COMPRESSÃO DE SINAIS MULTI-DIMENSIONAIS USANDO RECORRÊNCIA DE PADRÕES MULTIESCALAS.

Murilo Bresciani de Carvalho

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

Aprovada por:

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

Prof. Weiler Alves Finamore, Ph.D.

Prof. Gelson Vieira Mendonça, Ph.D.

Prof. Sergio Lima Neto, Ph.D.

Prof. Prof. Abraham Alcaim, Ph.D.

Prof. Marcelo da Silva Pinho, D. Sc.

RIO DE JANEIRO, RJ - BRASIL

MARÇO DE 2001

CARVALHO, MURILO BRESCIANI DE

Compressão de Sinais Multi-dimensionais usando Recorrência de Padrões Multiescalas [Rio de Janeiro] 2001

XI, 197 p., 29,7 cm (COPPE/UFRJ, D.Sc., Engenharia Elétrica, 2001)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Compressão de dados 2. Processamento Digital de Sinais

I. COPPE/UFRJ II. Título (série)

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

COMPRESSÃO DE SINAIS MULTI-DIMENSIONAIS USANDO RECORRÊNCIA DE PADRÕES MULTIESCALAS

Murilo Bresciani de Carvalho

Março/2001

Orientadores: Eduardo Antonio Barros da Silva
Weiler Alves Finamore

Programa de Engenharia Elétrica

Este trabalho propõe e analisa o desempenho de métodos de compressão de dados com perdas baseados em casamento aproximado de padrões multiescalas recorrentes. O casamento de padrões multiescalas aqui proposto é uma extensão do casamento de padrões convencional através da qual se torna possível aproximar um vetor por outro de comprimento diferente. São propostos dois novos algoritmos de compressão de dados com perdas chamados UMMP e MMP. Ambos são baseados no casamento aproximado de padrões multiescalas recorrentes e se utilizam de um dicionário que é construído a medida que o sinal de entrada é codificado. Os results-dos obtidos demonstram a eficácia do uso de escalas para melhorar o desempenho de sistemas baseados em casamento aproximado de padrões.

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

COMPRESSION OF MUITI-DIMENSIONAL SIGNALS BASED ON RECURRENT MULTISCALE PATTERNS

Murilo Bresciani de Carvalho

March/2001

Advisors: Eduardo Antonio Barros da Silva
Weiler Alves Finamore

Department: Electrical Engineering

This work proposes methods for lossy data compression based on approximate pattern maching with scales. The pattern matching with scales is an extension of ordinary pattern matching where a vector can be replaced by another vector of a different length. Two new lossy compression algorithms based in this new matching technique are proposed. Both employ a dictionary that is adaptively buit while encoding the input data. Simulation and theoretical results show that the method is effective to improve the performance of lossy compression methods based on approximate pattern matching.

Agradeço a todos que possibilitaram a realização deste trabalho: aos meus orientadores Prof. Eduardo Antonio Barros da Silva e Prof. Weiler Alves Finamore; ao Programa de Engenharia Elétrica da UFRJ; ao Departamento de Engenharia de Telecomunicações da UFF; aos colegas do LPS.

Sumário

Lista de Figuras

Capítulo 1

Introdução

Hoje, existe uma tendência para digitalização. O áudio digital substituiu os formatos analógicos em quase todas as aplicações. Métodos de processamento de vídeo digital são usados em HDTV e DVD. O custo decrescente dos chips VLSI e o desenvolvimento constante de técnicas de processamento digital de sinais de alto desempenho sugerem um uso ainda maior dos sistemas digitais no futuro. Portanto, é importante desenvolver representações eficientes de sinais digitais. Imagens digitais são usadas em diferentes áreas tais como medicina, prospecção geológica, entretenimento e multi-mídia. Devido ao grande volume de dados em uma imagen digital típica, os algoritmos de compressão de dados são de vital importância em aplicações envolvendo imagens digitais. Apesar da pesquisa em compressão de imagens já ser feita por cerca de 30 anos, o problema da compressão ainda permanece aberto.

Neste trabalho, nós propomos uma nova classe de algoritmos de compressão baseados em *casamento aproximado de padrões recorrentes usando escalas*, isto é, os algoritmos tentam representar um bloco de dados de entrada usando versões dilatadas ou contraídas de blocos previamente ocorridos. Dois tipos principais de algoritmos são estudados: o MMP ("Multi-dimensional Multiscale Parser") e o UMMP ("Universal Multiscale Matching Pursuits"). O algoritmo MMP usa versões dilatadas e contraídas de vetores pertencentes a um dicionário para representar um bloco de dados de entrada. Ele constrói o dicionário usando concatenações de blocos previamente codificados. O algoritmo UMMP é similar ao MMP mas procura representar um bloco de dados de entrada usando uma *combinação linear* de versões contraídas e dilatadas dos vetores do dicionário. Portanto, o UMMP faz uma de-

composição do sinal de entrada em uma base que é construída enquanto o dado é codificado.

O capítulo 2 apresenta uma definição formal do problema da compressão, medidas do conteúdo de informação de uma fonte e algumas soluções clássicas do problema da compressão.

O capítulo 3 contém um estudo das propriedades do casamento aproximado de padrões Gaussianos multi-escalas que justifica a construção dos algoritmos MMP e UMMP, bem como forma uma base para análise de desempenho.

