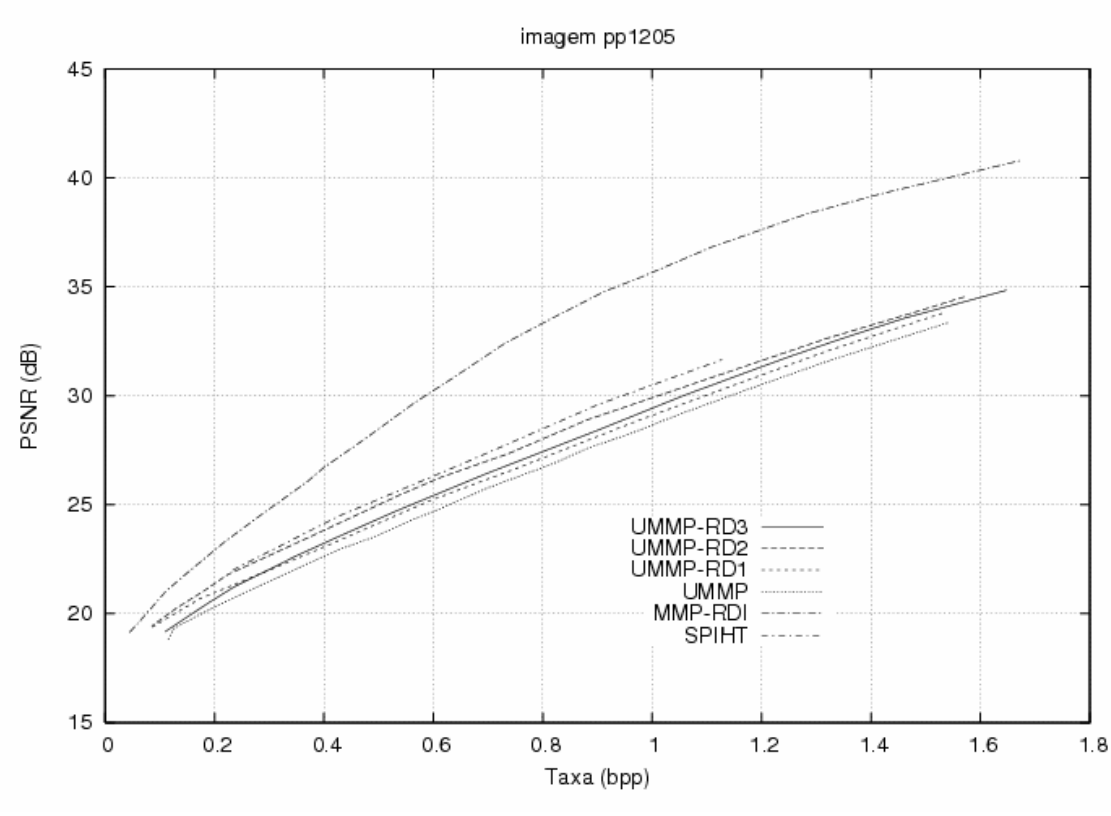


Figura 4.13: Taxa-distorção para *pp1205* comprimida pelos UMMP-RDs.

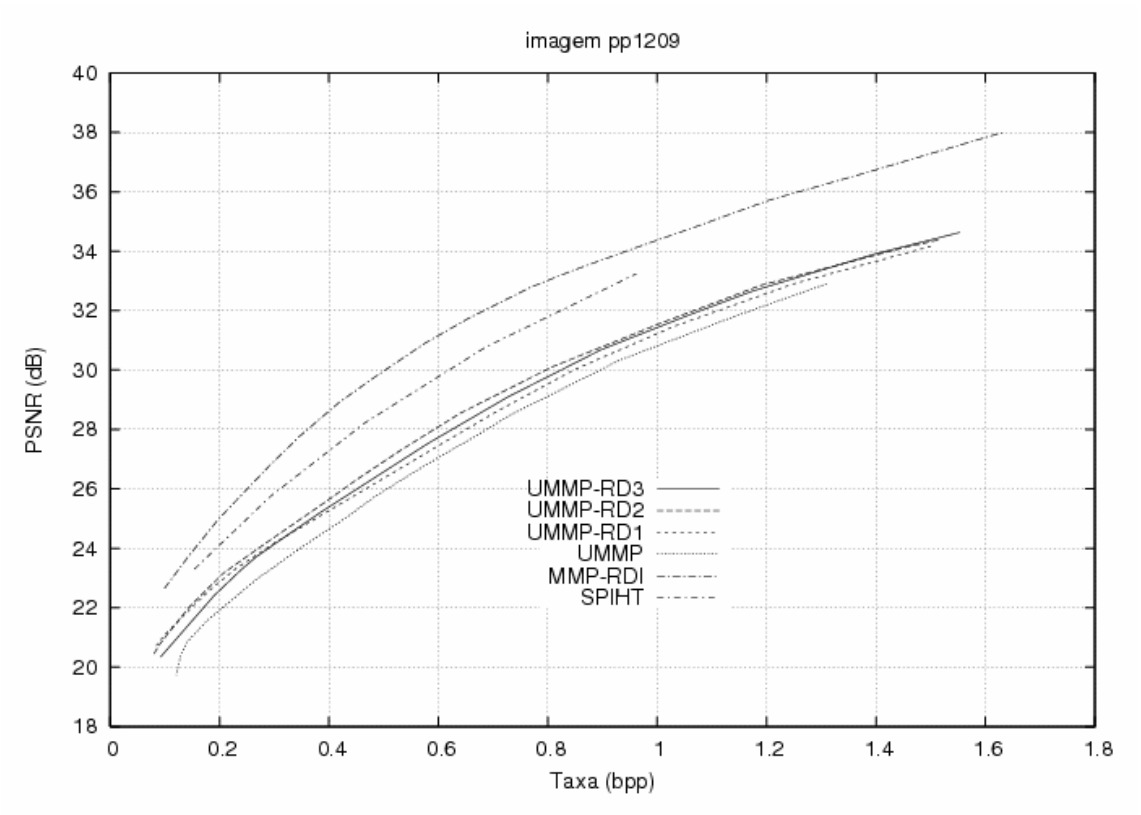


Figura 4.14: Taxa-distorção para *pp1209* comprimida pelos UMMP-RDs.

Da observação dos gráficos podemos tirar algumas conclusões. Primeiramente, percebe-se que, dentre as três versões do UMMP otimizado implementadas, o UMMP-RD2 obteve o melhor desempenho em todas as imagens para todas as taxas experimentadas, com o UMMP-RD3 atingindo resultados similares, a partir de determinada taxa, em quase todas as imagens (a única exceção foi para a imagem *pp1205*, caso em que o UMMP-RD2 foi superior para todas as taxas testadas).

O ganho conseguido com a implementação da idéia da segmentação dos dicionários no espaço de códigos se refletiu na superioridade do UMMP-RD2 em relação ao UMMP-RD1, que chegou a 0,7dB nas imagens suaves e a 1dB na imagem de texto (*pp1205*).

Com relação à melhora esperada pela incorporação da otimização da árvore de segmentação ao algoritmo UMMP original, temos que o ganho do UMMP-RD2 em relação ao UMMP original ficou entre 0,5dB e 1dB, na grande maioria dos casos, para as imagens suaves, e entre 1dB e 1,6dB para a imagem de texto.

Porém, mesmo com os ganhos de desempenho inseridos pela otimização e pela segmentação no espaço de códigos, o UMMP ainda ficou um pouco distante do MMP-

RDI, que foi superior em até 1,5dB a 0,4bpp para *lena*. Na maioria dos casos, entretanto, este déficit ficou entre 0,5dB e 1dB para as imagens suaves. Já para a imagem de texto, cuja codificação é o grande destaque do MMP-RDI, este foi melhor que o UMMP-RD2 em 1,5dB para taxas baixas e em 6dB a 1bpp.

Finalmente, em relação ao SPIHT, temos que o UMMP-RD2 ficou atrás em até 3,5dB (imagem *lena* codificada a 0,2bpp). Na grande maioria dos casos, para as imagens suaves, a diferença ficou entre 1dB e 2,5dB. Já para a imagem *pp1205*, o UMMP-RD2 ficou bem próximo do SPIHT, perdendo por, no máximo, 0,5dB.

No próximo capítulo, descrevemos o UMMP-H, que incorpora um codificador de Haar ao algoritmo UMMP-RD descrito neste capítulo.

# Capítulo 5

# Algoritmo UMMP-H

Neste capítulo, descrevemos a incorporação ao algoritmo UMMP otimizado de um codificador baseado na Transformada de Haar [13] na tentativa de obter uma melhora em seu desempenho. Implementamos duas versões deste novo algoritmo, o UMMP-H1 e o UMMP-H2.

## 5.1 Introdução

Os algoritmos MMP e UMMP, originalmente propostos em [5], se baseiam na idéia de casamento aproximado de padrões recorrentes com escalas para codificação, principalmente, de imagens. Estes algoritmos criaram uma nova classe de codificadores, já que são conceitualmente diferentes dos algoritmos largamente utilizados na atualidade para codificação de imagens, que são baseados em transformadas, como o SPIHT [4] e o JPEG [3], que se utilizam, respectivamente, da DWT e da DCT.

Neste capítulo, propomos a incorporação da idéia dos codificadores por transformada ao UMMP. Como pontapé inicial, implementamos um codificador simples que utiliza a Transformada de Haar para codificação de imagens e o incorporamos ao UMMP otimizado descrito no capítulo anterior.

Basicamente, o algoritmo faz a codificação de um bloco da imagem utilizando ambos os métodos e escolhe aquele que o codifica com menor custo. Esta escolha é sinalizada ao decodificador com o uso de um *flag* específico.

## 5.2 Codificadores por transformada

A classe de codificadores por transformada vem sendo largamente utilizada para codificação de diversos tipos de dados, como áudio, voz, imagem e vídeo. Com grande destaque na área de imagem, temos o SPIHT [4] e o JPEG2000 [14], que utilizam a DWT (Transformada Discreta de *Wavelet*), e o JPEG [3], que utiliza a DCT (Transformada Discreta dos Cossenos).

Um codificador por transformada normalmente executa sua tarefa em três passos: transformação, quantização e codificação por entropia. Assim, a um bloco de entrada é aplicada uma transformada, gerando seus coeficientes, que são, a seguir, quantizados. Por fim, aplica-se um codificador por entropia aos coeficientes quantizados para obtenção do *bitstream*.

O segredo deste método de codificação reside basicamente na etapa da transformação. Nela, deseja-se manipular os coeficientes do vetor de entrada de modo que a aplicação da quantização e da codificação por entropia feita aos coeficientes transformados seja mais eficiente do que se ela fosse feita diretamente aos coeficientes originais. Assim, é desejável que a transformada utilizada seja capaz de compactar a maior parte da energia do sinal de entrada em um pequeno número de coeficientes, para que menos coeficientes precisem ser codificados.

Uma transformada nada mais é do que a expansão de um sinal pertencente a determinado espaço vetorial $V$ em uma base desse espaço. Assim, qualquer vetor, de dimensão Nx1, $\mathrm{x} = (x_0\ x_1 \ldots x_{N-1})^T \in V$ pode ser representado como uma combinação linear de N vetores base $\mathrm{b_k}$:

$$\mathrm{x} = \sum_{k=0}^{N-1} \mathrm{X}_k \mathrm{b}_k = \mathrm{B.X} \tag{5.1}$$

onde $\mathrm{X_k}$ são as componentes do vetor **x** na base $B = \{\mathrm{b_k}\}_{k=0}^{N-1}$ e formam o vetor de coeficientes **X**, de dimensão Nx1, e B é a matriz NxN formada pelos vetores base $\mathrm{b_k}$.

A transformada inversa é aquela que recupera o vetor **x** a partir dos coeficientes transformados, e é dada, quando a matriz B for inversível, por:

$$X = \sum_{k=0}^{N-1} x_k a_k = A.x = B^{-1}.x \tag{5.2}$$

Em termos bidimensionais, uma matriz, ou uma imagem, **x**, de dimensão NxN, pode ser decomposta em uma base formada por $N^2$ imagens de dimensão NxN. Considerando somente as transformadas separáveis, classe na qual se enquadra a grande maioria das transformadas bidimensionais mais utilizadas, as imagens base podem ser decompostas como:

$$b_{k,l} = b'_k.b''^{T}_l \tag{5.3}$$

onde $b'_k$ e $b''_l$ são vetores Nx1. Neste caso, a transformada bidimensional pode ser escrita como:

$$x = B'.X.B''^{T} \tag{5.4}$$

onde $B'$ e $B''$ são matrizes NxN compostas, respectivamente, pelos vetores $b'_k$ e $b''_l$. Finalmente, a transformada inversa é dada por:

$$X = A'.x.A''^{T} = B'^{-1}.x.\left(B''^{-1}\right)^{T} \tag{5.5}$$

## 5.3 Codificador de Haar

Como a intenção neste trabalho é apenas introduzir a idéia da incorporação do conceito de codificação por transformada ao UMMP e testar se ela pode gerar um incremento em seu desempenho, utilizamos somente a Transformada de Haar [13], mesmo sabendo que sua capacidade de compactação de energia, inclusive para imagens suaves, não é muito grande se comparada a outras transformadas bastante utilizadas

para este fim, como a Transformada Discreta dos Cossenos (DCT) e versões mais complexas da Transformada Discreta de *Wavelet* (DWT).

A Transformada de Haar bidimensional é separável, sendo definida por:

$$X = H_n.x.H_n^T \tag{5.6}$$

onde $H_n$ é a matriz da Transformada de Haar de dimensão NxN. No caso em estudo, utilizaremos a transformada de dimensão 8x8, já que codificamos blocos com esse tamanho. Assim, temos que a matriz da Transformada de Haar utilizada é dada por:

$$H_8 = \frac{1}{\sqrt{8}} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ \sqrt{2} & \sqrt{2} & -\sqrt{2} & -\sqrt{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sqrt{2} & \sqrt{2} & -\sqrt{2} & -\sqrt{2} \\ 2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & -2 \end{bmatrix} \tag{5.7}$$

A Transformada de Haar, como se pode observar na Equação 5.7, é unitária e real, ou seja, $H_n^{-1} = H_n^T$. Assim, a sua inversa é dada por:

$$x = H_n^T.X.H_n \tag{5.8}$$

Para a etapa de quantização dos coeficientes obtidos pela aplicação da Transformada de Haar ao bloco de entrada, foi utilizado o algoritmo *MultiGrid Embedding* (MGE), proposto em [15]. O MGE basicamente decompõe o domínio da transformada em uma *quadtree* para fazer a codificação dos coeficientes em planos de bits. Ele utiliza um *grid* que vai sendo dividido, de uma forma bastante eficiente, sobre o domínio da transformada da imagem a fim de encontrar os coeficientes significativos para determinado plano de *bits*. Um exemplo de divisão do *grid* pode ser observado na Figura 5.1.

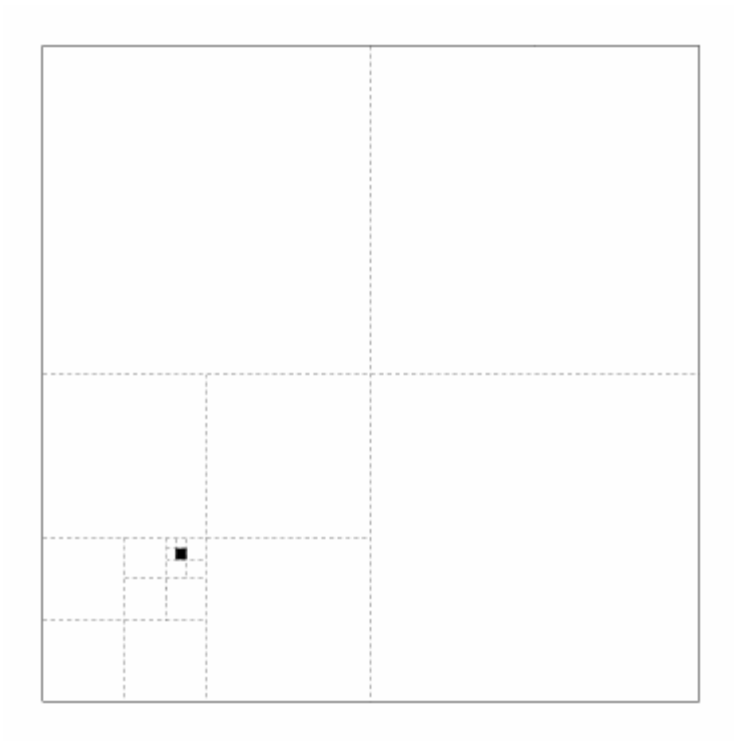

Figura 5.1: Exemplo de divisão em *quadtree* no MGE.

O algoritmo originalmente proposto em [15] pode ser implementado como descrito a seguir:

1. Inicialização: encontre o coeficiente $C_{x,y}$ com maior valor absoluto e obtenha $n = \lfloor \log_2 \left( \max_{(x,y)} \left\{ \left| C_{x,y} \right| \right\} \right) \rfloor$. Escolha a região a codificar (*G*) como sendo todo o domínio da transformada e marque-a como sendo uma "região zero" (*T = 0*).
2. Passo Dominante:
    a. Para cada "região zero", escreva '1' e marque a região como "região um" se ela possuir algum elemento significativo, ou seja, se ela possuir algum coeficiente cujo valor absoluto seja maior ou igual a $2^n$, e escreva '0' caso contrário.
    b. Se a região for uma "região um", divida-a em quatro novas regiões de mesmo tamanho, marcando cada uma delas como "região zero". Volte para 2a até que o tamanho das regiões seja 2x2.
    c. Mova todas regiões 2x2 para a Lista de Regiões Significantes (SGL) e escreva o sinal de cada elemento $C_{x,y}$ pertencente às regiões.

3. Passo de Refinamento: Para cada elemento da SGL, com exceção daqueles adicionados no último Passo Dominante, escreva o n-ésimo *bit* mais significativo de $\left|C_{x,y}\right|$.
4. Decremente *n* de uma unidade e volte para o Passo Dominante.

Por fim, os símbolos gerados pelo MGE são codificados por um codificador aritmético para obtenção do *bitstream* final.

O algoritmo completo que implementou o Codificador de Haar utilizado neste trabalho encontra-se no Apêndice D.

## 5.4 UMMP-H1

Após a implementação do Codificador de Haar descrito na seção anterior, passamos à tentativa de sua incorporação ao algoritmo UMMP otimizado. A primeira versão do algoritmo UMMP-H foi desenvolvida com a utilização do UMMP-RD2, já que este se mostrou um pouco superior ao UMMP-RD3, conforme detalhado no Capítulo 4.

A implementação do algoritmo, chamado de UMMP-H1, foi bastante simples, e consistiu em utilizar os dois codificadores (UMMP-RD2 e Haar) para codificar cada bloco 8x8 da imagem de entrada, e escolher aquele que obtivesse um resultado com menor custo *lagrangeano* [7, 8].

A escolha de qual codificador se saiu melhor foi sinalizada ao decodificador por meio de um *flag*, codificado por um codificador aritmético com modelo independente dos demais modelos utilizados no algoritmo.

Assim, com a imagem sendo dividida em blocos de dimensão 8x8 e varrida da esquerda para a direita e de cima para baixo, e considerando $X$ um bloco 8x8, o algoritmo foi implementado da seguinte forma:

**Procedimento** $\hat{X} = \text{codifica}(X)$

**Passo 1:** Obtém $\hat{X}_{Haar} = \text{codifica_haar}(X)$ e armazena o *bitstream* gerado pelo Codificador de Haar.

**Passo 2:** Calcula e armazena em $C_{Haar}$ o custo *lagrangeano* relativo ao Codificador de Haar.

**Passo 3:** Obtém $\hat{X}_{UMMP} = \text{codifica_ummprd2}(X)$ e armazena o *bitstream* gerado pelo UMMP-RD2.

**Passo 4:** Calcula e armazena em $C_{UMMP}$ o custo *lagrangeano* relativo ao UMMP-RD2.

**Passo 5:** Se $C_{Haar} < C_{UMMP}$, codifica o *flag* '0', faz $\hat{X} = \hat{X}_{Haar}$ e transmite o *bitstream* obtido no **Passo 1**.

Senão, codifica o *flag* '1', faz $\hat{X} = \hat{X}_{UMMP}$ e transmite o *bitstream* obtido no **Passo 3**.

Ressaltamos que também foi implementada uma versão do UMMP-H1 utilizando o UMMP-RD3 em substituição ao UMMP-RD2. Conforme esperado, os resultados obtidos, nesse caso, foram bastante semelhantes àqueles obtidos nos testes que utilizaram o UMMP-RD2.

Como já vimos, os algoritmos baseados no MMP e no UMMP possuem dicionários que se adaptam às características da imagem à medida que esta vai sendo comprimida. Para mantermos esta capacidade de adaptação no algoritmo UMMP-H1, implementamos também um esquema de atualização dos dicionários mesmo quando o algoritmo se decide pelo Codificador de Haar.

Este esquema foi elaborado com o intuito de se parecer ao máximo com aquele usado nas diferentes versões do UMMP-RD. Para isso, após a codificação de um bloco 8x8 da imagem original pelo Codificador de Haar, usamos os coeficientes transformados aproximados (ou seja, com os mesmos valores que serão obtidos pelo decodificador) e as imagens base da Transformada de Haar 8x8 para atualizar os dicionários.

Serão feitas atualizações a partir de cada uma das escalas existentes, em ordem, da menor para a maior, ou seja, primeiro atualizamos com um elemento 1x1, depois com elementos 2x1, e assim sucessivamente até a dimensão 8x8. Deste modo, continuaremos a ter uma "escala de origem" do elemento inserido em cada dicionário, o

que torna este esquema compatível com a idéia da segmentação no espaço de códigos usada no UMMP-RD2.

Para compatibilização com o UMMP-RD3, em cada escala, atualizamos os dicionários com dois elementos, sendo um proveniente diretamente das imagens base e outro da soma deste com a aproximação da imagem original até então. Isto só não é feito na escala 1x1, quando há apenas uma atualização, já que ela é responsável pela primeira aproximação (nível DC) da imagem original. Porém, como o dicionário inicial já possui os elementos compostos por todos os níveis possíveis, a escala 1x1 acaba não gerando nenhuma atualização de dicionário.

Inicialmente, dividimos o domínio da Transformada de Haar da forma mostrada na Figura 5.2, que indica as escalas atribuídas a cada um dos coeficientes da transformada.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1x1 | 2x2 | 4x4 | 4x4 | 8x8 | 8x8 | 8x8 | 8x8 |
| 2x1 | 2x2 | 4x4 | 4x4 | 8x8 | 8x8 | 8x8 | 8x8 |
| 4x2 | 4x2 | 4x4 | 4x4 | 8x8 | 8x8 | 8x8 | 8x8 |
| 4x2 | 4x2 | 4x4 | 4x4 | 8x8 | 8x8 | 8x8 | 8x8 |
| 8x4 | 8x4 | 8x4 | 8x4 | 8x8 | 8x8 | 8x8 | 8x8 |
| 8x4 | 8x4 | 8x4 | 8x4 | 8x8 | 8x8 | 8x8 | 8x8 |
| 8x4 | 8x4 | 8x4 | 8x4 | 8x8 | 8x8 | 8x8 | 8x8 |
| 8x4 | 8x4 | 8x4 | 8x4 | 8x8 | 8x8 | 8x8 | 8x8 |

Figura 5.2: Divisão do domínio da Transformada de Haar.

Assim, começamos atualizando os dicionários pelo elemento de dimensão 1x1 obtido pela aplicação da transformação de escala à imagem base $b_{0,0}$ ponderada pelo seu coeficiente $C_{0,0}$, que é uma primeira aproximação do bloco original. Como dissemos, este elemento não gera mudança nos dicionários, já que eles já o possuem.

A seguir, passamos à escala 2x1. Obtemos um bloco, que pode ser entendido como um incremento para a aproximação calculada na escala anterior, a partir da imagem base $b_{1,0}$ ponderada pelo seu coeficiente $C_{1,0}$. Atualizamos o dicionário com este elemento e também com a nova aproximação do bloco original, ambos contraídos para a dimensão 2x1.

A partir daí, realizamos o procedimento descrito no parágrafo anterior para todas as escalas, em ordem crescente, sempre obtendo um incremento para a aproximação do bloco original e atualizando os dicionários com ele e com a nova aproximação. O incremento em cada escala será o resultado do somatório das imagens base atribuídas a ela, conforme Figura 5.2, ponderadas pelos seus respectivos coeficientes.

Efetuamos diversos testes com o UMMP-H1, cujos resultados são mostrados ao final deste Capítulo, juntamente com os resultados obtidos com a segunda versão do UMMP-H, descrita na próxima seção.

## 5.5 UMMP-H2

A segunda tentativa de incorporação de um codificador por transformada, no caso, o Codificador de Haar descrito na Seção 5.3, ao UMMP-RD foi feita no algoritmo UMMP-H2. A sua diferença em relação ao UMMP-H1 consiste apenas em uma variação no Codificador de Haar, sendo o UMMP-RD2 implementado de forma idêntica à feita no UMMP-H1.

A idéia por trás da alteração efetuada é a codificação em separado do nível DC, que é dado pelo coeficiente $C_{0,0}$ da transformada, do bloco a ser codificado. Normalmente este coeficiente DC possui um valor absoluto bem mais elevado que os demais coeficientes, causando, no MGE, uma divisão prematura da árvore quaternária. Isso gera a necessidade de se indicar, com um *flag* para cada uma das diversas regiões criadas, que elas não possuem outros coeficientes significativos por diversos planos de *bits*, até que o segundo coeficiente com maior valor absoluto se torne significativo. Se não houvesse a divisão da região original, somente um *flag* indicativo seria necessário para cada um desses planos de *bits*, gerando uma grande economia de *bits*.

Assim, após a aplicação da Transformada de Haar no bloco de entrada, o nível DC é armazenado separadamente e o coeficiente DC da Transformada é zerado. A

seguir, inicia-se o MGE com uma pequena alteração em relação ao algoritmo mostrado na Seção 5.3. Agora, antes da execução de um Passo Dominante, é codificado o n-ésimo *bit* mais significativo do nível DC previamente armazenado.

Com relação ao algoritmo implementado, após a codificação do n-ésimo *bit* mais significativo do nível DC, é feita uma verificação da condição de parada da codificação, exatamente como descrito no Apêndice D.

Assim, a rotina implementada foi a mesma mostrada na Seção 5.4, com o acréscimo das alterações mencionadas no Codificador de Haar. Os resultados dos testes são mostrados e analisados a seguir.

## 5.6 Resultados experimentais

Nesta seção, mostramos os resultados obtidos nos testes executados com as duas versões do UMMP-H descritas nas seções anteriores. Utilizamos para os testes as imagens *aerial, baboon, barbara, bridge, f-16, gold, lena, pp1205 e pp1209*, todas de tamanho 512x512 *pixels* e mostradas no Apêndice E. Comparamos os resultados obtidos pelo UMMP-H1 e pelo UMMP-H2 com o MMP-RDI [6] e com a segunda versão do UMMP-RD (Capítulo 4).

A implementação, para as duas versões, utilizou o UMMP-RD2 exatamente como descrito e implementado no Capítulo 4 e o Codificador de Haar conforme descrito na Seção 5.3. As Figuras 5.3 a 5.11 contêm os gráficos obtidos.

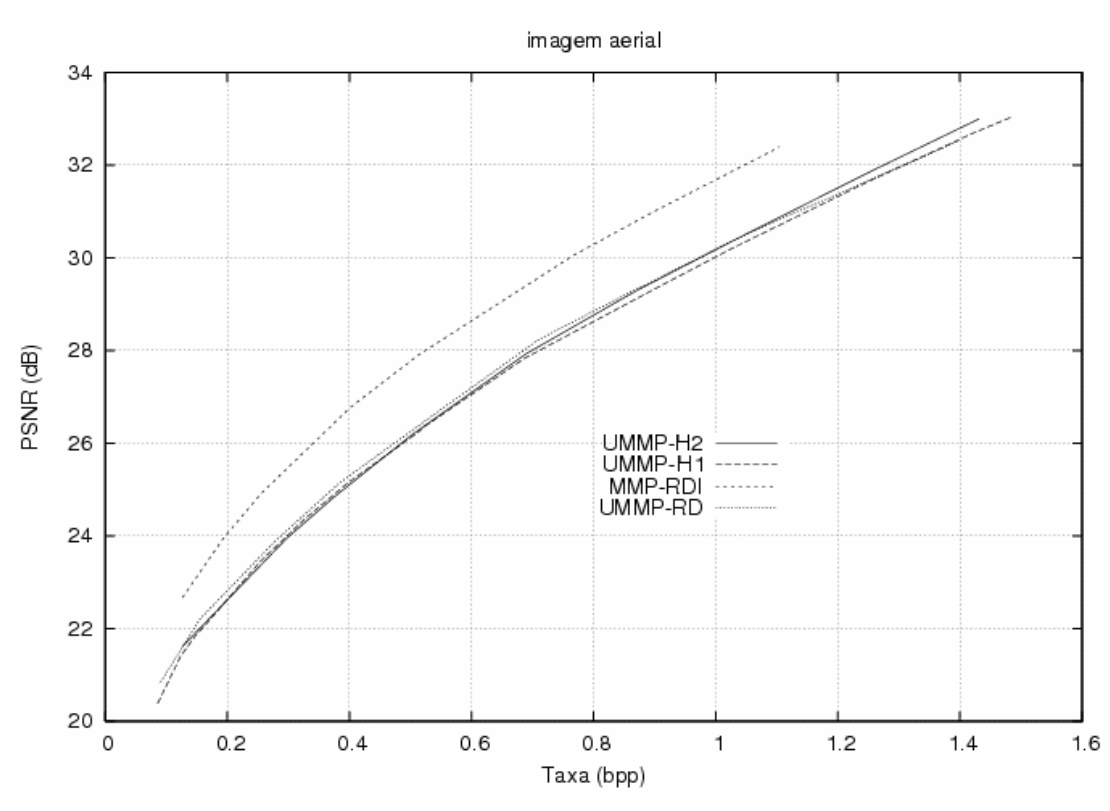


Figura 5.3: Taxa-distorção para *aerial* comprimida pelo UMMP-H.

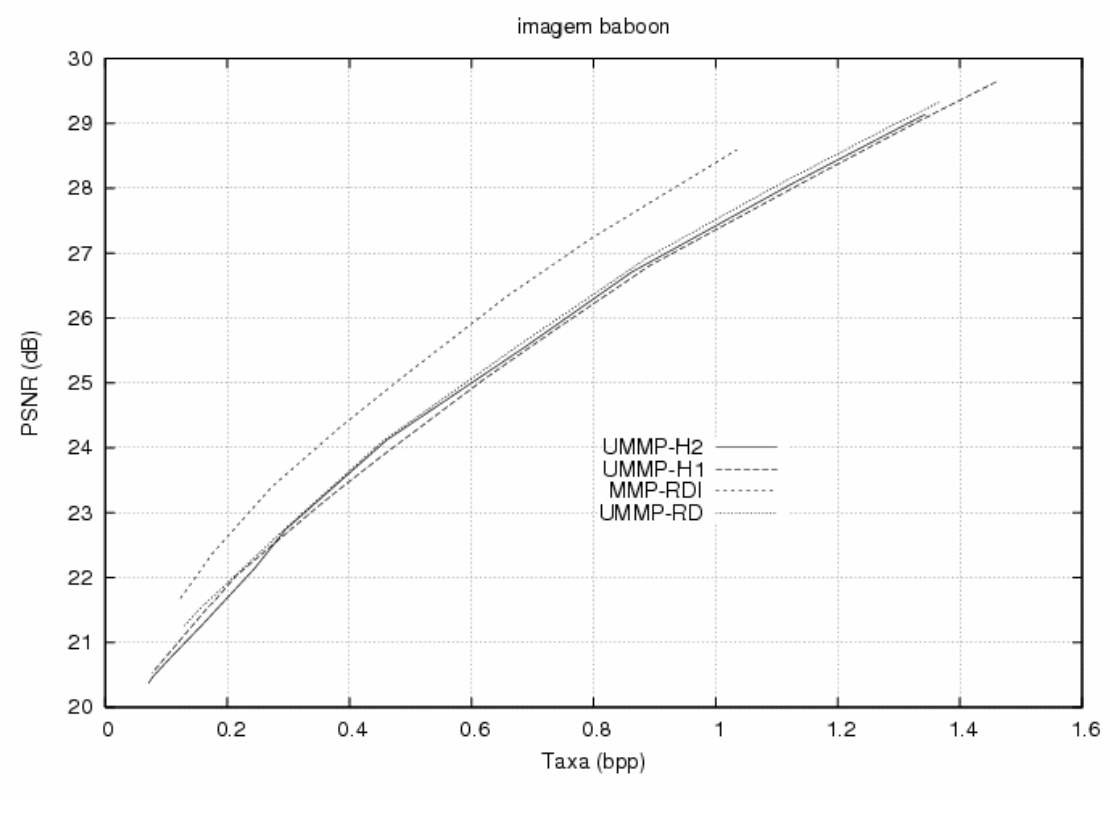


Figura 5.4: Taxa-distorção para *baboon* comprimida pelo UMMP-H.

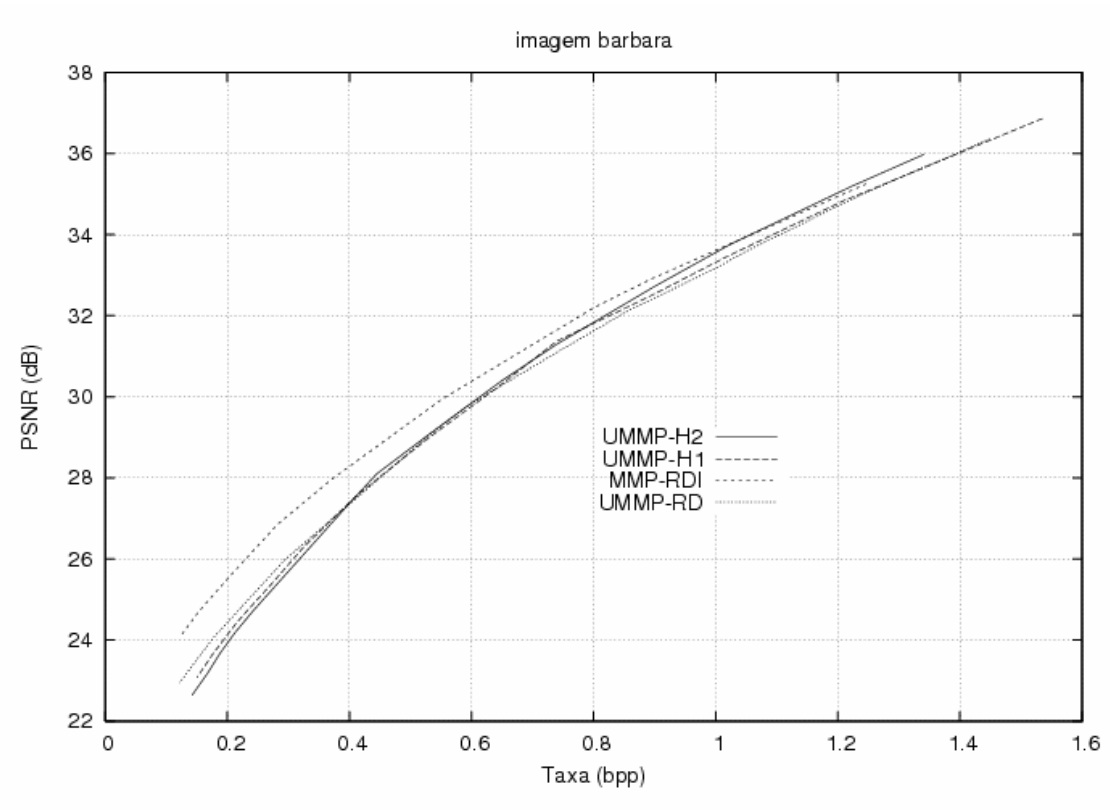


Figura 5.5: Taxa-distorção para *barbara* comprimida pelo UMMP-H.

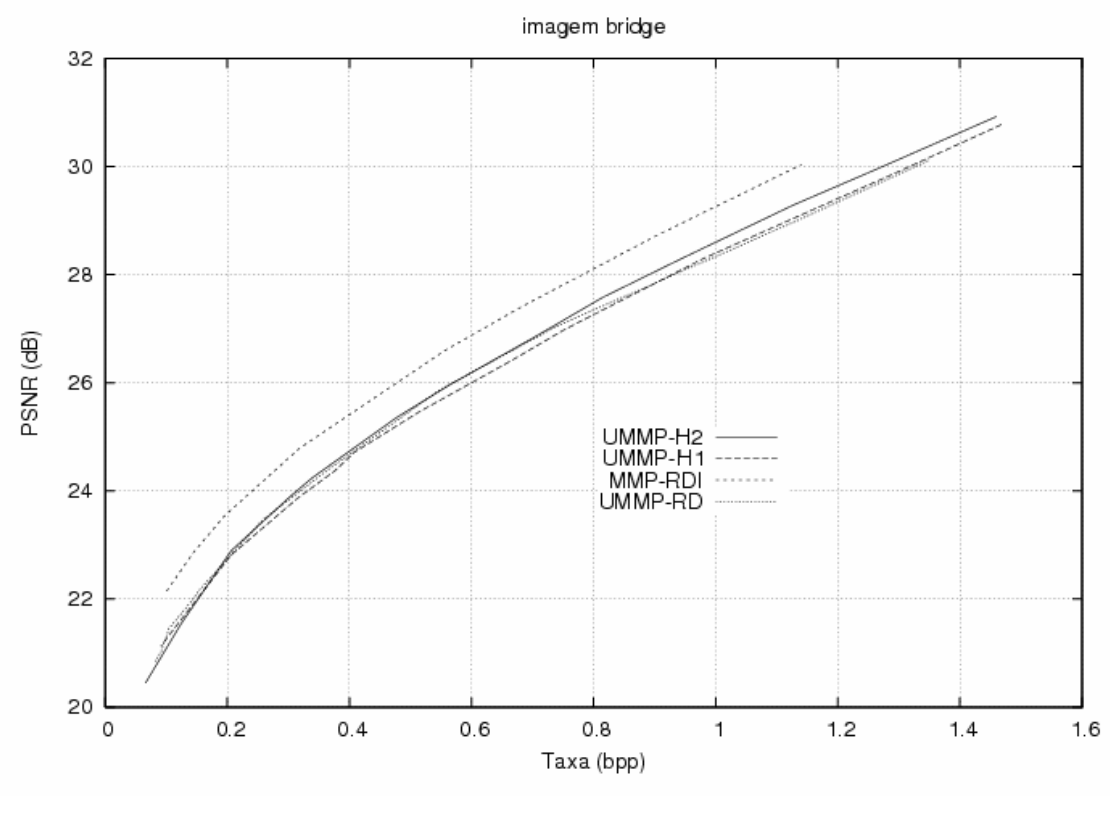


Figura 5.6: Taxa-distorção para *bridge* comprimida pelo UMMP-H.

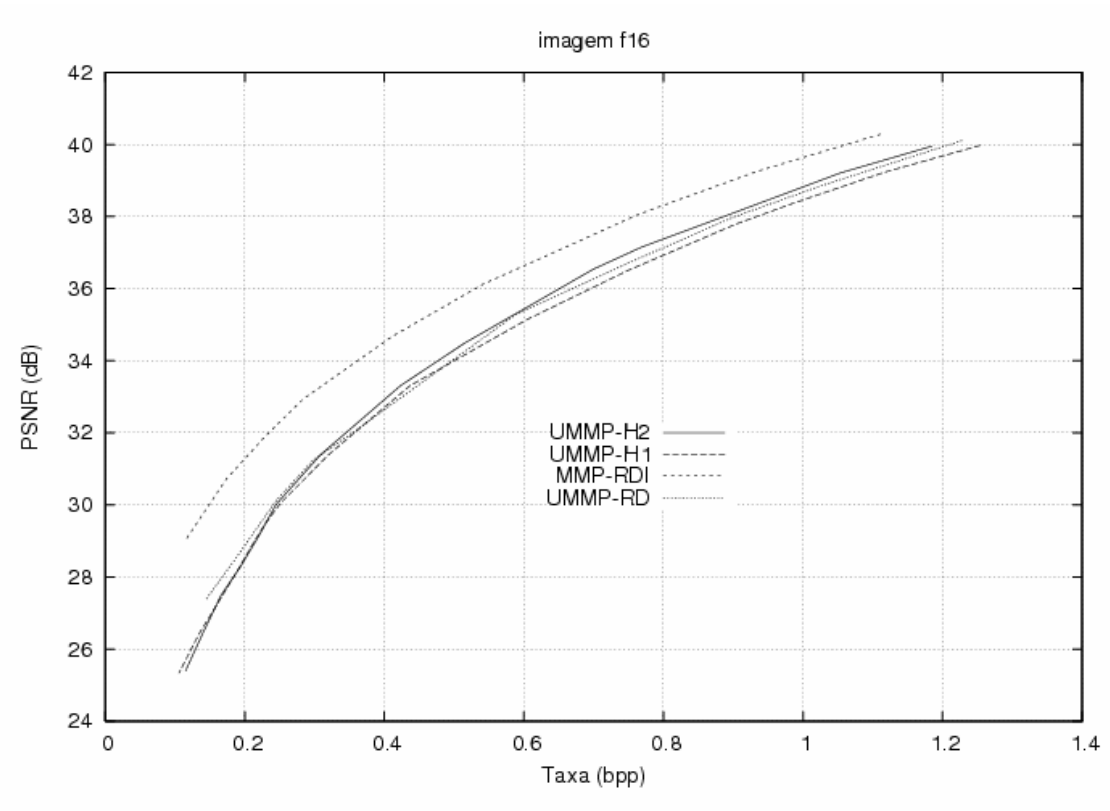


Figura 5.7: Taxa-distorção para *f-16* comprimida pelo UMMP-H.

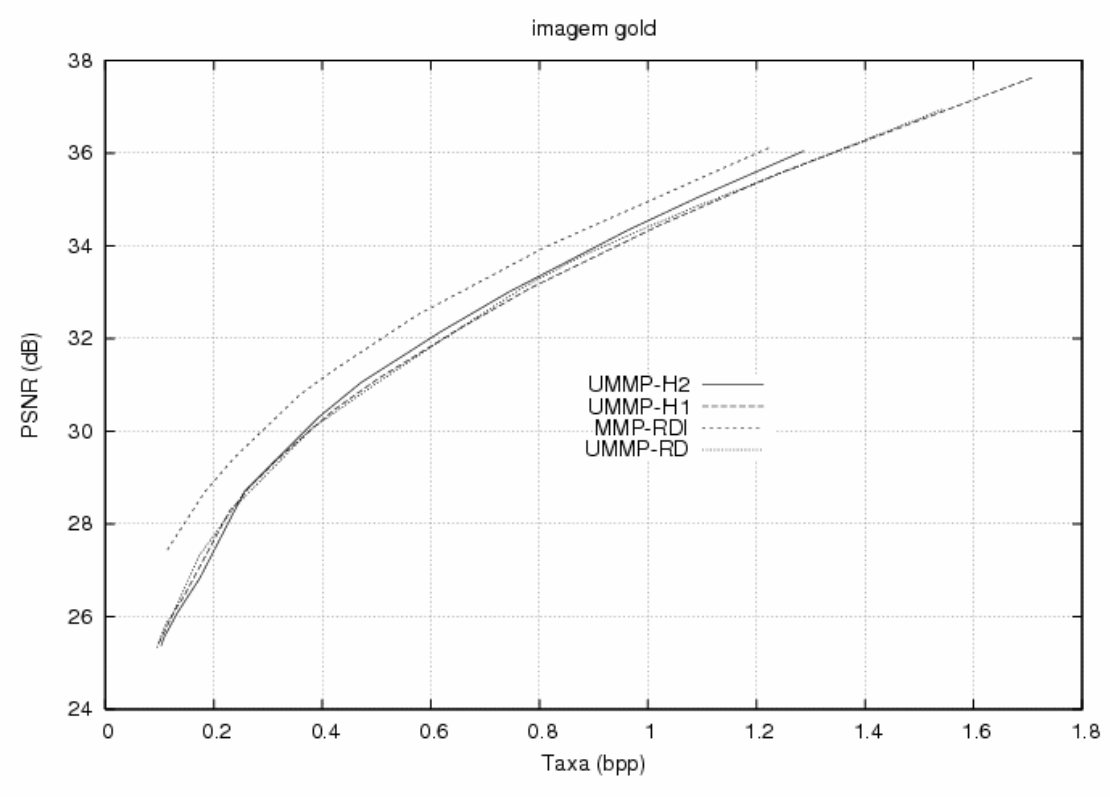


Figura 5.8: Taxa-distorção para *gold* comprimida pelo UMMP-H.

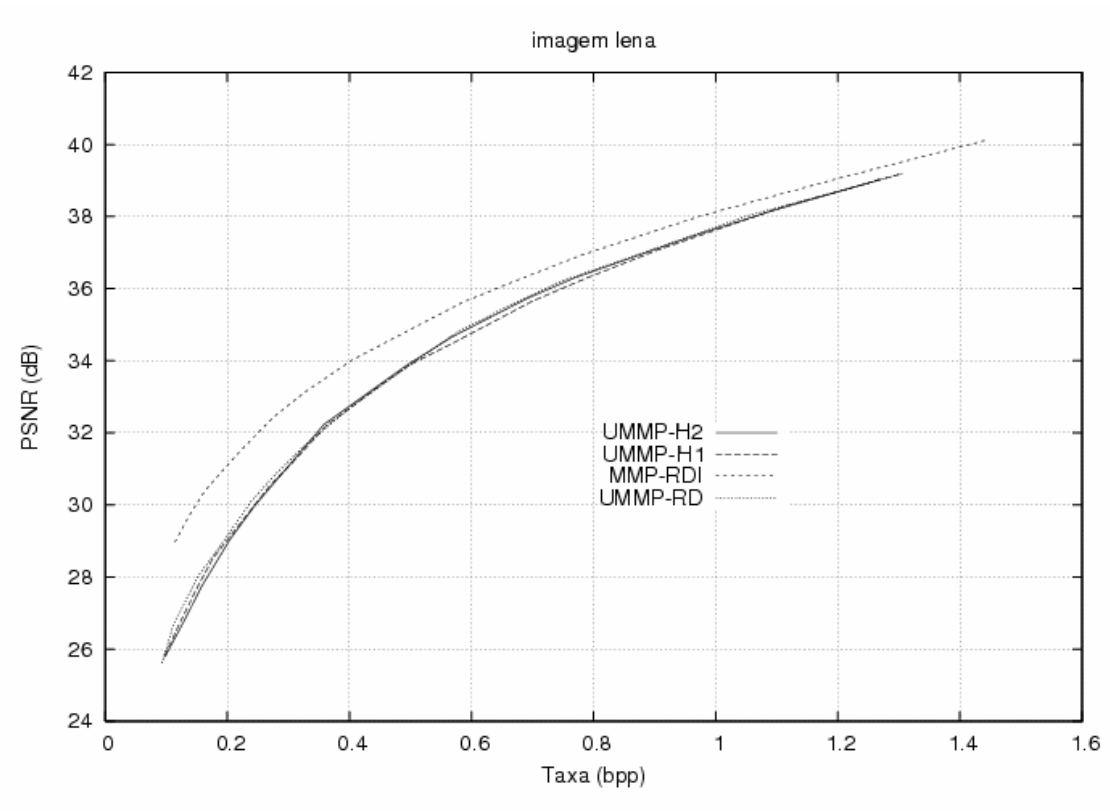


Figura 5.9: Taxa-distorção para *lena* comprimida pelo UMMP-H.

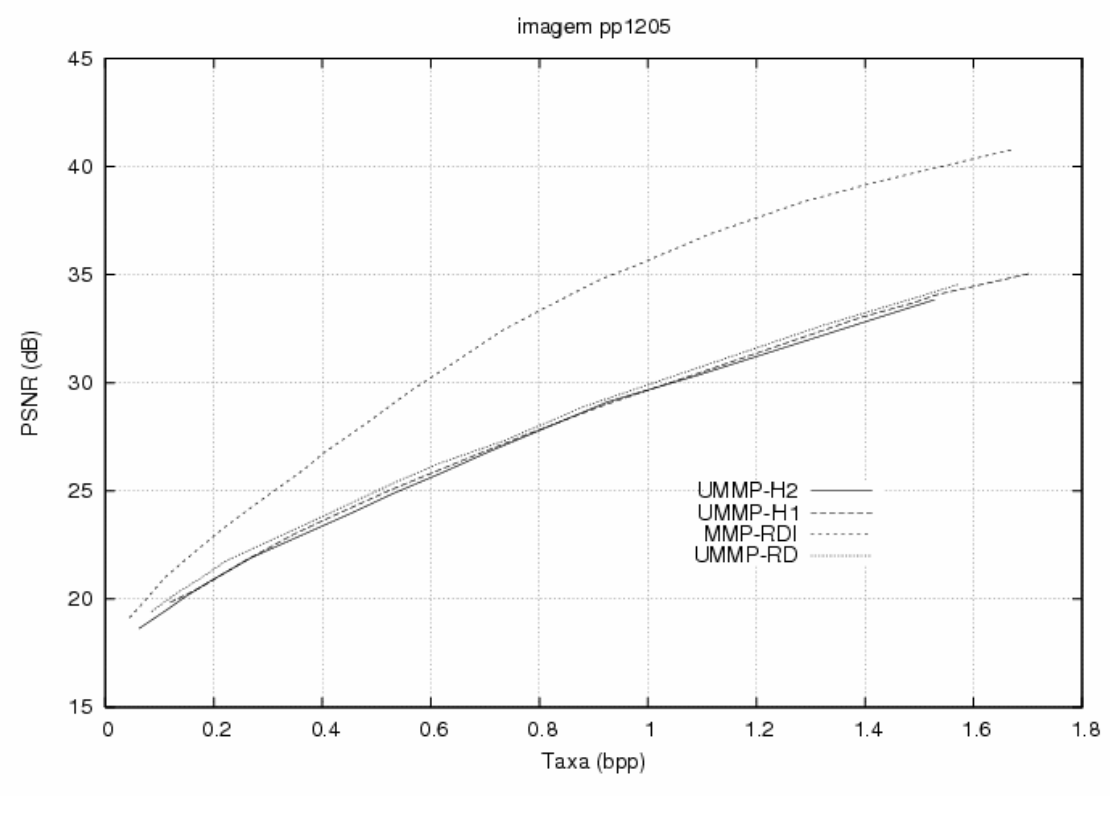


Figura 5.10: Taxa-distorção para *pp1205* comprimida pelo UMMP-H.

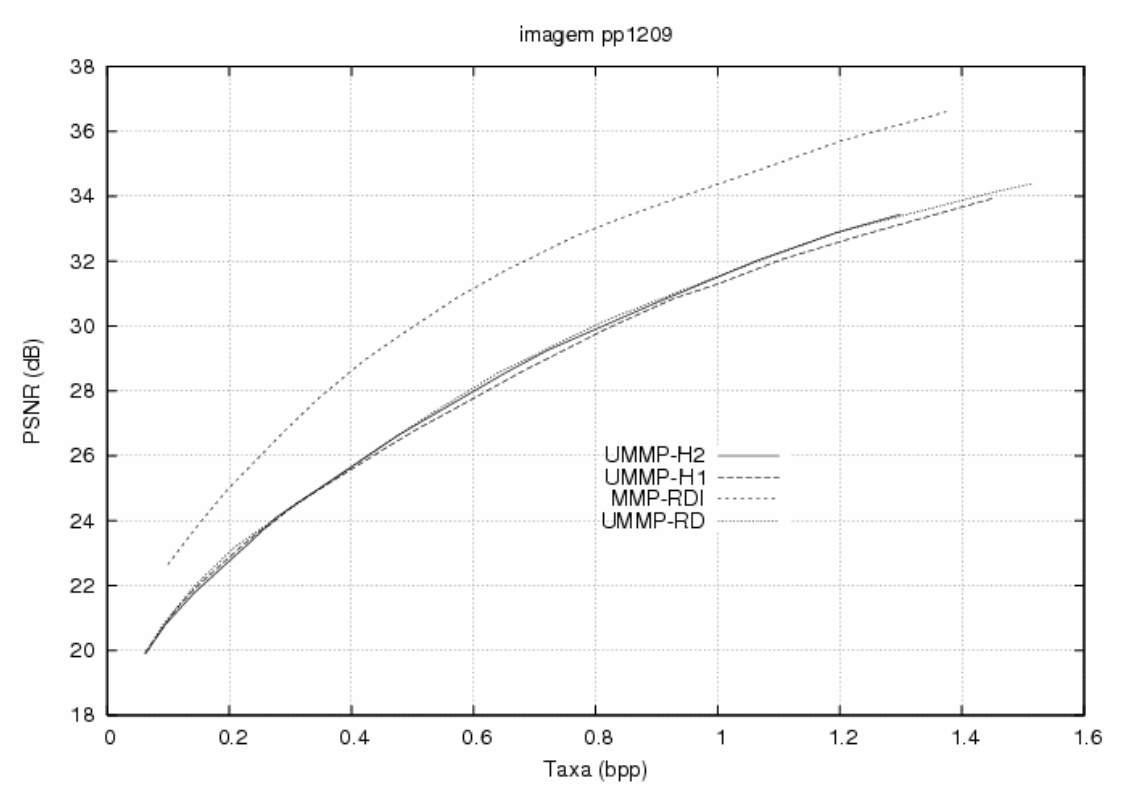


Figura 5.11: Taxa-distorção para *pp1209* comprimida pelo UMMP-H.

De maneira geral, o UMMP-H2 se mostrou um pouco superior ao UMMP-H1, especialmente para as taxas mais altas testadas, com a diferença chegando a 0,4dB a 1,0bpp para a imagem *f-16*. Somente para a imagem de texto (*pp1205*) o UMMP-H1 obteve um desempenho ligeiramente melhor que o UMMP-H2.

Comparando com o UMMP-RD, percebemos que houve um pequeno ganho por parte do UMMP-H2, de até 0,3dB (a 1,1bpp em *bridge*), para as taxas mais altas na maioria das imagens suaves, com o UMMP-RD sendo levemente superior para a imagem *pp1205*. Já para as taxas mais baixas, observamos que a introdução do conceito de codificação por transformada degradou um pouco o desempenho do UMMP-RD.

Ressaltamos, porém, que é de se esperar que a utilização de outras transformadas, como a DCT e DWTs mais sofisticadas, no lugar da Transformada de Haar, gere resultados melhores, inserindo um ganho maior ao desempenho do UMMP-RD.

Finalmente, observamos que somente para a imagem *barbara* nas taxas mais altas conseguimos obter resultados semelhantes, e até um pouco melhores, que o MMP-RDI.

Outra análise que foi feita nos testes se relacionou à quantidade de blocos codificada por cada um dos dois codificadores presentes no algoritmo. Ao todo, uma imagem de 512x512 *pixels* é dividida em 4.096 blocos de tamanho 8x8. A Figura 5.12 mostra o número de blocos UMMP (blocos que o UMMP-RD codificou com menor custo) e o número de blocos Haar (blocos que o Codificador de Haar codificou com menor custo) utilizados em cada teste realizado com o UMMP-H1 para a imagem *lena*. Também são mostrados os números de blocos UMMP e Haar que foram aproximados simplesmente pelo seu nível DC.

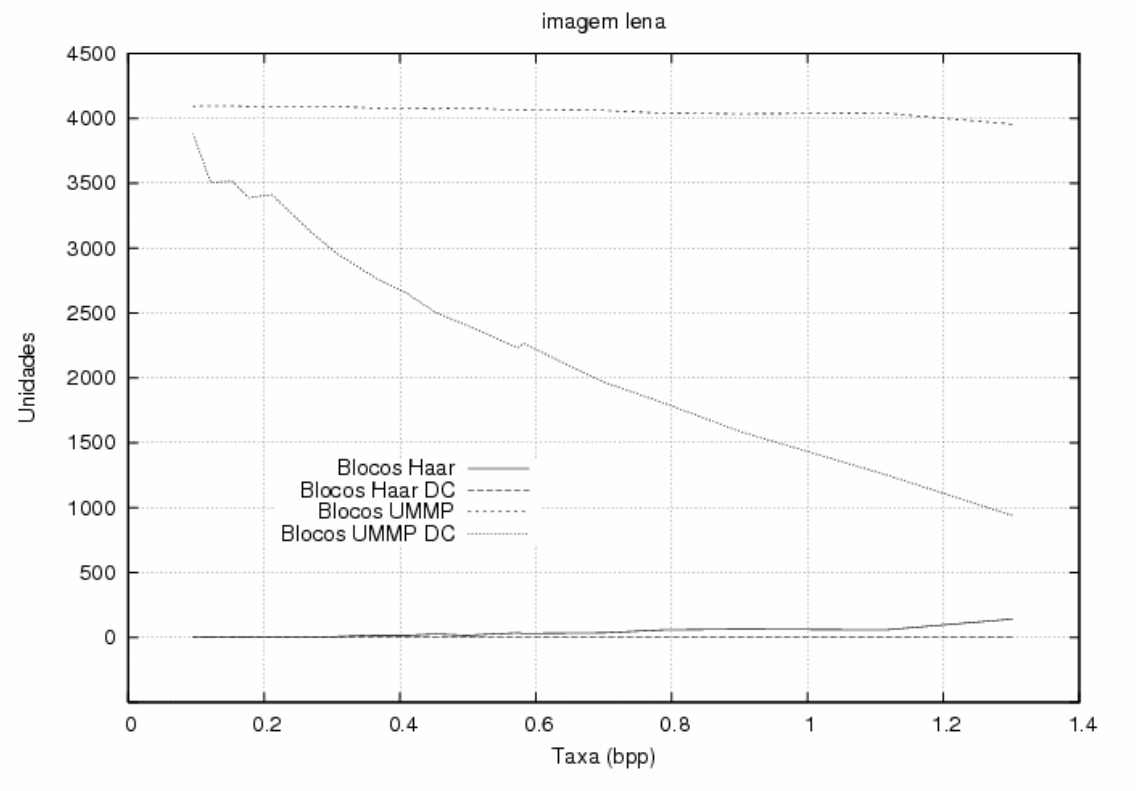


Figura 5.12: Tipos de blocos utilizados para *lena* comprimida pelo UMMP-H1.

Podemos observar que, mesmo para as taxas mais altas, o Codificador de Haar foi utilizado em muito poucos blocos. Isto explica o porquê do UMMP-H1 não ter conseguido resultados melhores que o UMMP-RD. Outro ponto a ser ressaltado é que não foi utilizado nenhum bloco Haar só com nível DC. Isto indica que o Codificador de Haar não foi muito eficiente para codificar somente o nível DC do bloco, fazendo-nos levar à mudança descrita na Seção 5.5 e à criação do UMMP-H2.

A Figura 5.13 mostra a quantidade de cada tipo de bloco utilizada na codificação da imagem *lena* pelo UMMP-H2.

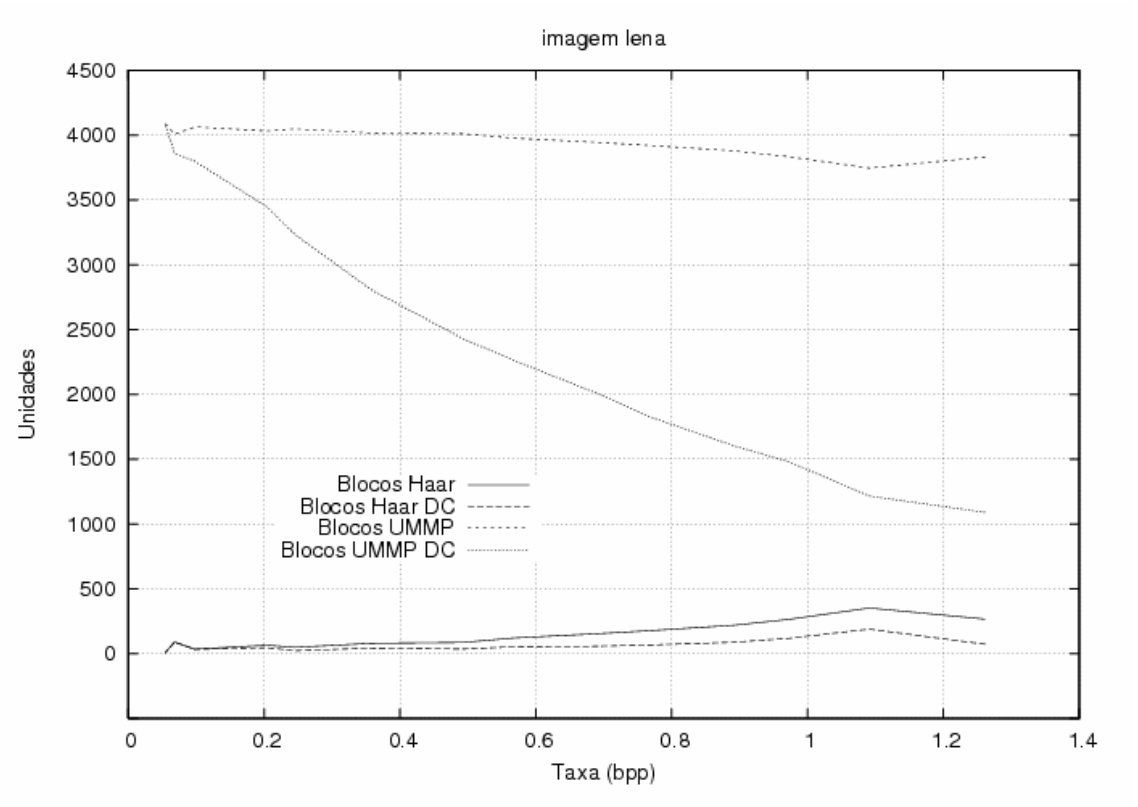


Figura 5.13: Tipos de blocos utilizados para *lena* comprimida pelo UMMP-H2.

Agora, percebemos que o algoritmo passou a utilizar mais o Codificador de Haar, especialmente para os blocos aproximados somente pelo seu nível DC, o que refletiu em um ganho de desempenho em relação ao UMMP-H1. Apesar disso, notamos que, relativamente à quantidade de blocos UMMP, o número de blocos Haar utilizados permaneceu pequeno, o que indica que ainda podemos obter maiores ganhos ao trocarmos o Codificador de Haar por outro que utilize uma transformada mais eficiente, como a DCT e outras DWTs mais sofisticadas. Ressaltamos, por fim, que este comportamento também foi observado para todas as outras imagens testadas.

# Capítulo 6

# Algoritmo MMP-SA

Neste capítulo introduzimos o algoritmo MMP-SA, que incorpora ao MMP [5] a idéia de codificação por aproximações sucessivas. Para isso, primeiramente propomos o algoritmo MMP-SA básico e, a seguir, mais três versões dele com algumas tentativas de obtenção de ganho de desempenho.

## 6.1 Introdução

Como já visto anteriormente, no UMMP, um sinal de entrada é decomposto em um espaço vetorial, sendo representado por uma combinação linear de elementos deste espaço que pertençam a um dicionário. Assim, pode-se dizer que no UMMP faz-se uma aproximação sucessiva do bloco a ser codificado, já que, a cada nova divisão do bloco, sua representação vai sendo melhorada com o acréscimo de uma nova componente.

O conceito básico de aproximações sucessivas é bastante simples. Podemos imaginar um sinal composto pela soma de diversos segmentos. Com o primeiro deles, obtém-se uma aproximação do sinal, e, à medida que os segmentos seguintes vão sendo somados, esta aproximação vai sendo melhorada. Assim, de acordo com o nível de fidelidade desejado, pode-se usar apenas um determinado número de segmentos, economizando-se espaço de armazenamento (ou banda de transmissão). Mais detalhes sobre este conceito podem ser obtidos em [16, 17].

Com isso em mente, propomos incluir também no MMP a idéia de aproximação sucessiva, criando o algoritmo MMP-SA. Este algoritmo codifica o bloco de entrada, utilizando o MMP, com determinada taxa. A seguir, obtém um bloco de erro, que é também codificado pelo MMP, gerando um sinal de erro codificado, que é usado para incrementar a aproximação do bloco original. Este procedimento pode ser repetido até que seja atingido o nível final de fidelidade desejado.

## 6.2 Descrição e implementação do MMP-SA

A aplicação do conceito de aproximações sucessivas ao MMP foi feita da maneira descrita nesta seção. Primeiramente, a imagem de entrada $X$ é codificada pelo MMP-RDI, gerando uma imagem aproximada $\hat{X}$. Esta será a primeira representação da imagem original, já podendo ser utilizada em aplicações que necessitem de uma grande taxa de compressão.

A seguir, obtemos uma segunda aproximação da imagem original, mais fiel a esta e com uma menor taxa de compressão. Para isto, extraímos o erro $R_1$ da primeira aproximação, dado por:

$$R_1 = X - \hat{X} \tag{6.1}$$

Aplicamos, então, o MMP-RDI a este resíduo, gerando o resíduo codificado $\hat{R}_1$, que é usado para incrementar a aproximação da imagem original, que passa a ser dada por:

$$\hat{X}' = \hat{X} + \hat{R}_1 \tag{6.2}$$

Este procedimento pode ser repetido até que seja atingida a fidelidade, ou a taxa, desejada, lembrando que, a cada novo passo executado, o erro da aproximação da imagem original diminui e a taxa de codificação aumenta.

Uma das características deste algoritmo é que, a partir de um *bitstream* gerando pelo MMP-SA com *n* etapas, o decodificador pode obter uma imagem reconstruída com

$m$ aproximações, onde $m \leq n$, ou seja, a representação para uma dada taxa contém todas as representações para as taxas menores.

Essa qualidade pode ser bastante útil em aplicações em que haja a possibilidade de escolha pelo usuário da taxa ou do erro desejados, como, por exemplo [17], em um servidor de imagens de alta qualidade. Nesse caso, seria desejável disponibilizar para os usuários versões das imagens com menor qualidade para rápida navegação pelas imagens disponíveis, já que a diversidade de usuários pode ser grande, com cada um tendo determinada largura de banda. Assim, sem essa característica no codificador usado para gerar as imagens de alta qualidade, seria necessária a existência de diversos arquivos, cada um com uma qualidade diferente, armazenados no servidor, o que geraria uma necessidade bem maior de espaço.

Na implementação do algoritmo MMP-SA feita neste trabalho, a imagem original não mais foi dividida em blocos como nos casos anteriores. Desta vez, a imagem inteira, de dimensão 512x512 *pixels*, foi diretamente codificada pelo MMP-RDI.

Como agora podemos ter blocos de tamanhos até 512x512, o número de dicionários no MMP-RDI seria $N_D = K + L + 1 = 9 + 9 + 1 = 19$. No entanto, a implementação de tal número de dicionários geraria uma grande necessidade de memória, fazendo-nos optar por implementar "apenas" 13 dicionários, os de dimensões 1x1, 2x1, 2x2, ..., 64x32 e 64x64.

Assim, para a aproximação de blocos de dimensões superiores a 64x64 foi utilizado o dicionário $D^{(64,64)}$ expandido para a dimensão requerida. É claro que, nesses casos, perdemos a possibilidade de crescimento diferenciado dos dicionários para as grandes dimensões, o que não deve ter gerado uma grande perda de desempenho, já que as imagens testadas foram rapidamente divididas em blocos de dimensões menores que 64x64, mesmo para baixas taxas. Isso indica que, no momento da aproximação de um bloco maior que 64x64 o seu dicionário ainda estaria praticamente inalterado em relação ao dicionário inicial.

Outro ponto interessante do algoritmo está na utilização de uma só árvore de segmentação durante toda a codificação, ou seja, a codificação do primeiro bloco de resíduo será feita partindo-se da árvore moldada no primeiro passo (codificação da imagem original), e assim sucessivamente. Por exemplo, seja a árvore mostrada na Figura 6.1 o resultado da otimização feita pelo MMP-RDI para a codificação da imagem

original. Na segunda etapa do algoritmo, a codificação do resíduo pelo MMP-RDI será executada a partir dessa árvore, como se cada um dos quatro sub-blocos ($X_3$, $X_9$, $X_{10}$, e $X_2$) fosse codificado separadamente.

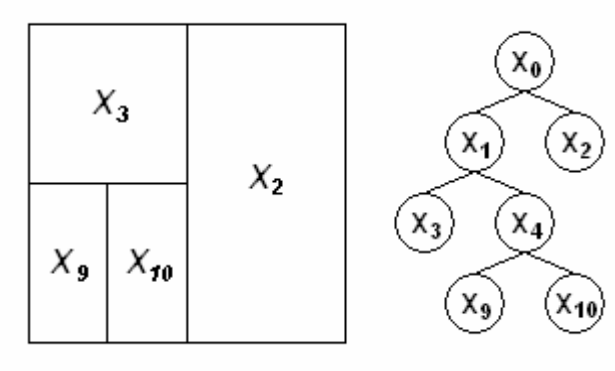


Figura 6.1: Exemplo de divisão de um bloco no MMP-RDI.

Com isso, podemos economizar *bits* indicativos de *flags* de segmentação da árvore, já que, como o bloco de resíduos foi gerado conforme a divisão gerada na aproximação anterior, seria pouco provável que, na nova aproximação, a otimização feita no MMP-RDI gerasse uma árvore menos segmentada.

Além disso, o estado do dicionário também é mantido de um passo para o outro, ou seja, o dicionário inicial de um dado passo é exatamente igual ao dicionário ao final do passo anterior.

Os parâmetros utilizados no algoritmo implementado foram os seguintes:

- o algoritmo MMP-RDI utilizado é o mesmo descrito no Capítulo 2 e no Apêndice A, com a alteração do tamanho máximo do bloco para 512x512;
- deve ser definido o número $N$ de aproximações ( $N \geq 2$ ) a serem executadas pelo algoritmo, sendo uma aproximação da imagem original e *N-1* aproximações de resíduos;
- para cada uma das aproximações feitas, deve-se utilizar um valor de $\lambda$ a ser passado como parâmetro para o MMP-RDI. Assim, deve-se definir uma seqüência decrescente de valores de lambda $\{\lambda_0, \lambda_1, \lambda_2, ..., \lambda_{N-1}\}$ a ser usada na codificação;

- os dicionários iniciais utilizados possuíram 513 elementos, sendo compostos por todos os níveis de cinza possíveis da imagem e seus simétricos, ou seja, $D^{(1,1)} = \{-256, -255,..., -1, 0, 1,..., 255, 256\}$. Os dicionários iniciais nas outras escalas foram obtidos a partir de $D^{(1,1)}$ pela aplicação de transformações de escala;
- todos os dicionários foram mantidos inalterados entre as diferentes etapas de aproximação, ou seja, os dicionários iniciais do MMP-RDI usado em uma aproximação são idênticos aos dicionários existentes ao término da aproximação anterior;
- a rotina $\text{codifica_mmprdi}(X, \lambda)$ engloba as fases da otimização da árvore de segmentação e da codificação propriamente dita.

Assim, considerando $X$ a imagem a ser codificada, o algoritmo foi implementado da seguinte forma:

**Procedimento** $\hat{X} = \text{codifica}(X)$

**Passo 1:** Faz $\lambda = \lambda_0$.

**Passo 2:** Codifica, utilizando o algoritmo MMP-RDI, a imagem de entrada, ou seja, obtém $\hat{X}_0 = \text{codifica_mmprdi}(X, \lambda)$.

**Passo 3:** Obtém o resíduo $R_1 = X - \hat{X}_0$.

**Passo 4:** Faz $n = 1$.

**Passo 5:** Faz $\lambda = \lambda_n$.

**Passo 6:** Codifica, utilizando o algoritmo MMP-RDI, o resíduo $R_n$, ou seja, obtém $\hat{R}_n = \text{codifica_mmprdi}(R_n, \lambda)$.

**Passo 7:** Obtém o resíduo $R_{n+1} = R_n - \hat{R}_n$.

**Passo 8:** Incrementa $n$ e, se $n < N$, vai para o **Passo 5**.

**Passo 9:** Retorna $\hat{X} = \hat{X}_0 + \sum_{i=1}^{N-1} \hat{R}_i$.

A Figura 6.2 mostra os resultados obtidos ao comprimirmos a imagem *lena* com o MMP-SA utilizando duas seqüências distintas de valores de lambda, a primeira delas com quatro valores, $Seq1 = \{1000, 500, 250, 100\}$, e a segunda com seis, $Seq2 = \{2000, 1000, 500, 250, 100, 50\}$.

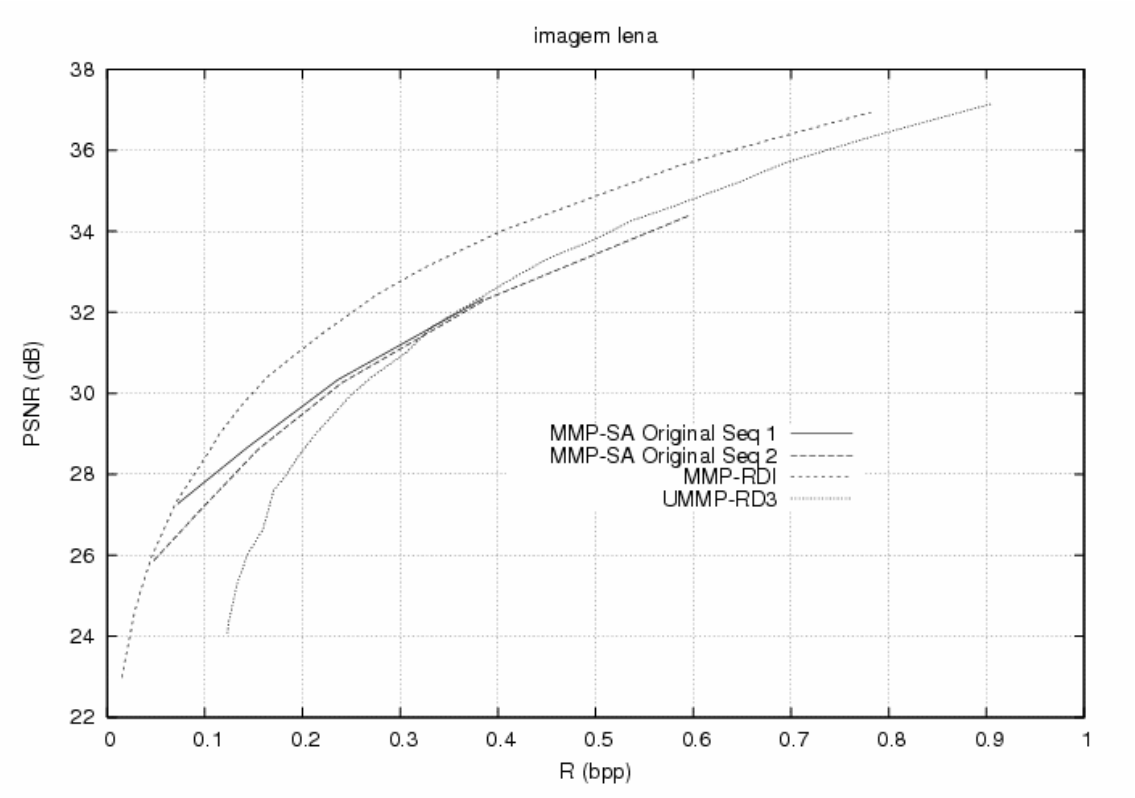


Figura 6.2: Taxa-distorção para *lena* comprimida pelo MMP-SA.

Pelo gráfico, notamos uma perda de desempenho em relação ao MMP-RDI. Porém, percebemos também que houve uma diferença razoável, especialmente para taxas mais baixas, entre os resultados obtidos com cada seqüência de valores de lambda, o que indica que o desempenho do algoritmo ainda pode ser incrementado com uma melhor escolha destes valores. Um teste mais aprofundado relacionado a este tema será desenvolvido na Seção 6.6.

## 6.3 MMP-SA2

Como visto anteriormente, o MMP-SA faz sucessivas aproximações da imagem original, sendo que, a partir da segunda delas, ele basicamente codifica resíduos. Assim, como é esperado que blocos de resíduos possuam boa parte dos coeficientes com baixas amplitudes, pode ser interessante usarmos um dicionário inicial que reflita melhor esta característica. Por isso, essa versão do MMP-SA utiliza um dicionário inicial diferente.

O dicionário inicial $D^{(1,1)}$, de dimensão 1x1, foi implementado com base na Equação 6.3, sendo $q$ a diferença entre as amplitudes $\left|D_i^{(1,1)}\right|$ de dois elementos consecutivos do dicionário [18].

$$\begin{cases} q = 2, & se\ \left|D_i^{(1,1)}\right| \leq 10 \\ q = 4, & se\ 10 \leq \left|D_i^{(1,1)}\right| \leq 22 \\ q = 8, & se\ 22 \leq \left|D_i^{(1,1)}\right| \leq 86 \\ q = 13, & se\ \left|D_i^{(1,1)}\right| > 86 \end{cases} \tag{6.3}$$

Os dicionários das outras dimensões foram obtidos a partir de $D^{(1,1)}$ por meio da adequada transformação de escala. Os demais aspectos da implementação são os mesmos utilizados no algoritmo original.

Ressaltamos que, como no primeiro passo codificamos a imagem original, não há a necessidade de o dicionário possuir elementos com valores negativos, já que nenhum deles será encontrado na imagem original. Assim, poderia ser interessante inserirmos tais elementos somente a partir do segundo passo. Entretanto, percebemos nos testes que a presença dos elementos negativos no dicionário desde o início da execução do algoritmo provocou perdas desprezíveis.

Codificamos a imagem *lena* com o MMP-SA2 utilizando as mesmas seqüências de lambdas da seção anterior. A Figura 6.3 mostra os resultados obtidos no teste, comparando-os, também, com os resultados do algoritmo original.

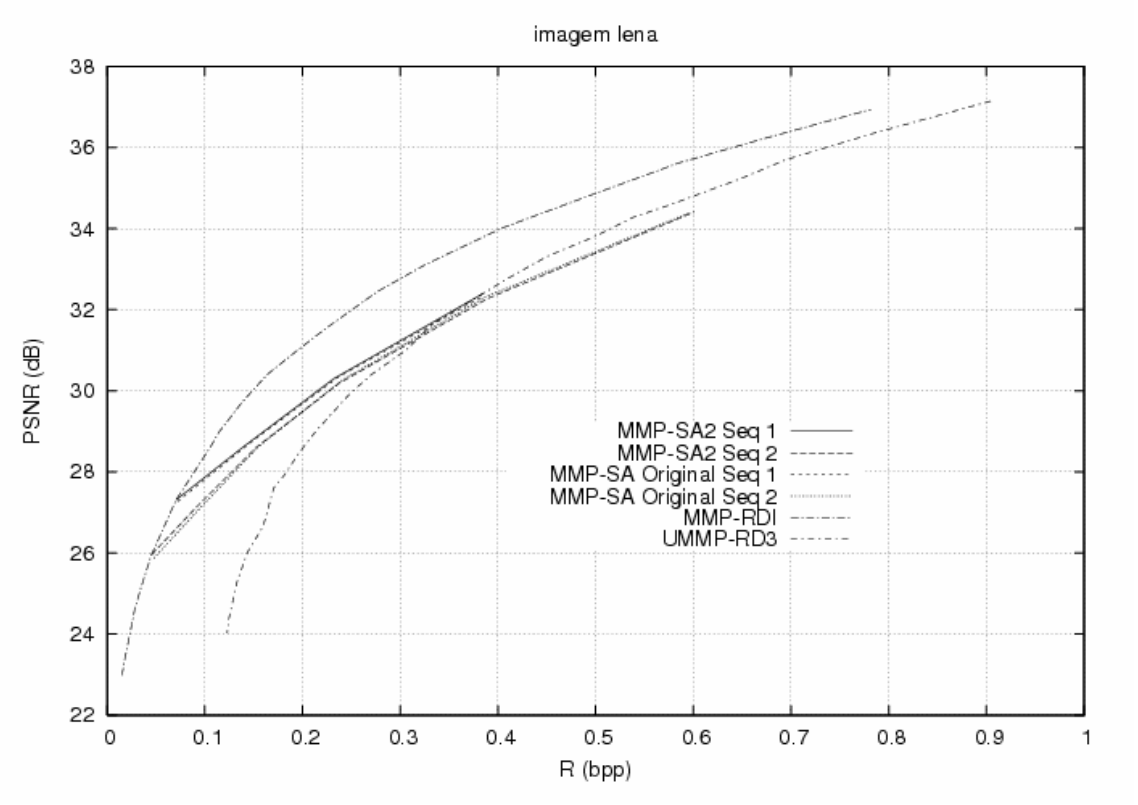


Figura 6.3: Taxa-distorção para *lena* comprimida pelo MMP-SA2.

Podemos notar que a utilização do dicionário inicial descrito nesta seção proporcionou um pequeno ganho de desempenho ao MMP-SA. A diferença entre os resultados obtidos com as duas seqüências persiste.

## 6.4 MMP-SA3

A terceira versão do MMP com aproximações sucessivas, implementada neste trabalho, tenta tirar proveito de uma outra característica que o algoritmo possui. Como já foi dito anteriormente, o MMP-SA utiliza apenas uma árvore de segmentação durante toda a codificação, ou seja, ao iniciar uma nova etapa de aproximação de um bloco de resíduo, o algoritmo MMP-RDI parte da árvore moldada na etapa anterior. Também já foi visto que, a partir da segunda etapa de aproximação, o MMP-RDI codifica apenas blocos de resíduos, cujos coeficientes serão, com maior probabilidade, próximos de zero.

Analisando essas duas características conjuntamente, podemos supor que a seguinte situação deve ocorrer com certa freqüência: no início de uma nova etapa de aproximação, o otimizador do MMP-RDI opta por não dividir o bloco atual e aproximá-lo com o elemento nulo do dicionário. Como exemplo, retornemos à árvore da Figura 6.1, supondo que ela seja a árvore inicial de uma nova etapa de aproximação. O otimizador do MMP-RDI verificará, baseado no custo *lagrangeano*, se é melhor manter ou partir o nó $X_3$. Como tal nó representa um bloco de resíduo, ele tem boa chance de poder ser fielmente representado por um bloco de zeros, caso em que o otimizador certamente optará por não dividi-lo e aproximá-lo pelo elemento nulo do dicionário de dimensão apropriada.

Propomos, então, a criação de um símbolo específico para indicar ao decodificador esta decisão do otimizador. Agora, o novo símbolo pode substituir o par de símbolos (*flag* indicativo da não partição do nó e índice do elemento nulo do dicionário) usados no algoritmo original para essa situação.

Com isso, temos três *flags* de segmentação, indicando as três possíveis decisões do otimizador quanto a um nó da árvore: dividi-lo (*flag* '0'), mantê-lo (*flag* '1', que deve ser seguido por um índice de elemento do dicionário) e aproximá-lo diretamente pelo bloco nulo (*flag* '2').

Ressaltamos que o bloco nulo continua fazendo parte dos dicionários, e que o esquema original ainda pode ser utilizado pelo algoritmo caso gere um custo menor que o do novo esquema, ou seja, se o custo do *flag* '1' somado ao custo de codificação do índice do elemento nulo for menor que o custo do *flag* '2'.

Os testes foram feitos com a imagem *lena*, que foi codificada pelo MMP-SA3 com as mesmas duas seqüências de lambdas utilizadas nos testes anteriores e com o mesmo dicionário inicial usado no MMP-SA original. Os resultados são mostrados na Figura 6.4.

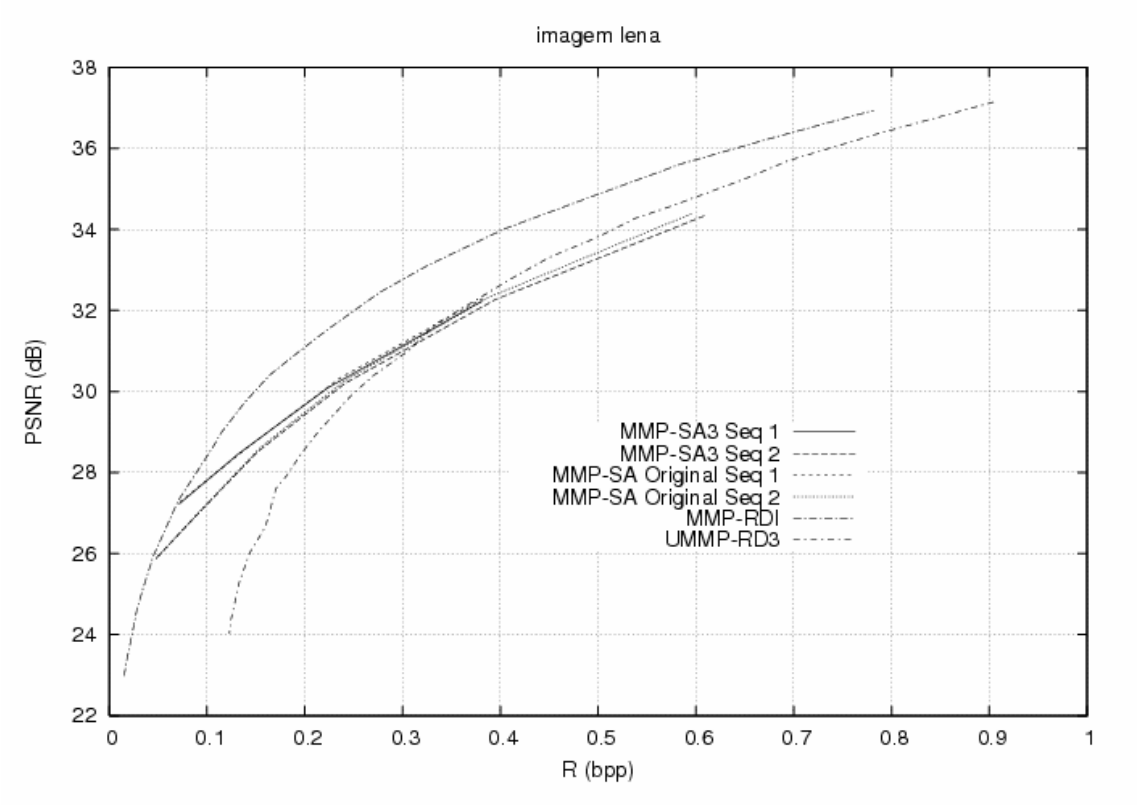


Figura 6.4: Taxa-distorção para *lena* comprimida pelo MMP-SA3.

O gráfico nos mostra que o MMP-SA3 obteve um desempenho equivalente ao do algoritmo original para as taxas mais baixas e foi inferior para as taxas mais altas testadas. Este resultado nos mostra que, pelo menos para a imagem testada e com as seqüências de valores de lambdas utilizadas, a inovação do MMP-SA3 em relação ao MMP-SA original não obteve êxito, nem mesmo para as taxas mais baixas, casos em que se espera que a situação descrita no início desta seção ocorra mais freqüentemente. Ressaltamos, porém, que somente com testes mais diversificados poderíamos afirmar algo mais conclusivo sobre o MMP-SA3.

## 6.5 MMP-SA4

A última versão do MMP-SA implementada neste trabalho simplesmente junta em um mesmo algoritmo as inovações propostas no MMP-SA2 e no MMP-SA3. Assim, o MMP-SA4 utiliza o terceiro *flag* de segmentação, conforme descrito na Seção 6.4, e tem seu dicionário inicial de acordo com o mostrado na Seção 6.3.

O algoritmo utilizou as mesmas seqüências de valores de lambda usadas nos testes anteriores para comprimir a imagem *lena*. Os resultados obtidos são mostrados no gráfico da Figura 6.5.

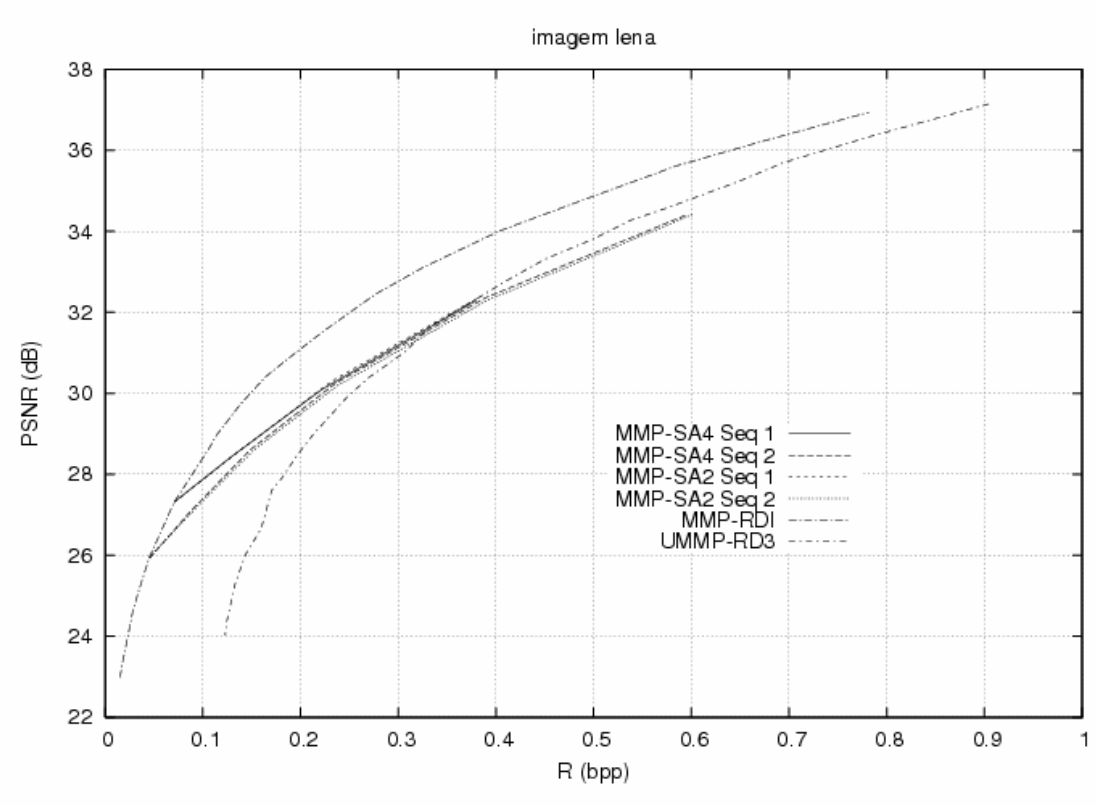


Figura 6.5: Taxa-distorção para *lena* comprimida pelo MMP-SA4.

Comparamos o MMP-SA4 com o MMP-SA2 porque este era o de melhor desempenho até agora. Percebemos que ambos obtiveram desempenho bastante semelhante, com um pequeno ganho do MMP-SA4. Interessante notar que, agora, a idéia descrita na Seção 6.4, que é a diferença entre o MMP-SA4 e o MMP-SA2, gerou algum ganho de desempenho, o que não havia ocorrido com o MMP-SA3 em relação ao MMP-SA original. Isto pode ser explicado pelo fato de o dicionário inicial não possuir mais todas as amplitudes do sinal de entrada, o que nos faz pensar que a situação que favorece o MMP-SA4 (e o MMP-SA3) ocorre mais freqüentemente.

Assim, demonstramos aquilo que foi afirmado na seção anterior, ou seja, que qualquer conclusão definitiva acerca do desempenho do MMP-SA3 em relação ao algoritmo original (e, analogamente, do MMP-SA4 em relação ao MMP-SA2) ainda é bastante prematura.

## 6.6 Avaliação acerca da seqüência de lambdas

Como pudemos observar nos resultados obtidos nas seções anteriores, o desempenho do MMP-SA é influenciado pela escolha da seqüência de valores de lambda. Assim, nesta seção, procuramos buscar uma boa heurística para obtermos seqüências que melhorem o desempenho do algoritmo.

Inicialmente, podemos destacar duas características que esperamos encontrar nos resultados obtidos com o MMP-SA, independentemente de sua versão. A primeira, e mais trivial delas, é que a curva do MMP-SA sempre se inicia a partir de um ponto da curva do MMP-RDI, já que a sua primeira etapa é simplesmente a aplicação do MMP-RDI à imagem original. Vale lembrar, porém, que o MMP-SA não divide previamente a imagem em blocos, com é usualmente feito no MMP-RDI, o que pode gerar algum pequeno desvio dessa característica nos resultados aqui apresentados.

A outra propriedade esperada do algoritmo é que, à medida que aumentamos o primeiro valor de lambda, o resultado obtido com o segundo valor se aproxima da curva R-D do MMP-RDI. Isto pode ser explicado pelo fato de que, quanto maior for o primeiro lambda utilizado, o resultado do primeiro passo da codificação da imagem original vai se tornando mais precário, gerando um sinal de erro cada vez mais parecido com a imagem original. Como o segundo passo é a mera codificação do resíduo pelo MMP-RDI com o segundo valor de lambda, e este resíduo é bem próximo da imagem original, o resultado deste passo se aproxima da curva do MMP-RDI.

Tomando por base essas duas características do algoritmo, buscamos executar diversos testes com seqüências de valores de lambdas com propriedades variadas, a fim de tentarmos caracterizar quais tipos de seqüências geram melhores resultados. Para isso, utilizamos a versão 4 do MMP-SA, de acordo com o descrito na Seção 6.5, já que foi a que obteve resultados ligeiramente melhores. A imagem *lena* foi utilizada em todos o testes feitos nesta seção.

Inicialmente, optamos por testar seqüências com apenas dois valores de lambda, para verificarmos, especialmente, se é melhor que os dois valores sejam mais próximos ou mais afastados.

Foram testados 64 diferentes pares de valores de lambda, sendo que, para o primeiro deles, foram usados 9 valores distintos, a saber: 500, 1000, 2500, 5000, 10000, 20000, 40000, 80000 e 150000. A Tabela 6.1 relaciona os pares utilizados

Tabela 6.1: Pares de lambdas usados nos testes do MMP-SA.

| Primeiro Valor | Segundos Valores |
|---|---|
| 500 | 250, 100 e 50 |
| 1000 | 500, 250, 100 e 50 |
| 2500 | 1000, 500, 250, 100 e 50 |
| 5000 | 2500, 1000, 500, 250, 100 e 50 |
| 10000 | 8000, 5000, 1000, 500, 250, 100 e 50 |
| 20000 | 15000, 10000, 8000, 5000, 1000, 500, 250, 100 e 50 |
| 40000 | 15000, 10000, 8000, 5000, 1000, 500, 250, 100 e 50 |
| 80000 | 30000, 15000, 10000, 8000, 5000, 1000, 500, 250, 100 e 50 |
| 150000 | 80000, 30000, 15000, 10000, 8000, 5000, 1000, 500, 250, 100 e 50 |

Os resultados das simulações encontram-se nas Figuras 6.6 a 6.14, onde cada gráfico representa os testes que tiveram pares iniciados com um dos oito primeiros valores de lambda. O resultado individual de cada par é representado por um ponto localizado na coordenada *(R,D)* correspondente ao resultado obtido na segunda etapa do algoritmo, ou seja, para o segundo valor do par de lambdas. Como os valores obtidos na primeira etapa são iguais para todos os testes com determinado lambda inicial, este ponto só é mostrado uma vez em cada gráfico e, como já visto, deve encontrar-se sobre a curva do MMP-RDI.

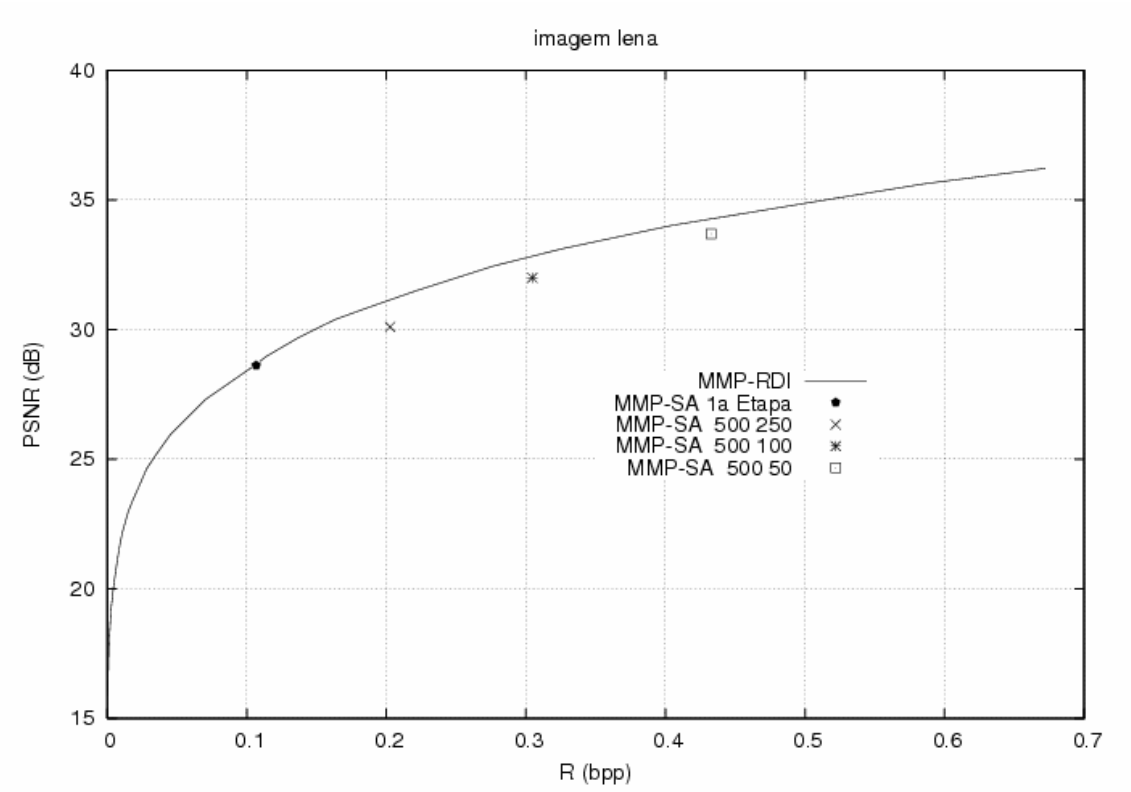


Figura 6.6: Resultados do MMP-SA para os pares *{500, x}*.

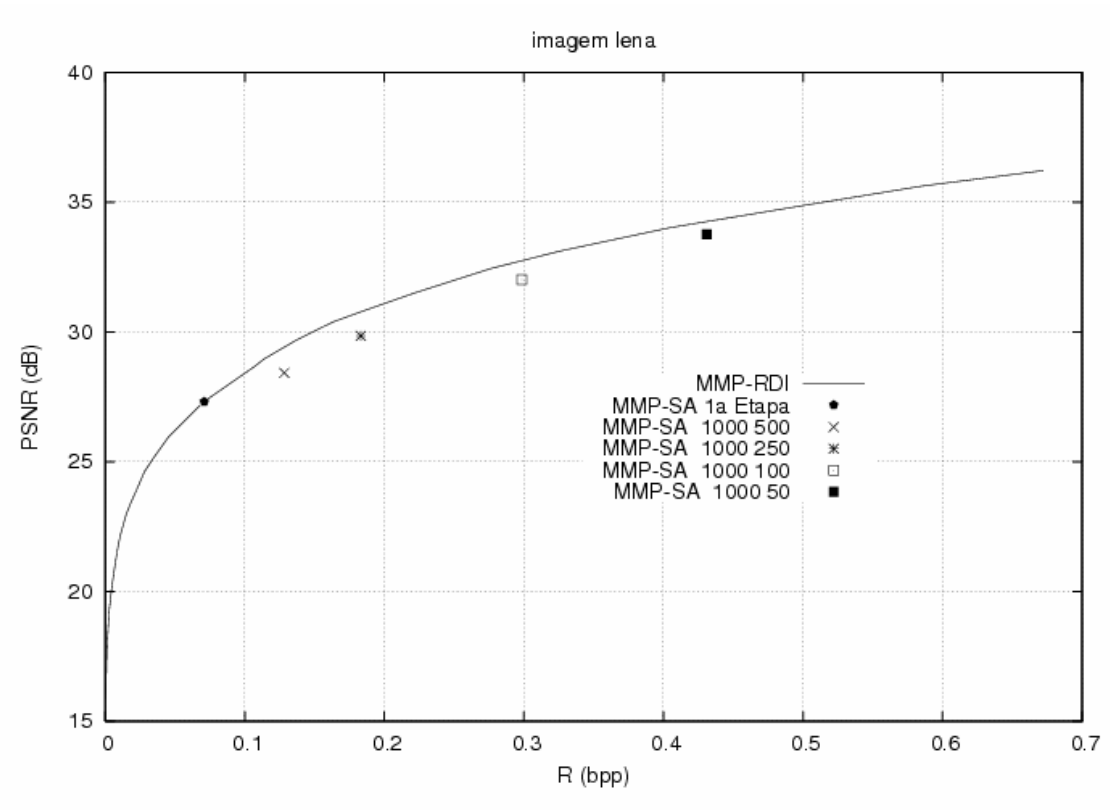


Figura 6.7: Resultados do MMP-SA para os pares *{1000, x}*.

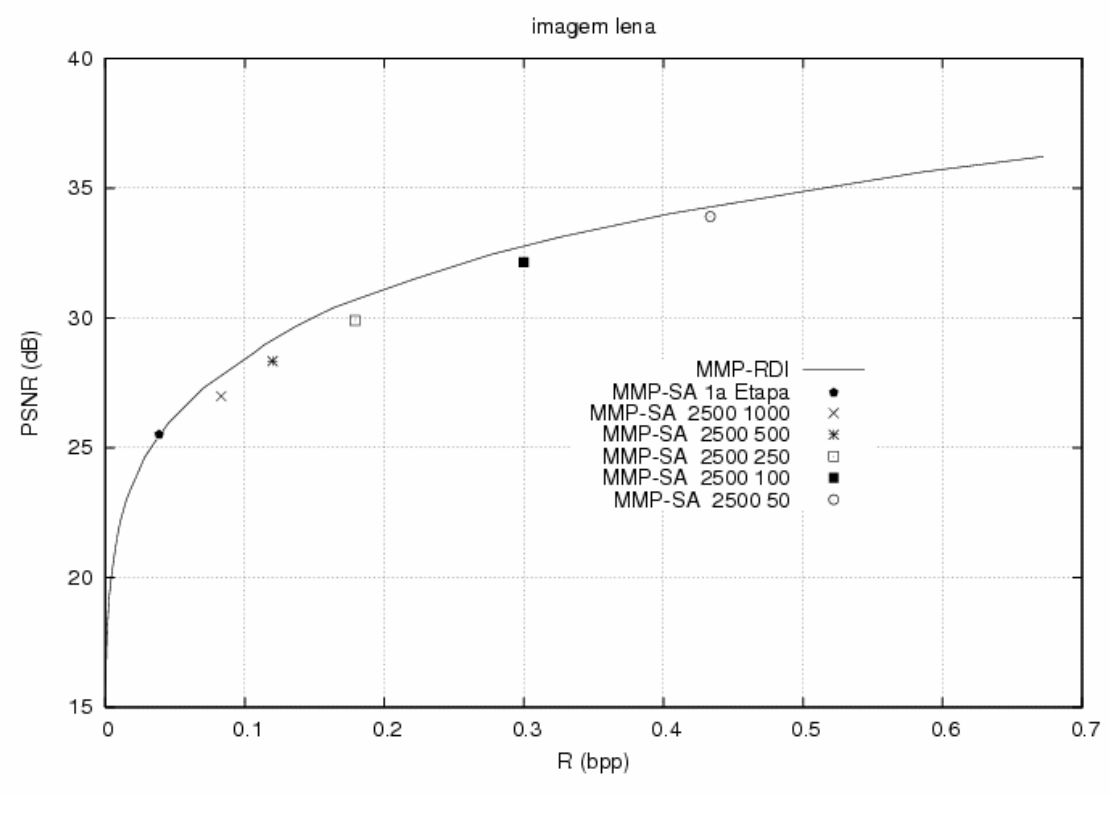


Figura 6.8: Resultados do MMP-SA para os pares *{2500, x}*.

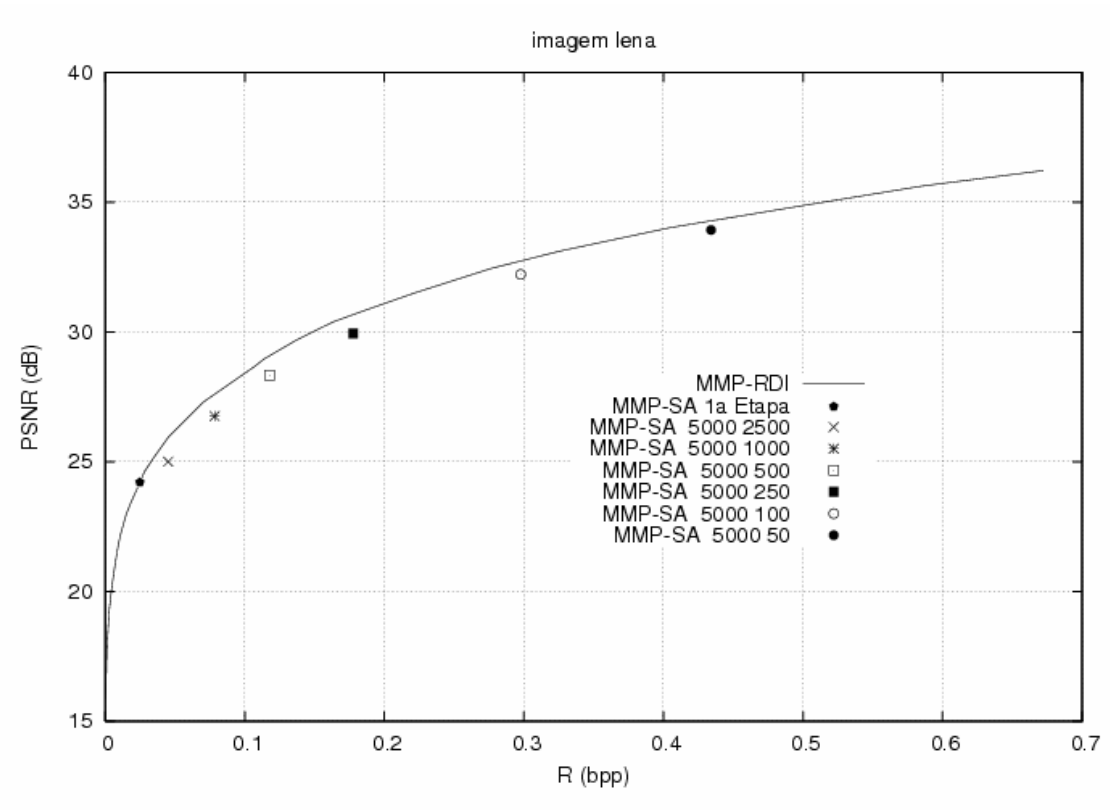


Figura 6.9: Resultados do MMP-SA para os pares *{5000, x}*.

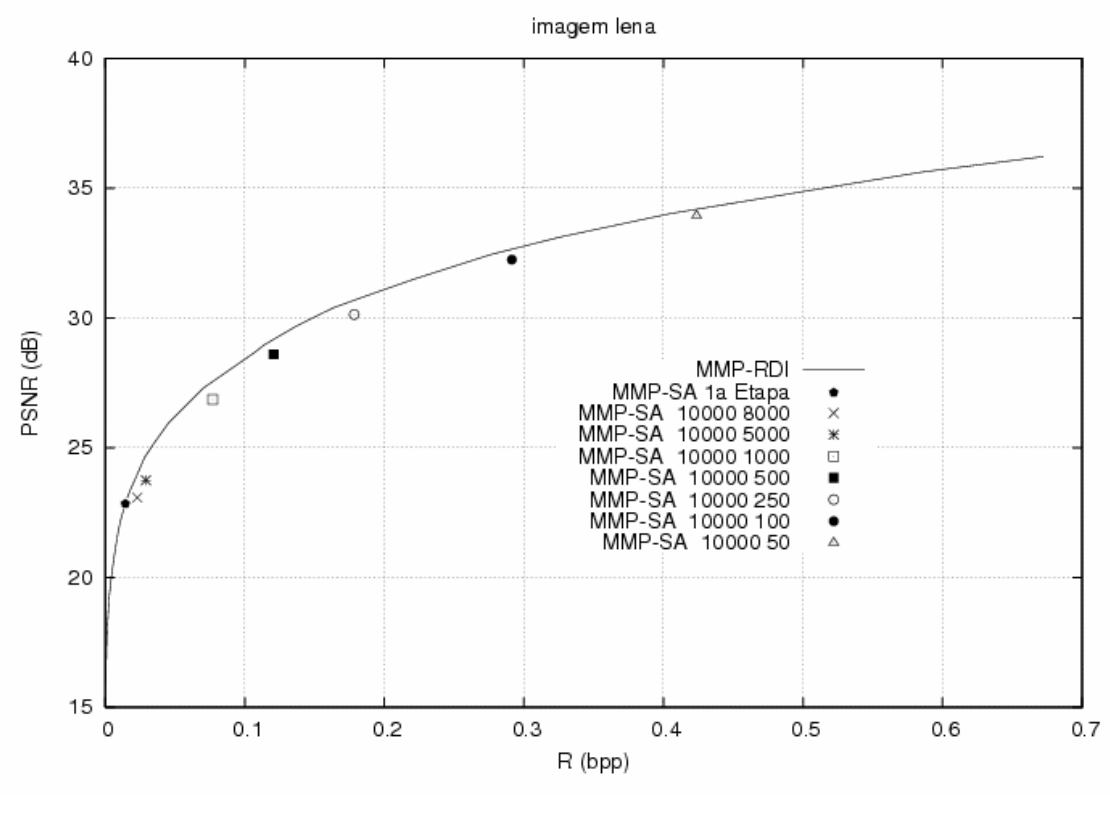


Figura 6.10: Resultados do MMP-SA para os pares *{10000, x}*.

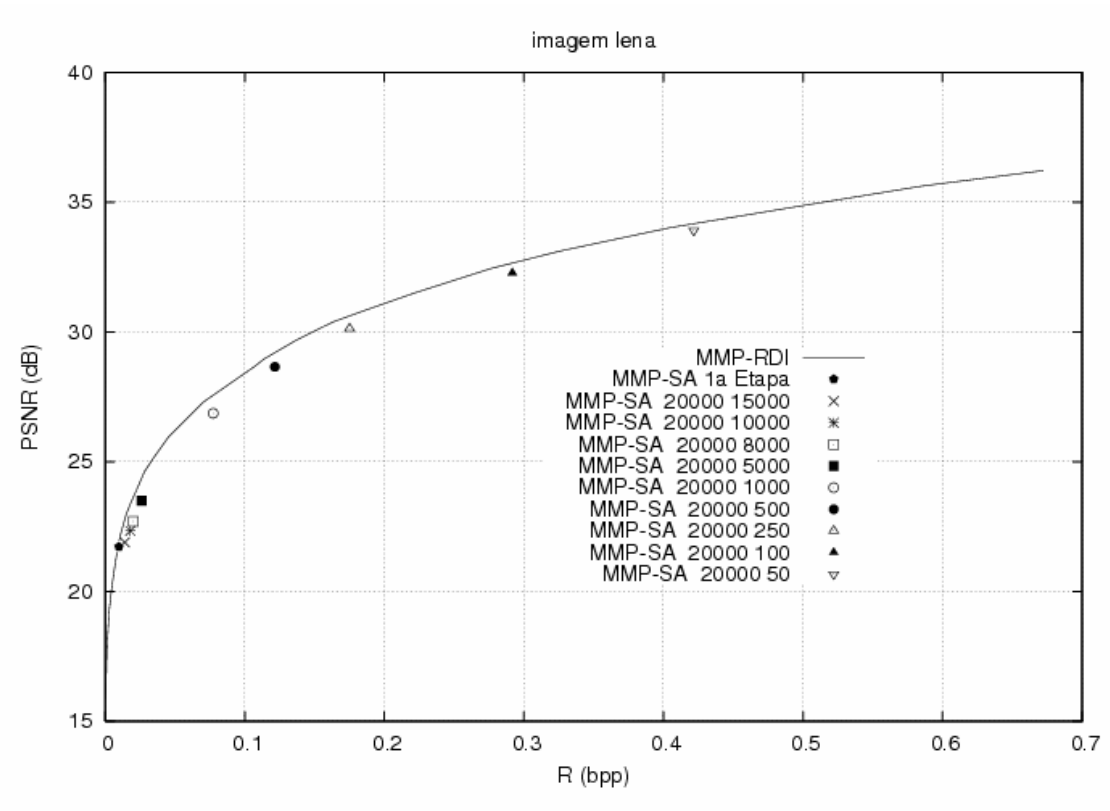


Figura 6.11: Resultados do MMP-SA para os pares *{20000, x}*.

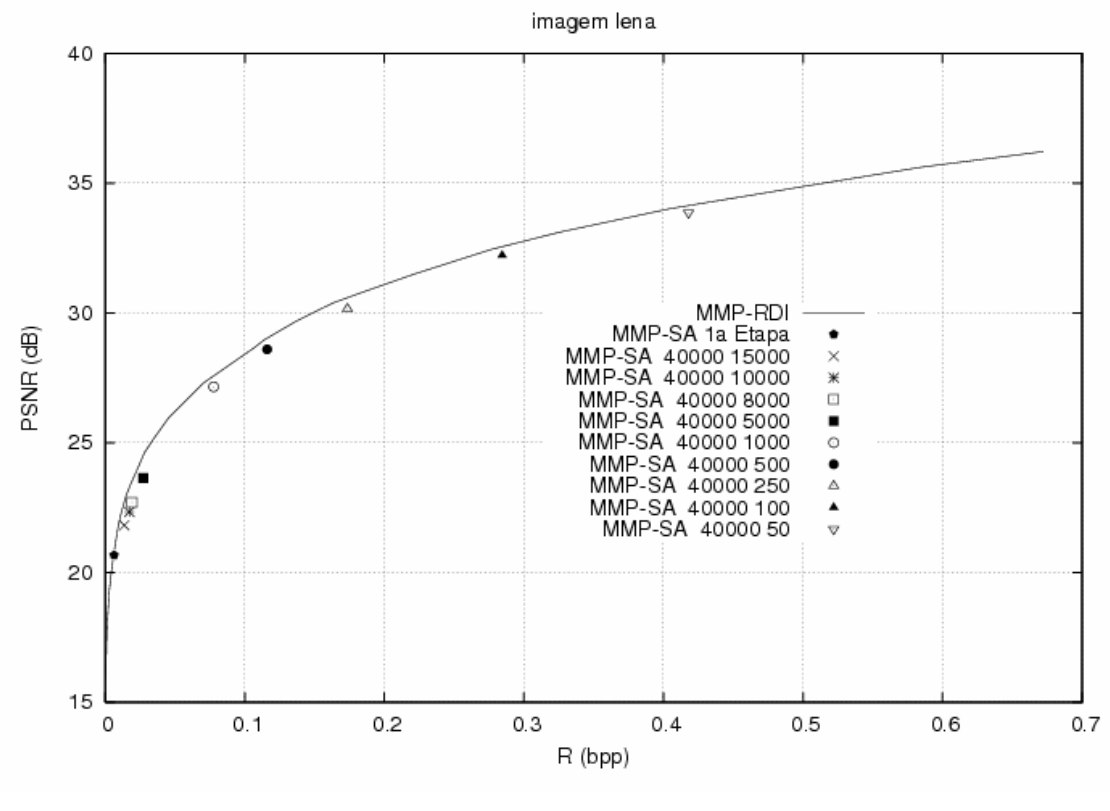


Figura 6.12: Resultados do MMP-SA para os pares *{40000, x}*.

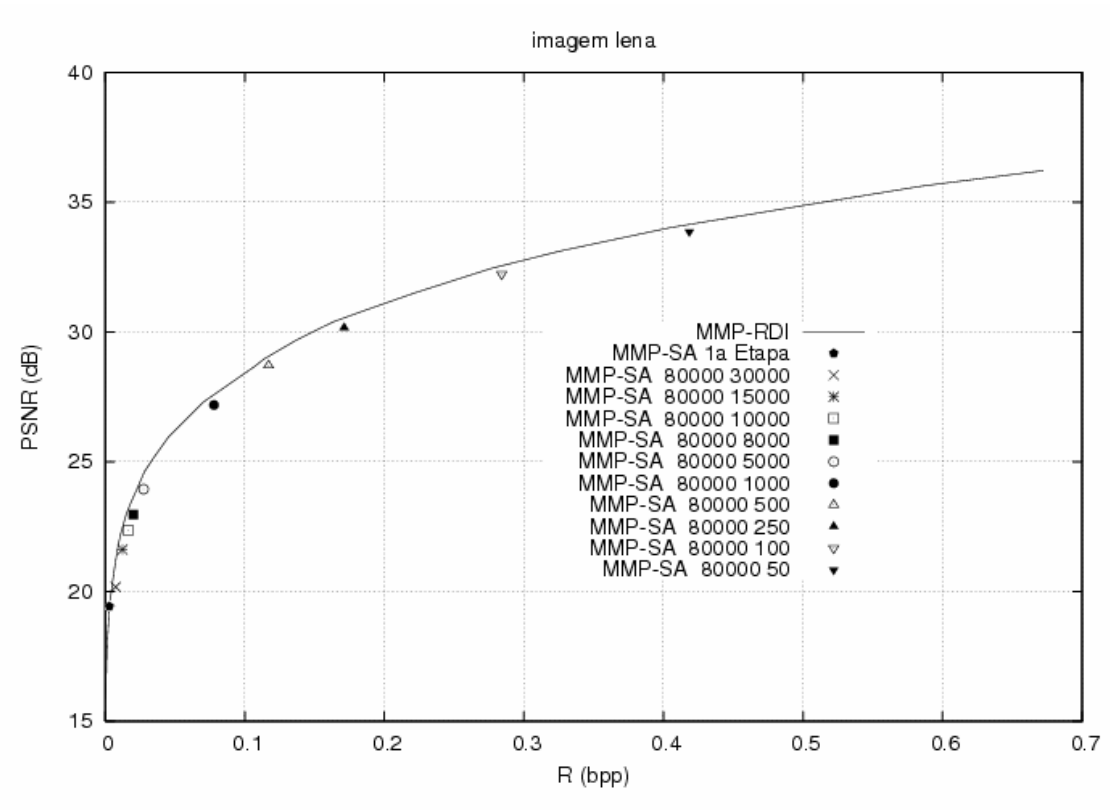


Figura 6.13: Resultados do MMP-SA para os pares *{80000, x}*.

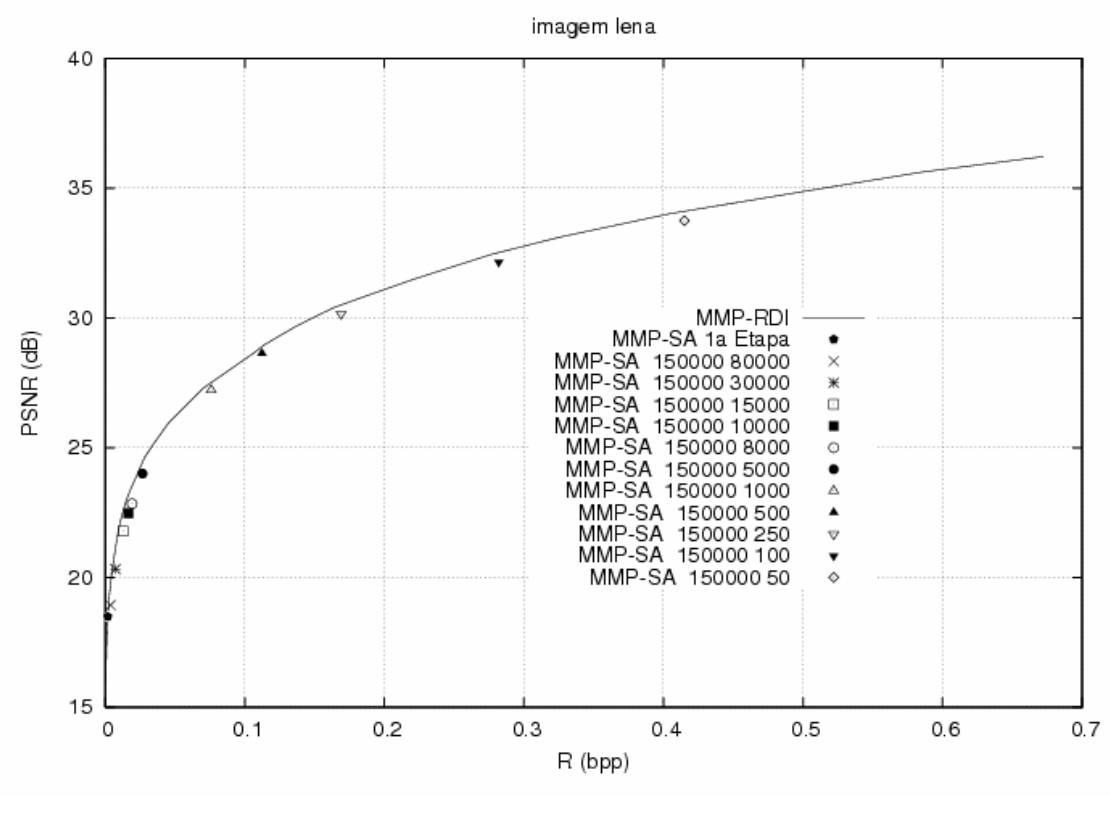


Figura 6.14: Resultados do MMP-SA para os pares *{150000, x}*.

Observando os gráficos mostrados, podemos notar que ambas as características esperadas, citadas anteriormente, se fizeram presentes nos resultados. Realmente, o ponto referente à primeira etapa do algoritmo ficou sobre a curva do MMP-RDI em todos os testes, além disso, os pontos referentes à segunda etapa se aproximaram da curva à medida que o primeiro valor de lambda foi crescendo, como mostra a Tabela 6.2, que apresenta as perdas dos pontos obtidos na segunda etapa do MMP-SA em relação ao MMP-RDI para três taxas distintas.

Tabela 6.2: Perdas do MMP-SA em relação ao MMP-RDI.

| **Primeiro Lambda** | **0,4 bpp** | **0,3 bpp** | **0,2 bpp** |
|---|---|---|---|
| 500 | 0,58 dB | 0,84 dB | 1,04 dB |
| 1000 | 0,50 dB | 0,73 dB | 0,93 dB |
| 2500 | 0,37 dB | 0,62 dB | 0,79 dB |
| 5000 | 0,36 dB | 0,52 dB | 0,72 dB |
| 10000 | 0,26 dB | 0,41 dB | 0,55 dB |
| 20000 | 0,26 dB | 0,38 dB | 0,49 dB |
| 40000 | 0,28 dB | 0,34 dB | 0,43 dB |
| 80000 | 0,28 dB | 0,32 dB | 0,38 dB |
| 150000 | 0,27 dB | 0,33 dB | 0,34 dB |

Outra conclusão que podemos tirar da Tabela 6.2 é que os pontos obtidos para as taxas maiores ficaram mais próximos da curva do MMP-RDI, ou seja, os segundos passos com valores de lambda menores, que geram taxas maiores, tiveram um desempenho mais próximo do MMP-RDI. Assim, pelo menos com base nos dados obtidos até aqui, podemos concluir que um par de valores onde o segundo é bem menor que o primeiro gera melhores resultados para o MMP-SA.

Notamos, ainda, que, em nenhum dos casos testados, obtivemos um ponto acima da curva do MMP-RDI, ou seja, não conseguimos resultados melhores que os do MMP-RDI.

Por fim, juntamos todos os resultados em um só gráfico, mostrado na Figura 6.15. Somente são mostrados os pontos relativos ao segundo passo do algoritmo.

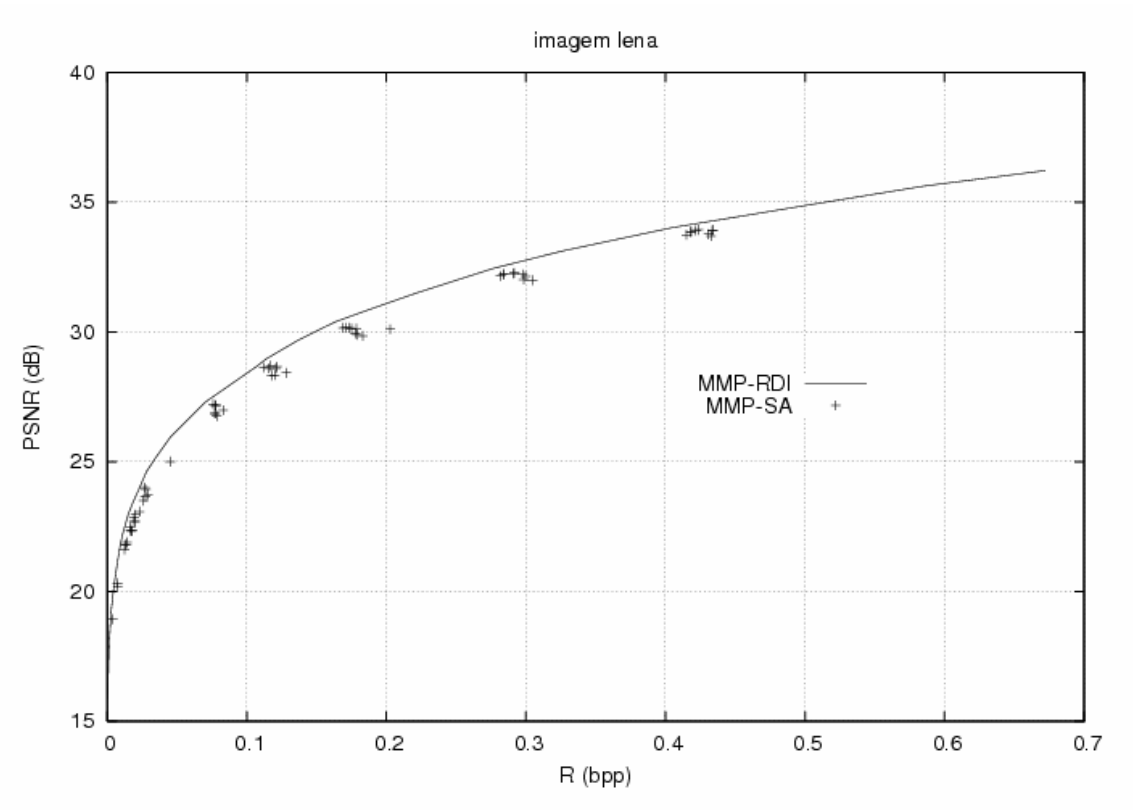


Figura 6.15: Resultados dos testes com o MMP-SA com dois lambdas.

Mais uma vez, vemos que, em nenhum caso, o MMP-SA obteve um resultado melhor que o MMP-RDI. Lembramos, também, que os melhores resultados foram obtidos nos testes com os maiores primeiros valores de lambda, como já discutido, ou seja, quando a seqüência de dois lambdas vai se degenerando para uma "seqüência" de apenas um valor de lambda, o que nada mais é do que o MMP-SA se tornando o próprio MMP-RDI.

Concluídos os testes com dois lambdas, passamos a executar novos testes com o MMP-SA, agora, utilizando seqüências com três valores de lambda. Foi utilizado o mesmo algoritmo dos testes com pares de lambdas, descritos anteriormente. Desta vez, testamos 120 diferentes seqüências a fim de percebermos alguma outra característica que possa incrementar o desempenho do MMP-SA. A Tabela 6.3 relaciona todas as trincas utilizadas.

Tabela 6.3: Trincas de lambdas usadas nos testes do MMP-SA.

| **Primeiro Valor** | **Segundos Valores** | **Terceiros Valores** |
|---|---|---|
| 250 | 100 | 50 |
| 500 | 100 | 50 |
| | 250 | 100 e 50 |
| 1000 | 100 | 50 |
| | 250 | 100 e 50 |
| | 500 | 250, 100 e 50 |
| 5000 | 100 | 50 |
| | 250 | 100 e 50 |
| | 500 | 250, 100 e 50 |
| | 1000 | 500, 250, 100 e 50 |
| 10000 | 100 | 50 |
| | 250 | 100 e 50 |
| | 500 | 250, 100 e 50 |
| | 1000 | 500, 250, 100 e 50 |
| | 5000 | 1000, 500, 250, 100 e 50 |
| 20000 | 100 | 50 |
| | 250 | 100 e 50 |
| | 500 | 250, 100 e 50 |
| | 1000 | 500, 250, 100 e 50 |
| | 5000 | 1000, 500, 250, 100 e 50 |
| | 10000 | 5000, 1000, 500, 250, 100 e 50 |
| 40000 | 100 | 50 |
| | 250 | 100 e 50 |
| | 500 | 250, 100 e 50 |
| | 1000 | 500, 250, 100 e 50 |
| | 5000 | 1000, 500, 250, 100 e 50 |
| | 10000 | 5000, 1000, 500, 250, 100 e 50 |
| | 20000 | 10000, 5000, 1000, 500, 250, 100 e 50 |
| 80000 | 100 | 50 |
| | 250 | 100 e 50 |
| | 500 | 250, 100 e 50 |
| | 1000 | 500, 250, 100 e 50 |
| | 5000 | 1000, 500, 250, 100 e 50 |
| | 10000 | 5000, 1000, 500, 250, 100 e 50 |
| | 20000 | 10000, 5000, 1000, 500, 250, 100 e 50 |
| | 40000 | 20000, 10000, 5000, 1000, 500, 250, 100 e 50 |

Mostramos nas Figuras 6.16 a 6.24 alguns dos gráficos gerados a partir dos resultados obtidos nos testes. Analogamente ao feito no caso anterior, cada gráfico ilustra os resultados para trincas com os mesmos primeiro e segundo valores. Como os dois pontos *(R,D)* relativos às duas primeiras etapas do algoritmo são os mesmos para todas as trincas de cada gráfico, eles são mostrados uma única vez em cada figura.

Assim, para cada seqüência de valores, somente a coordenada relativa ao resultado do terceiro passo do algoritmo é mostrada.

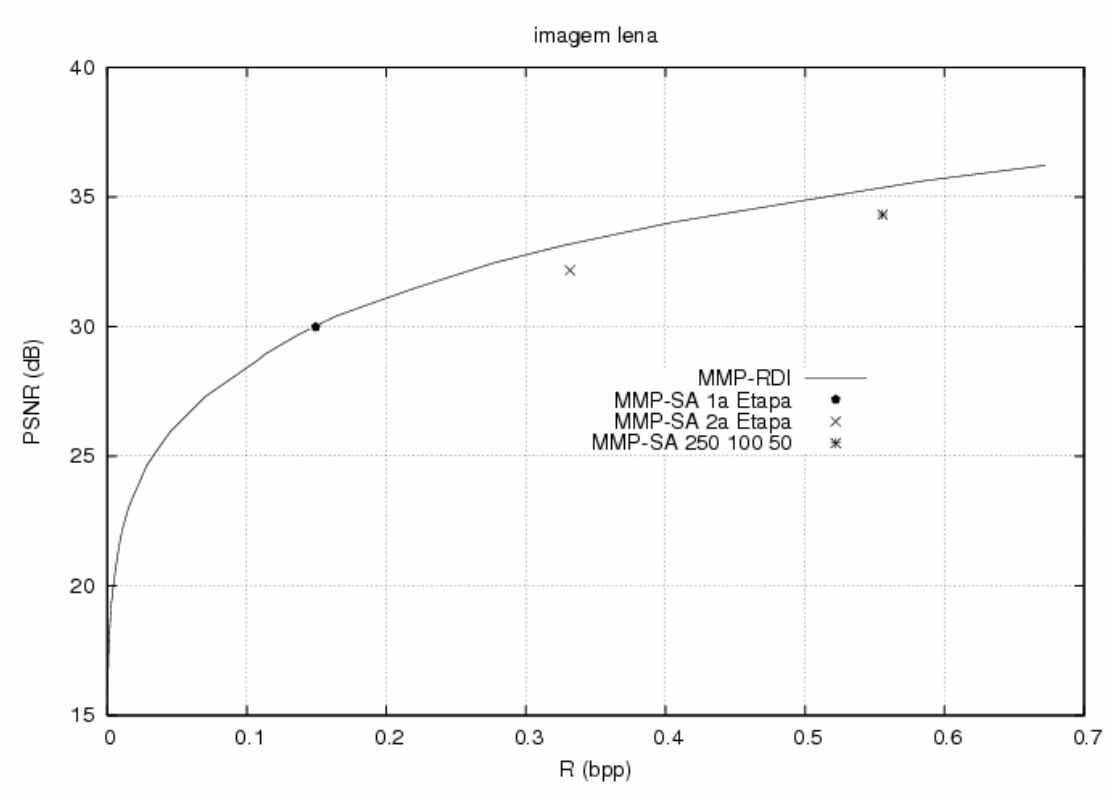


Figura 6.16: Resultados do MMP-SA para os pares *{250,100, x}*.

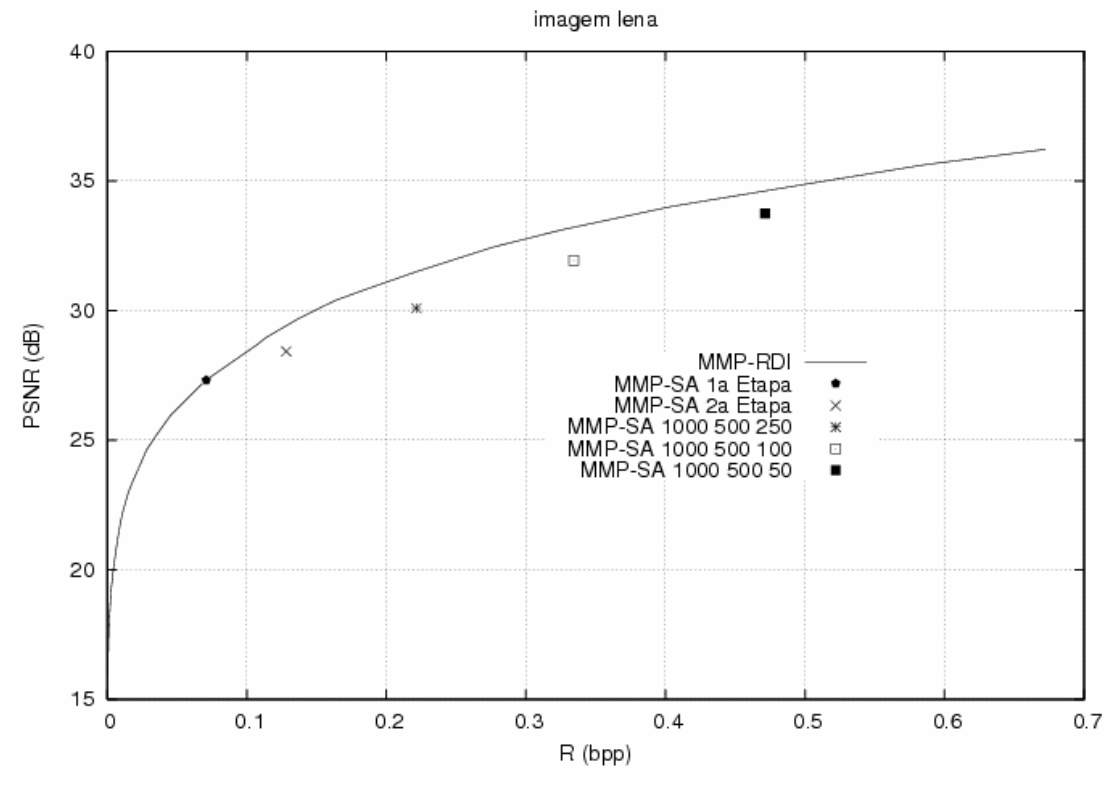


Figura 6.17: Resultados do MMP-SA para os pares *{1000,500, x}*.

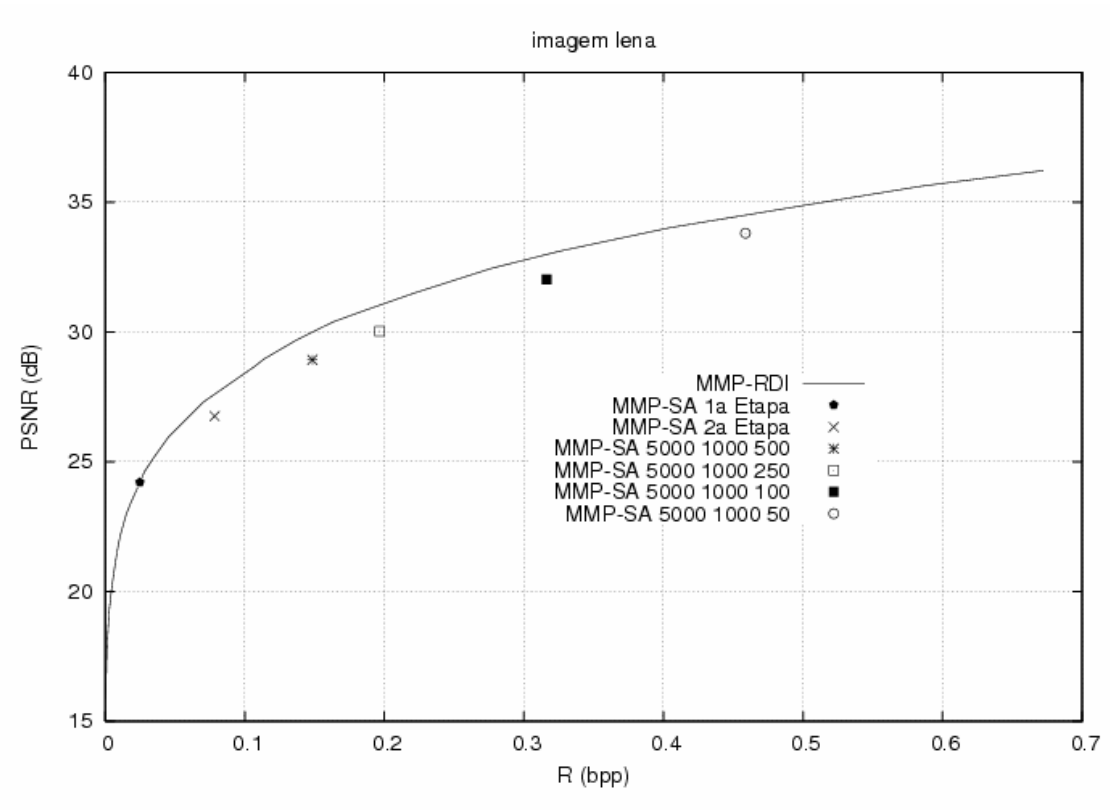


Figura 6.18: Resultados do MMP-SA para os pares *{5000,1000, x}*.

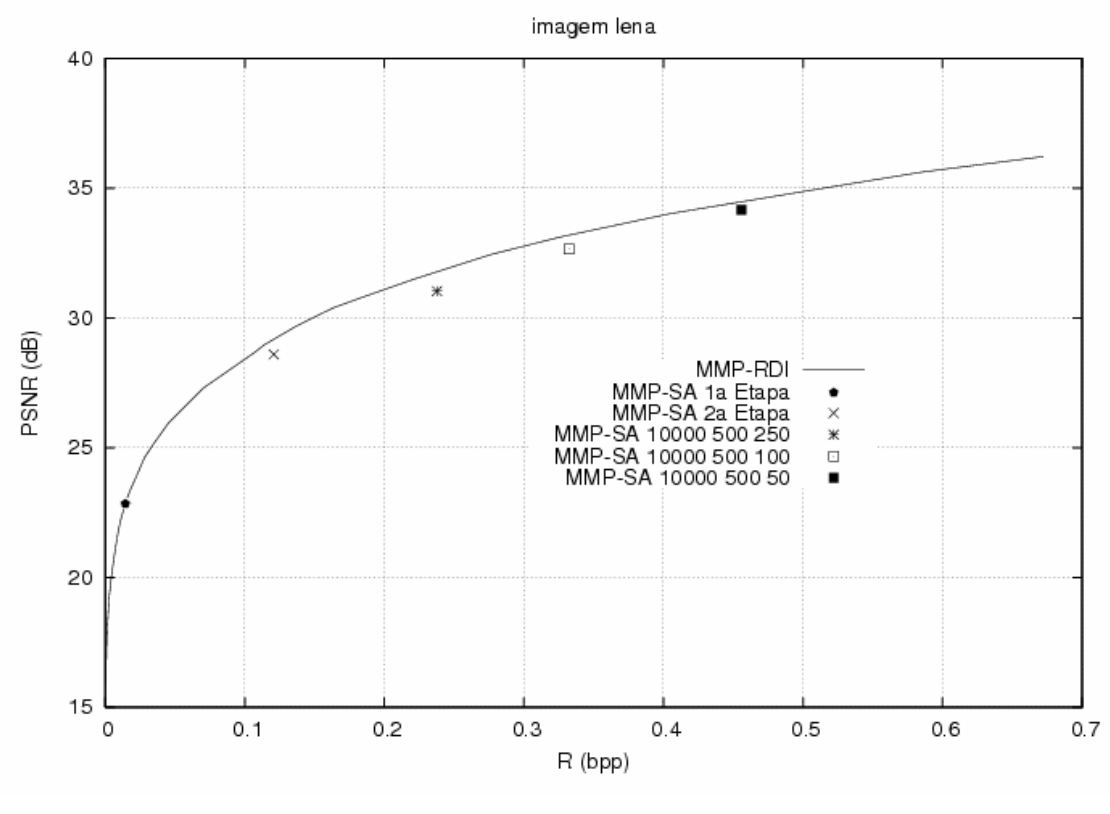


Figura 6.19: Resultados do MMP-SA para os pares *{10000,500, x}*.

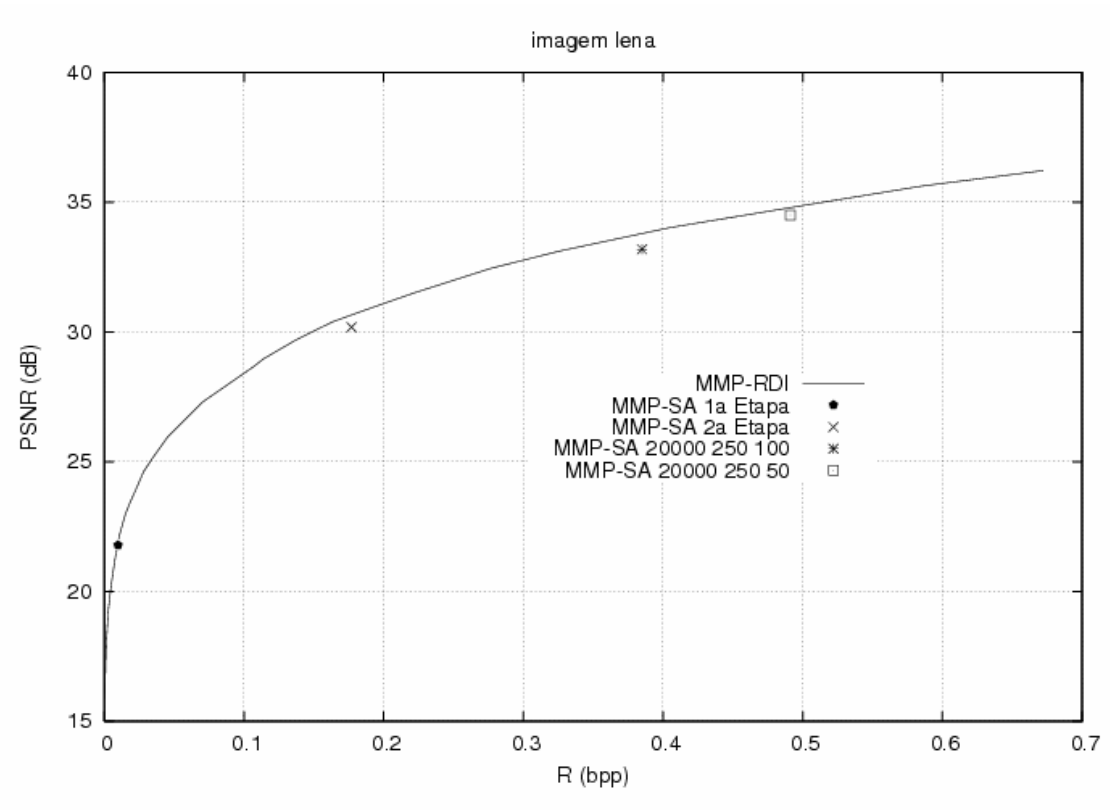


Figura 6.20: Resultados do MMP-SA para os pares *{20000,250, x}*.

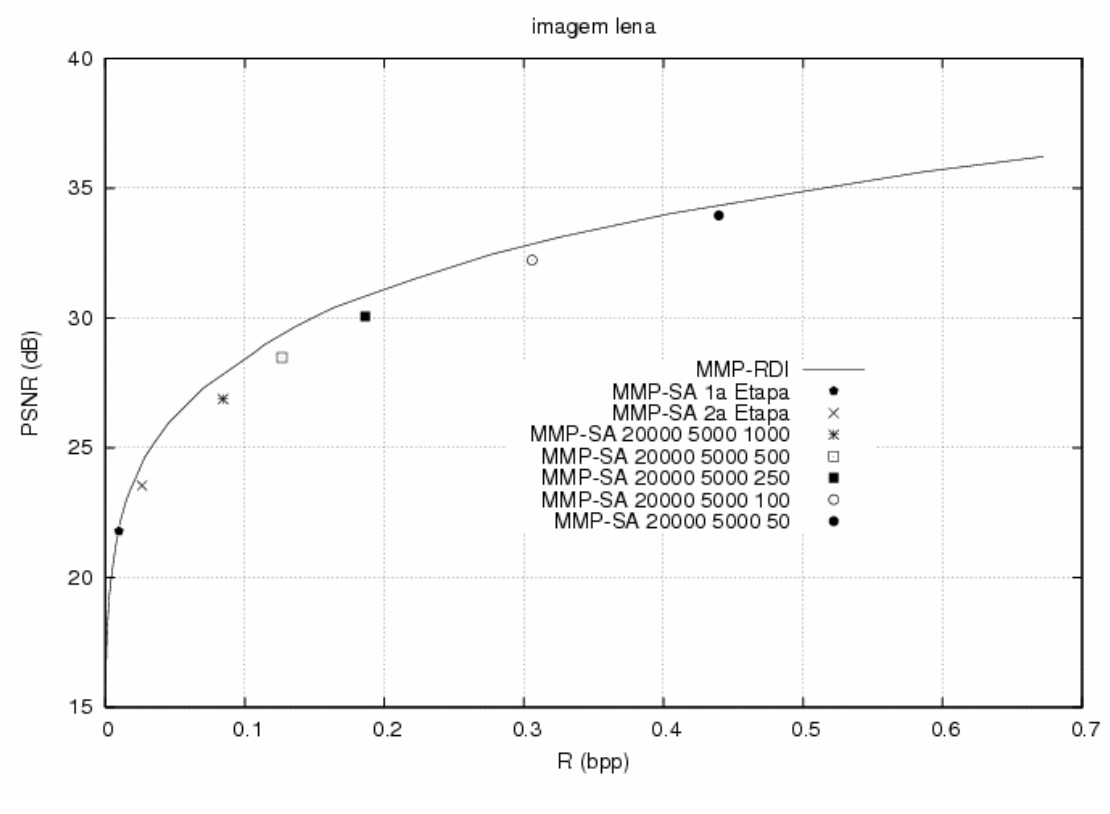


Figura 6.21: Resultados do MMP-SA para os pares *{20000,5000, x}*.

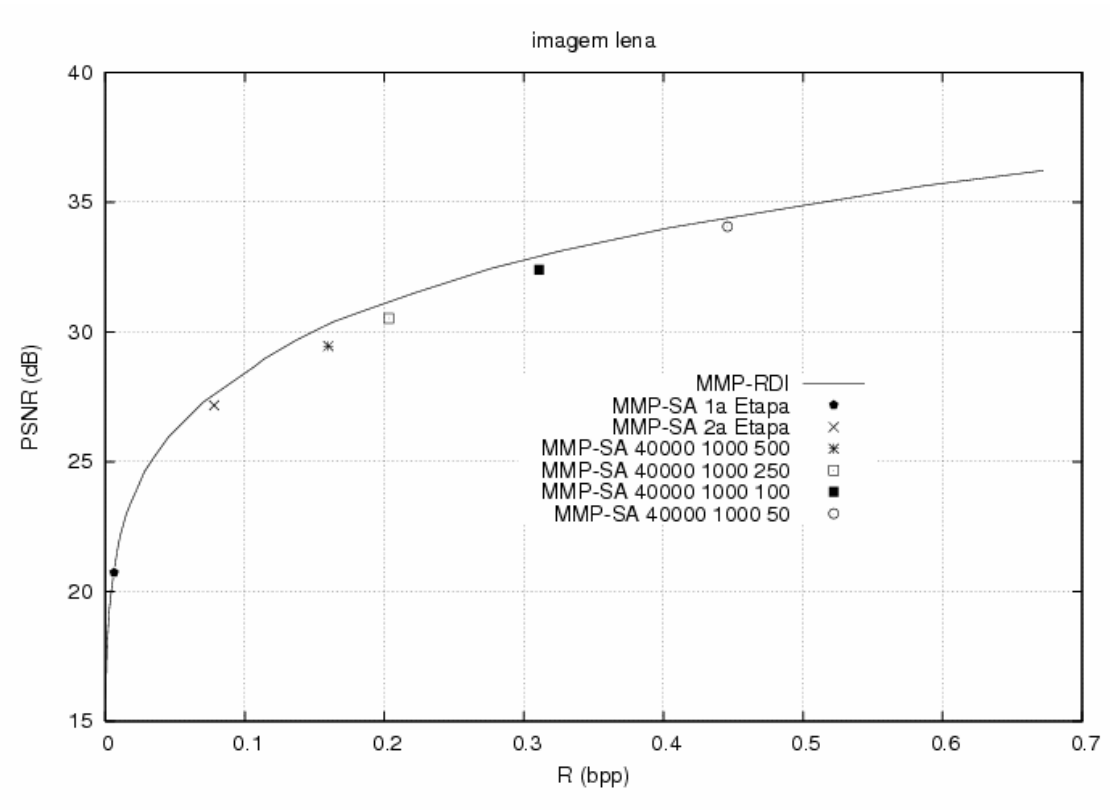


Figura 6.22: Resultados do MMP-SA para os pares *{40000,1000, x}*.

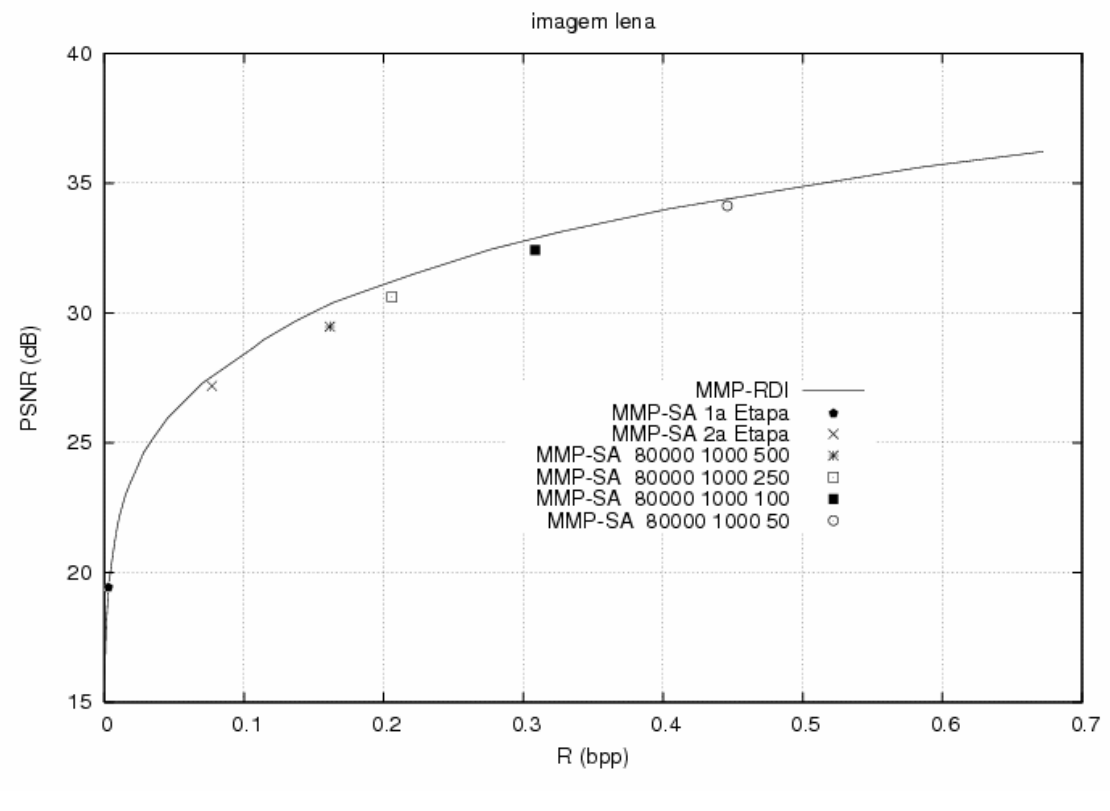


Figura 6.23: Resultados do MMP-SA para os pares *{80000,1000, x}*.

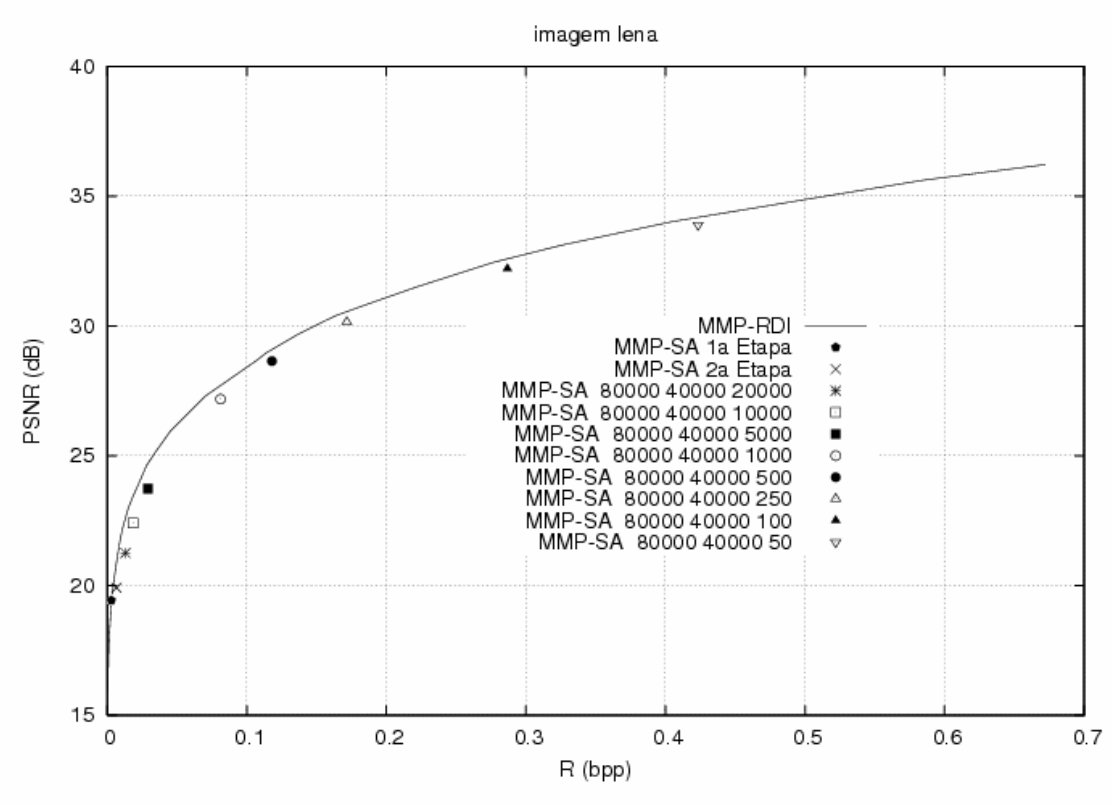


Figura 6.24: Resultados do MMP-SA para os pares *{80000,40000, x}*.

Novamente, podemos observar que o ponto relativo à primeira etapa do algoritmo ficou, em todos os casos, sobre a curva do MMP-RDI. A outra característica esperada também pôde ser percebida nos testes com seqüências de três lambdas, como mostra a Tabela 6.4, que apresenta as menores perdas em relação ao MMP-RDI conseguidas, para cada lambda inicial, na terceira etapa do algoritmo e a que taxa elas foram obtidas. Fica claro que os pontos foram se aproximando da curva do MMP-RDI à medida que o primeiro valor de lambda foi crescendo.

Tabela 6.4: Menores perdas do MMP-SA em relação ao MMP-RDI.

| **Primeiro Lambda** | **Perda Mínima (dB)** | **Taxa (bpp)** |
|---|---|---|
| 500 | 0,73 | 0,55 |
| 1000 | 0,62 | 0,57 |
| 5000 | 0,43 | 0,49 |
| 10000 | 0,32 | 0,46 |
| 20000 | 0,29 | 0,49 |
| 40000 | 0,27 | 0,46 |
| 80000 | 0,25 | 0,48 |

Outro ponto observado foi que, para todos os diferentes lambdas iniciais testados, o resultado mais próximo ao MMP-RDI foi obtido com o terceiro valor de lambda sendo o menor possível, ou seja, para a maior taxa atingida, o que nos indica que, possivelmente, os resultados podem se aproximar ainda mais do MMP-RDI para taxas mais elevadas do que as testadas neste trabalho.

A Tabela 6.5 mostra com que segundos valores de lambda obtivemos os melhores resultados na terceira etapa do algoritmo para cada teste com os mesmos lambdas inicial e final.

Tabela 6.5: Melhor segundo lambda para trincas com os mesmos lambdas inicial e final.

| **Primeiro Lambda** | **Terceiro Lambda** | **Perda Mínima (dB)** | **Segundo Lambda** |
|---|---|---|---|
| 500 | 50 | 0,72 | 100 |
| 1000 | 50 | 0,62 | 100 |
| 5000 | 50 | 0,43 | 250 |
| 10000 | 50 | 0,32 | 500 |
| 20000 | 50 | 0,29 | 250 |
| 40000 | 50 | 0,27 | 500 |
| 80000 | 50 | 0,25 | 250 |

Percebemos que há uma certa variação no melhor segundo lambda, porém, eles ficaram sempre com valores bem menores que o do lambda inicial. Isto nos indica que, para os testes realizados, obtivemos um melhor desempenho ao utilizarmos um maior espaçamento entre os primeiro e o segundo valores do que entre o segundo e o terceiro. Exemplificando, vimos que a seqüência *{20000, 500, 50}* foi melhor para o desempenho do MMP-SA do que a seqüência *{20000, 10000, 50}*.

Finalmente, como para as seqüências de apenas dois valores de lambda, notamos, que, em nenhum dos casos testados, obtivemos um desempenho melhor que o do MMP-RDI, como destaca a Figura 6.25, que mostra todos os resultados da terceira etapa do MMP-SA em um só gráfico.

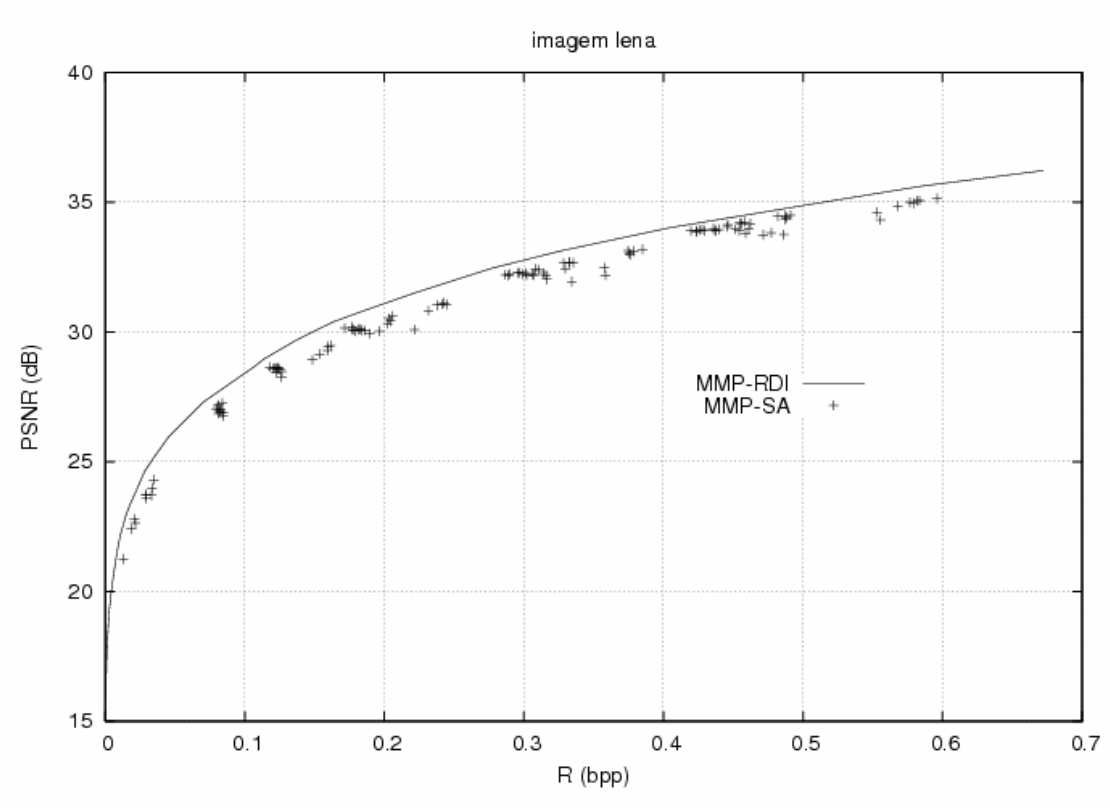


Figura 6.25: Resultados dos testes com o MMP-SA com três lambdas.

# Capítulo 7

# Conclusão

A primeira parte deste trabalho consistiu em apresentarmos os princípios básicos dos algoritmos e conceitos utilizados. Introduzimos o MMP original [5], bem como suas versões com otimização taxa-distorção [5, 6], e o UMMP [5].

A seguir, a partir do Capítulo 4, iniciamos a implementação das três idéias a serem introduzidas e desenvolvidas neste trabalho. A primeira delas foi a tentativa de melhoria do UMMP por meio da incorporação de otimização taxa-distorção ao algoritmo original. Propusemos três versões do UMMP-RD, sendo aquela que considerou a idéia de segmentação do dicionário em espaço de códigos (UMMP-RD2) a que obteve resultados um pouco superiores.

O ganho do UMMP-RD2 em relação ao UMMP original ficou, na maioria dos casos, entre 0,5dB e 1dB para as imagens suaves, e entre 1dB e 1,6dB para a imagem de texto (*pp1205*). Entretanto, apesar destes ganhos, o UMMP-RD ainda ficou abaixo do MMP-RDI, com um déficit que variou, na maioria dos casos, entre 0,5dB e 1dB para as imagens suaves, e entre 1,5dB (taxas baixas) e 6dB (a 1bpp) para a imagem *pp1205*.

Por estes resultados, percebemos que dificilmente o UMMP poderá superar o MMP, já que a diferença de desempenho entre os dois se mostrou considerável, especialmente para a imagem de texto, caso em que não alcançou nem mesmo o SPIHT e cuja codificação é o ponto forte do MMP. Assim, pareceu-nos que a principal característica do MMP, que é a sua independência com relação a pressuposições acerca da imagem, não se faz presente com tanto destaque no UMMP. Com isso, pelo menos com o desenvolvimento feito até o presente momento, o UMMP não se mostra muito atraente para codificação de imagens.

O segundo objetivo deste trabalho foi alcançado no Capítulo 5, ao propormos e implementarmos com sucesso o algoritmo UMMP-H, que incorporou o conceito de Codificação por Transformada ao UMMP-RD. Como a proposta foi apenas introduzir a idéia, optamos por utilizar a Transformada de Haar ao invés de outras com características mais apropriadas para codificação de imagens.

O UMMP-H1, primeira versão do algoritmo, não logrou melhorar o desempenho do UMMP-RD, já que, como mostra a análise feita no Capítulo 5, poucos foram os blocos da imagem original que o Codificador de Haar codificou com menor custo que o UMMP-RD2. Com isto em mente, implementamos o UMMP-H2, que utiliza uma forma diferente de se codificar o nível DC do bloco. O novo algoritmo conseguiu utilizar mais blocos Haar, obtendo um certo ganho de desempenho em relação ao UMMP-RD, que chegou a 0,3dB a 1,1bpp em *bridge*. O pequeno ganho reflete o ainda reduzido número de blocos Haar utilizados, nos levando a crer que o uso de outras transformadas mais adequadas, como a DCT e DWTs mais sofisticadas, pode gerar uma maior melhoria de desempenho.

Outra conclusão interessante acerca do UMMP-H é que, com ele, aumentamos a capacidade de adaptação do algoritmo às características da imagem, já que temos dois tipos de codificadores a ele incorporados. Se a imagem original for suave, o codificador por transformada, por ser mais eficiente para este tipo de imagem, será responsável pela codificação de mais blocos do que se a imagem original possuir mais componentes de alta freqüência, caso em que codificadores que usam recorrência de padrões com escalas se mostram superiores. Também para imagens mistas podemos obter ganhos, já que cada parte da imagem pode ser codificada por um tipo diferente de codificador.

A terceira fase do trabalho, consistente na incorporação do conceito de Aproximações Sucessivas ao MMP-RDI, foi realizada no Capítulo 6. Conseguimos implementar quatro versões do algoritmo MMP-SA, que a cada passo executado gera uma nova aproximação para a imagem original. A principal característica do algoritmo é a sua capacidade de gerar um *bitstream* que abriga diferentes aproximações da imagem, cada qual com uma característica de taxa e distorção. Nos primeiros testes executados, o desempenho do algoritmo foi inferior ao do MMP-RDI, mostrando-se, entretanto, bastante suscetível à variação da seqüência de valores de lambda utilizada.

Por isso, ainda no Capítulo 6, fizemos um estudo sobre a influência da seqüência de valores de lambda no desempenho do MMP-SA. Fizemos diversos testes com uma vasta gama de pares e trincas de valores, concluindo que, aparentemente, quanto maior

for diferença entre os valores, mais o algoritmo se aproxima do MMP-RDI e que, para as seqüências de três valores, é mais vantajoso usar um espaçamento entre o primeiro e segundo valores maior que o usado entre o segundo e terceiro.

Assim, apesar do desempenho do MMP-SA ter sido inferior ao do MMP-RDI em todos os testes executados, a sua característica de codificação por aproximações sucessivas pode fazer com que ele seja de grande utilidade em certas aplicações, como a descrita na Seção 6.2.

Finalmente, encerramos afirmando que as três propostas contidas neste trabalho obtiveram êxito, já que nos fizeram chegar a diversas conclusões sobre algumas características dos algoritmos que utilizam casamento de padrões recorrentes com escalas, especialmente que o UMMP não se mostrou, até o momento, muito competitivo com relação ao MMP e que novas vertentes de utilização do conceito podem ser exploradas.

Ficam em aberto algumas questões, que podem ser pesquisadas futuramente, como:

- a tentativa de incorporação ao UMMP-RD de alguns outros conceitos já incorporados ao MMP otimizado, como os introduzidos em [6, 9, 12];
- a utilização da idéia introduzida no UMMP-H com outros codificadores mais eficientes, como o MMP-RDI (ou seus derivados) e codificadores por transformada mais apropriados, como os que utilizam a DCT ou DWTs mais elaboradas;
- a busca de dicionários iniciais para o MMP-SA que possam favorecer o uso do *flag* introduzido no MMP-SA3; e

# Apêndice A

# Algoritmo MMP

Algumas definições devem ser feitas para a implementação do algoritmo. Devemos elaborar um conjunto de $N_D = K + L + 1$ dicionários iniciais, sendo $2^K$ x $2^L$ as dimensões do bloco inicial e definir uma distorção-alvo $d_*$ e uma transformação de escala.

No caso em estudo, a transformação de escala utilizada é a mesma definida pelas equações 2.5 e 2.6. Como os blocos são 8x8, temos $N_D = 7$. Em [5], a estratégia de inicialização do dicionário leva em conta a distorção-alvo $d_*$, sendo o dicionário inicial de dimensão 1x1 dado por:

$$D_0^{(1,1)} = \{\min(X_{m,n}), \min(X_{m,n}) + \delta, ..., 0, \delta, ..., \max(X_{m,n})\}$$

$$\delta = \frac{2(\max(X_{m,n}) - \min(X_{m,n}))}{sqrt(d_*)} \tag{A.1}$$

onde $\min(X_{m,n})$ e $\max(X_{m,n})$ são, respectivamente, os valores mínimo e máximo dos *pixels* da imagem (normalmente 0 e 255). Os dicionários iniciais nas outras dimensões são obtidos pela aplicação da transformação de escala em $D_0^{(1,1)}$.

Por motivos de limitação de memória, o tamanho dos dicionários deve ser limitado com o auxílio de uma estratégia de atualização. Em [5], o limite de cada dicionário foi de 32.768 elementos, e, sempre que um novo elemento devesse ser adicionado a um dicionário já cheio, o vetor menos usado era descartado.

Por último, vale lembrar que a seqüência de índices *i* e os *flags* de segmentação *0* e *1* é, ainda, codificada por um codificador aritmético, melhorando o desempenho do algoritmo. A seguir encontra-se o algoritmo de compressão MMP:

**Procedimento** $\hat{X}_j^{m,n} = \text{codifica}\left(X_j^{m,n}, \eta_0, \delta\right)$

**Passo 1:** Procura, no dicionário $D^{m,n}$, onde $m = 2^{\left\lfloor \frac{\eta_0+1}{2} \right\rfloor}$ e $n = 2^{\left\lfloor \frac{\eta_0}{2} \right\rfloor}$, 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 $\eta_0 == 0$, ou seja, 1x1 ($m == 1$ 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 $\delta$, ou seja, $\left\| X_j^{m,n} - S_i^{m,n} \right\|^2 \leq \delta^2.m.n$, 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 $k = m/2$ 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 $l = n/2$.

**Passo 6:** Computa $\hat{X}_{2j+1}^{k,l} = \text{codifica}\left(X_{2j+1}^{k,l}, \eta_0 - 1, \delta\right)$.

**Passo 7:** Computa $\hat{X}_{2j+2}^{k,l} = \text{codifica}\left(X_{2j+2}^{k,l}, \eta_0 - 1, \delta\right)$.

**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 $n = 0,1,\ldots,N_D - 1$,

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

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

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

# Apêndice B

# Algoritmo MMP-RDI

Novamente, utilizou-se a transformação de escala descrita pelas equações 2.5 e 2.6. O dicionário inicial, devido à otimização, não fica mais condicionado à distorção-alvo, sendo o de dimensão 1x1 composto por 64 níveis igualmente espaçados, ou seja:

$$D_0^{(1,1)} = \{0, 4, 8, ..., 248, 252\} \tag{B.1}$$

Os dicionário iniciais nas outras dimensões é obtido pela aplicação da transformação de escala em $D_0^{(1,1)}$.

As rotinas de otimização e de codificação são descritas a seguir.

**Procedimento** $\{\hat{X}_l^{m,n}, T\} = \text{otimiza}(X_j^{m,n}, T_0, \eta_0)$

**Passo 1:** Faz $T = T_0$.

**Passo 2:** Procura, no dicionário $D^{m,n}$, onde $m = 2^{\left\lfloor \frac{\eta_0+1}{2} \right\rfloor}$ e $n = 2^{\left\lfloor \frac{\eta_0}{2} \right\rfloor}$, o elemento $S_i^{m,n}$ que representa $X_l^{m,n}$ com menor custo $J_{n_l}$, armazenando $i$, $J_{n_l}$ e o dicionário de origem. O elemento $S_i^{m,n}$ é armazenado em $\hat{X}_l^{m,n}$. O custo $J_{n_l}$ 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 $J_{n_l}$ menor que o escolhido no **Passo 2**. Se isso

ocorrer, substitui $S_i^{m,n}$ por $S_{i_R}^{m,n}$, armazenando $i_R$, $J_{n_l}$ e o dicionário de origem. O custo $J_{n_l}$ 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 $\eta_0 == 0$, ou seja, 1x1 ($m == 1$ e $n == 1$), incrementa $\overline{C}_i^{m,n}$ ou $C_{i_R}^{m,n}$, dependendo da origem do elemento escolhido, e retorna $\hat{X}_j^{m,n}$ e *T*.

Senão, vai para o **Passo 5.**

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

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

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

**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 $k = m/2$ 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 $j = n/2$.

**Passo 9:** Computa $\left\{\hat{X}_{2l+1}^{k,j}, T_1\right\} = \text{otimiza}\left(X_{2l+1}^{k,j}, T, \eta_0 - 1\right)$.

**Passo 10:** Computa $\left\{\hat{X}_{2l+2}^{k,j}, T_2\right\} = \text{otimiza}\left(X_{2l+2}^{k,j}, T, \eta_0 - 1\right)$.

**Passo 11:** Atualiza a árvore de segmentação *T* de acordo com $T_1$ e $T_2$.

**Passo 12:** Se $J_l \le J_{2l+1} + J_{2l+2} + \lambda.R_{0l}$, vai para o **Passo 13**.

Senão, vai para o **Passo 19**.

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

**Passo 14:** Decrementa os contadores $\overline{C}_{1F}^{w,y}$ e $\overline{C}_h^{w,y}$ ou $C_{h_R}^{w,y}$ nas escalas *w* x *y* referentes aos nós-folha 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 $\overline{C}_{0F}^{m,n}$.

**Passo 16:** Incrementa o contador $\overline{C}_{1F}^{m,n}$ e o contador $\overline{C}_{i}^{m,n}$ ou $\overline{C}_{i_R}^{m,n}$, 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 *T*, que as sub-árvores $S(n_{2l+1})$ e $S(n_{2l+2})$ foram podadas e retorna $\hat{X}_j^{m,n}$ e *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 $\hat{X}_l^{m,n} = \left(\hat{X}_{2l+1}^{k,j} \hat{X}_{2l+2}^{k,j}\right)$.

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

Para $n = 0,1,\ldots,N_D - 1$,

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

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

**Passo 21:** Faz $J_l = J_{2l+1} + J_{2l+2} + \lambda.R_{0l}$.

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

**Procedimento** $\hat{X}_l^{m,n} = \text{codifica}\left(X_l^{m,n}, \eta_0, T\right)$

**Passo 1:** Se $T_{2l+1} == 0$ e $T_{2l+2} == 0$, ou $\eta_0 == 0$, vai para o **Passo 2**.

Senão, vai para o **Passo 5.**

**Passo 2:** Procura, no dicionário $D^{m,n}$, onde $m = 2^{\left\lfloor \frac{\eta_0+1}{2} \right\rfloor}$ e $n = 2^{\left\lfloor \frac{\eta_0}{2} \right\rfloor}$, o elemento $S_i^{m,n}$ que representa $X_l^{m,n}$ com menor distorção custo $J_{n_l}$, armazenando-o em $\hat{X}_l^{m,n}$.

**Passo 3:** Se a escala atual for $\eta_0 == 0$, ou seja, 1x1 ($m == 1$ 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 $k = m/2$ 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 $j = n/2$.

**Passo 7:** Computa $\hat{X}_{2l+1}^{k,l} = \text{codifica}\left(X_{2l+1}^{k,j}, \eta_0 - 1, T\right)$.

**Passo 8:** Computa $\hat{X}_{2l+2}^{k,l} = \text{codifica}\left(X_{2l+2}^{k,j}, \eta_0 - 1, T\right)$.

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

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

Para $n = 0,1,\ldots,N_D - 1$,

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

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

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

# Apêndice C

# Algoritmo UMMP

O algoritmo UMMP codifica um vetor de entrada $\mathrm{X} = (x_0, x_1, ..., x_{N-1})^T$ usando um dicionário *D*, inicialmente composto por *M* elementos, $D = \{S_0, S_1, ..., S_{M-1}\}$, como o *frame* no qual será decomposto X, e uma distorção-alvo $d^*$. Além disso define-se uma função para a transformação de escala $T_N^M : \Re^M \rightarrow \Re^N$ que mapeia um vetor de tamanho *M* em outro de tamanho *N*.

**Procedimento** $\hat{R}^{i,j} = \text{codifica}(R^{i,j}, D, d^*)$

**Passo 1:** Transforma os elementos do dicionário *D* para que tenham o mesmo tamanho *n* de $R^{i,j}$, isto é: $S_k^S = T_n[S_k]$, para todo *k*.

**Passo 2:** Procura o elemento $S_{k_{i,j}}^S$ que minimiza a distorção $\left\| R^{i,j} - S_{k_{i,j}}^S \right\|$ e faz $\hat{R}^{i,j} = S_{k_{i,j}}^S$.

**Passo 3:** Codifica o índice $k_{i,j}$.

**Passo 4:** Se $\left\| R^{i,j} - \hat{R}^{i,j} \right\|^2 \leq n.d^*$, codifica o *flag* '1' e retorna $\hat{R}^{i,j}$.

Senão, vai para o **Passo 5.**

**Passo 5:** Calcula o resíduo $R^{i+1,j} = R^{i,j} - \hat{R}^{i,j}$ e codifica o *flag* '0'.

**Passo 6:** Divide $R^{i+1,j} = \left(R^{i+1,2j+1} \quad R^{i+1,2j+2}\right)$ onde $R^{i+1,2j+1}$ e $R^{i+1,2j+2}$ têm tamanho $n/2$.

**Passo 7:** Computa $\hat{R}^{i+1,2j+1} = \text{codifica}(R^{i+1,2j+1}, D, d^*)$.

**Passo 8:** Computa $\hat{R}^{i+1,2j+2} = \text{codifica}(R^{i+1,2j+2}, D, d^*)$.

**Passo 9:** Faz $\hat{R}^{i+1,j} = \left(\hat{R}^{i+1,2j+1} \quad \hat{R}^{i+1,2j+2}\right)$.

**Passo 10:** Atualiza $\hat{R}^{i,j} = S^{S}_{k_{i,j}} + \hat{R}^{i+1,j}$.

**Passo 11:** Atualiza o dicionário *D*, ou seja, faz $D = D \cup \hat{R}^{i,j} \cup \hat{R}^{i+1,j}$.

**Passo 12:** Retorna $\hat{R}^{i,j}$.

# Apêndice D

# Algoritmo do Codificador de Haar

O Codificador de Haar implementado neste trabalho utiliza a Transformada de Haar [13] e o algoritmo MGE [15], descritos no Capítulo 5, para codificar blocos de dimensão 8x8. Assim, as imagens a serem codificadas são divididas em blocos 8x8, que são processamento seqüencialmente de cima para baixo e da esquerda para a direita.

Para cada bloco, o algoritmo primeiramente aplica a Transformada de Haar 8x8, de acordo com as Equações 5.6 e 5.7. A seguir, ele faz a codificação dos coeficientes da transformada utilizando o método MGE, cujos símbolos de saída são codificados por um codificador aritmético com modelos distintos para cada tipo de símbolo gerado.

O método MGE divide o domínio da transformada em uma *quadtree* de modo a conseguir localizar os coeficientes significativos eficientemente. No caso, o domínio possui dimensão 8x8 e a menor região utilizada nesta implementação foi de tamanho 2x2. Assim, temos 21 regiões possíveis, sendo dezesseis de dimensão 2x2, quatro de dimensão 4x4 e uma 8x8, como mostrado na Figura D.1, que também ilustra a árvore quaternária correspondente.

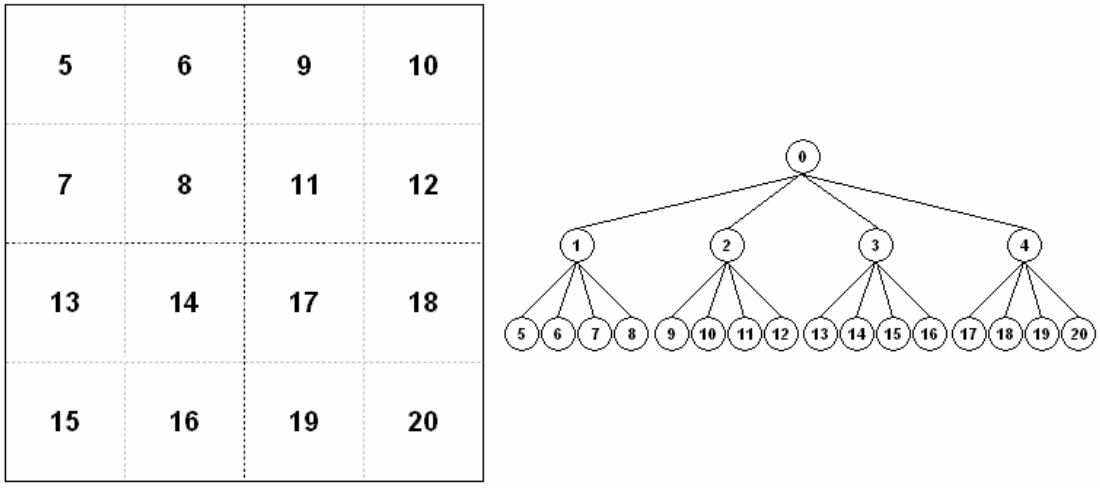


Figura D.1: Divisão em *quadtree* utilizada na implementação do Codificador de Haar.

O algoritmo do Codificador de Haar implementado é descrito a seguir:

**Procedimento** $\hat{R} = \text{codifica_haar}(R)$

**Passo 1:** Aplica a Transformada de Haar ao bloco de entrada $R$, de dimensão 8x8, isto é, faz $R_H = \mathrm{H}_8.R.\mathrm{H}_8^{\mathrm{T}}$.

**Passo 2:** Faz $n = 10$.

**Passo 3:** Marca o domínio da transformada como "região zero".

**Passo 4:** Escolhe a região a codificar como sendo todo o domínio da transformada e, se ela for uma "região zero", vai para o **Passo 5**.
Senão, vai para o **Passo 6.**

**Passo 5:** Se a região possuir algum coeficiente significante, codifica "1" e marque a região como sendo "região um". Senão, codifica "0" e vai para o **Passo 12**.

**Passo 6:** Se a região ainda não foi dividida anteriormente, divide-a em 4 regiões de dimensão 4x4 e as marca com "região zero".

**Passo 7:** Para cada uma das regiões 4x4, verifica se ela é uma "região zero". Em caso positivo, vai para o **Passo 8**. Caso contrário, vai para o **Passo 9.**

**Passo 8:** Se a região possuir algum coeficiente significante, codifica "1" e a marca como "região um". Caso contrário, codifica "0". Vai para o **Passo 10**.

**Passo 9:** Se a região ainda não foi dividida anteriormente, divide-a em 4 regiões de dimensão 2x2 e, para cada uma delas, verifica se há elemento significante. Em

caso positivo, codifica “1” e marca a região como “região um”, caso contrário, codifica “0” e marca a região como “região zero”.

**Passo 10:** Para cada um dos coeficientes das regiões 2x2 geradas no **Passo 9**, faz: adiciona-o à Lista de Coeficientes Significantes (SCL), codifica o n-ésimo bit mais significativo do seu valor absoluto e o seu sinal, e verifica a condição de término da codificação.

**Passo 11:** Para cada coeficiente presente na SCL, e que não tenha sido incluído nela no último passo dominante, faz: codifica o n-ésimo bit mais significativo do seu valor absoluto e verifica a condição de término da codificação.

**Passo 12:** Decrementa *n* de uma unidade e, se $n > 0$, vai para o **Passo 4**.
Se não, retorna.

Alguns aspectos importantes do algoritmo implementado devem ser ressaltados. O primeiro plano de *bits* analisado possui $n = 10$, já que, como as imagens testadas têm como valor máximo de seus elementos $p_{MAX} = 255$, o maior coeficiente transformado possível será $C_{MAX} = 2040$. Assim, somente poderá haver coeficientes significantes com $n \leq 10$, pois $2^{10} < 2040 < 2^{11}$.

O algoritmo cria uma Lista de Coeficientes Significantes (SCL), que serve para armazenar a informação de quais coeficientes se tornaram significantes em algum plano de *bits*, seja o atual, seja algum plano anterior. Essa lista deve ser inicializada vazia;

O codificador aritmético utilizado para codificar os símbolos gerados pelo MGE possui um modelo distinto para cada um dos quatro tipos de símbolos existentes, a saber:

- o que indica se uma região possui ou não algum coeficiente significativo (Passos 5, 8 e 9);
- o n-ésimo *bit* mais significativo do valor absoluto de coeficiente adicionado à SCL no plano de *bits* atual (Passo 10);
- o que indica o sinal de coeficiente adicionado à SCL no plano de *bits* atual (Passo 10); e
- o n-ésimo *bit* mais significativo do valor absoluto de coeficiente adicionado à SCL em um plano de *bits* anterior (Passo 11).

O algoritmo utilizou, ainda, um critério de parada com o objetivo de indicar em que ponto a codificação do bloco de entrada deve ser terminada, considerando a variação do custo da codificação. Basicamente, a verificação é feita em determinados pontos do algoritmo (Passos 10 e 11), onde é calculado o custo da codificação até ali, que é comparado ao menor custo.

Para isso, na primeira verificação feita é armazenado o valor do custo até aquele ponto. A seguir, em cada verificação, se o custo no ponto for menor que o custo no ponto de menor custo até então, este passa a ser o ponto de menor custo, e a codificação continua. Caso contrário, a codificação só continua se atendidas duas condições: o custo tiver crescido menos de 50% em relação ao menor custo e tiverem sido feitas menos de 100 verificações desde o último ponto de menor custo. Finalmente, quando alguma verificação indicar que foi atingida a condição de parada, o algoritmo recupera seu estado no último ponto de menor custo registrado e se encerra.

# Apêndice E

# Imagens originais

Mostramos, neste apêndice, as imagens originais utilizadas nos testes ao longo deste trabalho. Foram usadas nove diferentes imagens: *aerial, baboon, barbara, bridge, f-16, gold, lena, pp1205 e pp1209*, todas de tamanho 512x512 *pixels*. As sete primeiras imagens, mostradas nas Figuras E.1 a E.7, foram obtidas no *site http://sipi.usc.edu/ services/database/Database.html*, e as duas últimas, mostradas nas Figuras E.8 e E.9, foram digitalizadas do *IEEE Transactions on Image Processing*, volume 9, número 7, de julho de 2000, páginas 1.205 e 1.209, respectivamente.

As imagens *aerial, baboon, barbara, bridge, f-16, gold* e *lena* são consideradas imagens suaves, por apresentarem componentes de alta freqüência pouco significativas. Já a imagem *pp1205* é uma imagem de texto, que tem a propriedade de conter elevadas componentes de alta freqüência, já que possui muitas bordas. Finalmente, a imagem *pp1209* é chamada de mista, por apresentar áreas de texto, como na *pp1205*, e áreas suaves, como nas outras sete imagens.

Figura E.1: Imagem *aerial* original, 512x512 *pixels* e 8bpp.

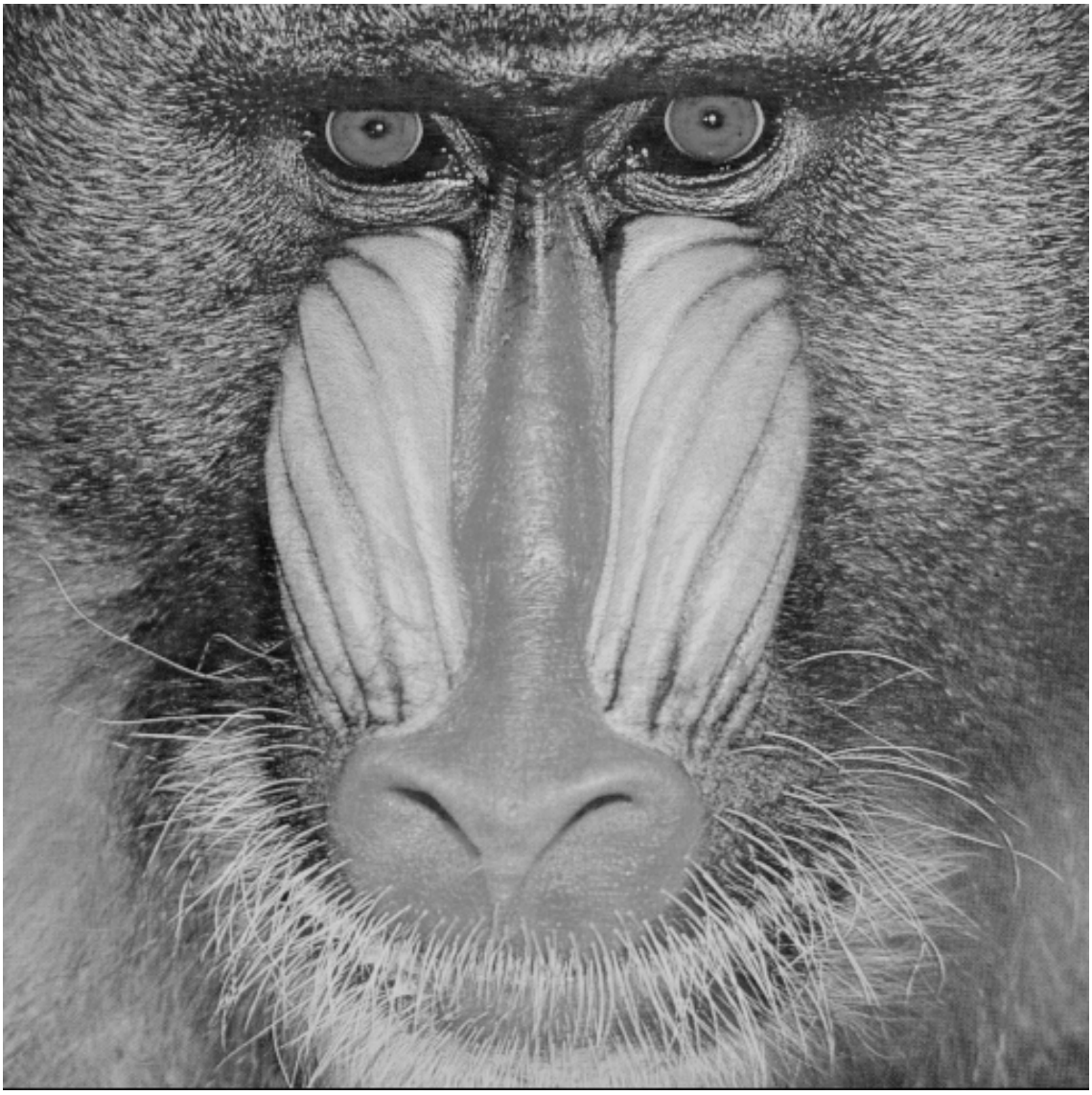

Figura E.2: Imagem *baboon* original, 512x512 *pixels* e 8bpp.

Figura E.3: Imagem *barbara* original, 512x512 *pixels* e 8bpp.

Figura E.4: Imagem *bridge* original, 512x512 *pixels* e 8bpp.

Figura E.5: Imagem *f-16* original, 512x512 *pixels* e 8bpp.

Figura E.6: Imagem *gold* original, 512x512 *pixels* e 8bpp.

Figura E.7: Imagem *lena* original, 512x512 *pixels* e 8bpp.

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

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

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

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

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

A (quantized) version of $\hat{\alpha}_c^{cod}$, $\hat{\alpha}_r^{cod}$ and $\hat{\beta}^{cod}$ 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

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

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

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

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 $\alpha_c$, $\alpha_r$, $\beta$ 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 $\alpha_c^0$, $\alpha_r^0$ and $\beta^0$.
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 $\alpha_c^k$, $\alpha_r^k$ and $\beta^k$ by substituting $\alpha_c^{k-1}$, $\alpha_r^{k-1}$ and $\beta^{k-1}$ 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 $\|\boldsymbol{f}_c^{(\alpha_c^k,\alpha_r^k,\beta^k)} - \boldsymbol{f}_c^{(\alpha_c^{k-1},\alpha_r^{k-1},\beta^{k-1})}\| + \|\boldsymbol{f}_r^{(\alpha_c^k,\alpha_r^k,\beta^k)} - \boldsymbol{f}_r^{(\alpha_c^{k-1},\alpha_r^{k-1},\beta^{k-1})}\|$ is less than a prescribed bound.
5) Using $\alpha_c^k$, $\alpha_r^k$, $\beta^k$ 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

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

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

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

where

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

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

Figura E.8: Imagem *pp1205* original, 512x512 *pixels* e 8bpp.

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 |

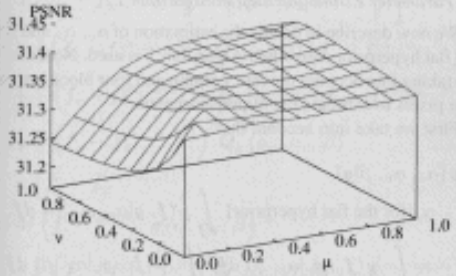


Fig. 6. PSNR for different values of $\mu$ and $\nu$ on the *Lena* image compressed at 0.29 bpp.

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 $\mu_c$ and $\mu_r$, defined in (37) and (38), were used for $\alpha_c$ and $\alpha_r$. The values used in the experiments were $\mu_c = \mu_r = \mu \in \{0.0, 0.1, \cdots, 1\}$. The normalized confidence parameter $\nu$, defined in (39), belongs to the same range. The 3-D plot in Fig. 6 shows the PSNR as a function of $\mu$ and $\nu$ 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 $\eta^{cod} = \beta^{cod} = 36.36^{-1}$ 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 $\mu$

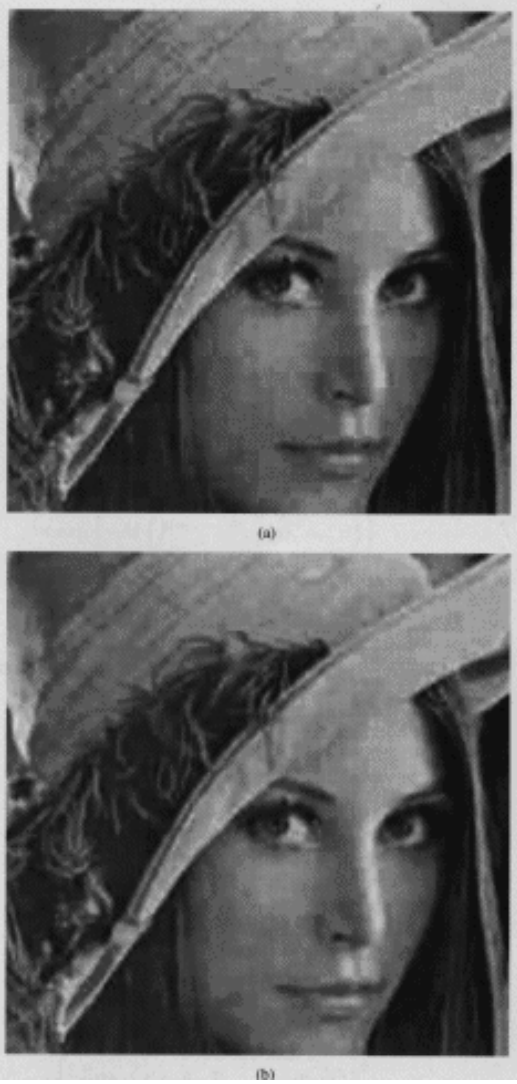


Figura E.9: Imagem *pp1209* original, 512x512 *pixels* e 8bpp.

# Referências Bibliográficas


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

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

[3] W.B. Pennebaker and J.L. Mitchel, "JPEG Still Image Data Compression Standard", Van Nostrand Reinhold, 1994.

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

[5] M. B. de Carvalho, "Compressão de Sinais Multidimensionais Usando Recorrência de Padrões Multiescalas", tese de doutorado, Programa de Engenharia Elétrica, COPPE/UFRJ, Rio de Janeiro, RJ, Março de 2001.

[6] E. B. Lima Filho, "Compressão de Imagens Utilizando Recorrência de Padrões Multiescalas com Critério de Continuidade Interblocos", tese de mestrado, Programa de Engenharia Elétrica, COPPE/UFRJ, Rio de Janeiro, RJ, Abril de 2004.

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

[8] H. Everett, “Generalized Lagrange Multiplier Method for Solving Problems of Optimum Allocation of Resources”, In Operation Research, v.11, pp. 399-417, 1963.

[9] W. S. S. Júnior, “Compressão de Imagens Utilizando Recorrência de Padrões Multiescalas com Segmentação Flexível”, tese de mestrado, Programa de Engenharia Elétrica, COPPE/UFRJ, Rio de Janeiro, RJ, Dezembro de 2004.

[10] S. G. Mallat and Z. Zhang, “Matching Pursuits with Time-frequency Dictionaries”, In IEEE Transactions on Signal Processing, vol. 41, No. 12, pp. 3397-3415, December 1993.

[11] M. Vetterli and J. Kovacevic, “Wavelets and Subband Coding”, Prentice-Hall, 1995.

[12] F. S. Pinagé, “Avaliação do Desempenho de Algoritmos de Compressão de Imagens Usando Recorrência de Padrões Multiescalas”, tese de mestrado, Programa de Engenharia Elétrica, COPPE/UFRJ, Rio de Janeiro, RJ, Julho de 2005.

[13] A. K. Jain, “Fundamentals of Digital Image Processing”, Prentice-Hall, 1989.

[14] ISO/IEC JTC1/SC29/WG11, “JPEG2000 Verification Model 5.3 (Technical Description)”, November 1999.

[15] T.-H. Lan and A. H. Tewfik, “MultiGrid Embedding (MGE) Image Coding”, In 1999 IEEE International Conference on Image Processing, pp. 369-373, Kobe, October 1999.

[16] E. A. B. da Silva, D. G. Sampson and M. Ghanbari, “A Successive Approximation Vector Quantizer for Wavelet Transform Image Coding”, In IEEE Transactions on Image Processing, vol. 5, No. 2, pp. 299-310, February 1996.

[17] E. A. B. da Silva, D. A. Fonini Jr. and M. Craizer, “Successive Approximation Quantization for Image Compression”, In IEEE Circuits and Systems Magazine, pp. 20-45, 2002.

[18] N. M. M. Rodrigues, E. A. B. da Silva, M. B. de Carvalho et al., “Better Adaptive Dictionaries for Recurrent Pattern Image Coding”, submetido à IEEE Transactions on Image Processing, 2007.