No capítulo 4 nós apresentamos descrições do algoritmo UMMP e de suas variações. O desempenho destes algoritmos é avaliado por meio de simulações em computador.

O capítulo 5 contém descrições do algoritmo MMP e suas variações. O desempenho destes algoritmos é avaliado por meio de modelos analíticos e simulações em computador, usando uma variedade de imagens digitais como fonte. Os desempenhos do MMP e do UMMP são comparados aos de outros sistemas de compressão.

No capítulo 6 nós introduzimos uma versão otimizada do MMP, segundo um critério de taxa versus distorção. Resultados de simulações são usados para ilustrar a melhoria de desempenho obtida.

O efeito de blocagem no MMP é abordado no capítulo 7, onde é desenvolvida uma técnica de redução do efeito de blocagem por meio de pós-filtragem adaptativa.

O capítulo 8 mostra resultados da aplicação do MMP à compressão de imagens previamente transformadas por uma DWT ("Discrete Wavelet Transform").

Nós concluímos no capítulo 9 onde é feito um resumo dos principais resultados obtidos. Alguns problemas abertos relacionados são também apresentados, bem como sugestões para futuros desenvolvimentos.

Capítulo 2

O problema da compressão

Diante do problema da compressão, nós somos levados a indagar o quanto é possível comprimir um sinal em particular. A resposta a esta questão depende em geral do grau de precisão que consideramos necessário para representar o sinal: É necessário reproduzir um sinal exatamente igual ao original ou basta uma representação aproximada? Em outras palavras, nós queremos saber quantos bits são necessários para descrever uma fonte de dados com uma dada qualidade de reprodução. Este problema foi inicialmente estudado por Shannon, que criou um um campo de estudo chamado teoria da informação [1, 2]. Na próxima seção iremos rever alguns resultados clássicos de um ramo da teoria da informação conhecido como "teoria taxa-distorção" [3] que estuda a compressão de uma fonte sujeita a um critério de fidelidade.

2.1 Teoria taxa-distorção

A teoria taxa-distorção [3] estuda a compressão de uma fonte sujeita a um critério de fidelidade. Nesta teoria uma fonte é caracterizada pelas suas propriedades estatísticas. Nos últimos 50 anos, os resultados obtidos na teoria da taxa-distorção concentraram-se em grande parte na derivação de limitantes de desempenho relacionando a taxa de codificação com a distorção que pode ser obtida para uma dada fonte. Esta relação entre a taxa de codificação e a distorção é conhecida como função $\mathsf{R}(\mathsf{D})$. A função $\mathsf{R}(\mathsf{D})$ de uma fonte especifica o numero *médio* de bits necessários para representar qualquer saída produzida pela fonte com distorção *média* menor

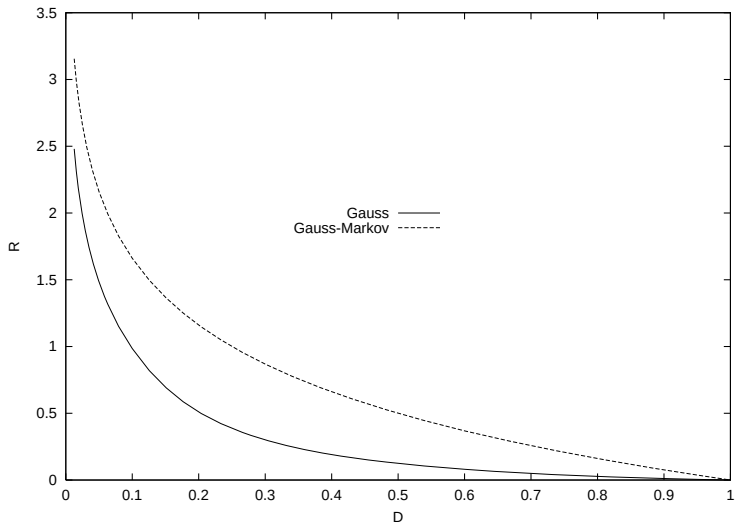


Figura 2.1: Função $R(D)$ para a fonte Gaussiana ($\sigma^2 = 1$) e para a fonte Gauss-Markov ($\sigma^2 = 1, \rho = 0.78$).

ou igual a D. Pode-se mostrar que a função $R(D)$ de qualquer fonte é uma função decrescente e convexa definida no intervalo $[0, D_{max}]$ [2]. D_{max} é a menor distorção média que pode ser obtida utilizando-se zero bits para especificar qualquer saída produzida pela fonte (na taxa zero, utiliza-se sempre o mesmo padrão para representar qualquer saída). Em outras palavras, se nós queremos mais qualidade, nós temos que pagar o preço usando mais bits. A função $R(D)$ estabelece o limite de desempenho de qualquer código de compressão de dados, ou seja a taxa $R^{\mathcal{C}}$ de qualquer código $\mathcal{C}$ tem de respeitar a relação $R^{\mathcal{C}} \geq R(D)$.

Por exemplo, uma fonte Gaussiana é caracterizada estatisticamente por uma variável aleatória (VA) [4] x cuja densidade de probabilidades é uma função Gaussiana. A função $R(D)$ de uma fonte Gaussiana de média $\mu = 0$ e variância σ^2, quando a distorção é medida pelo erro médio quadrático, é dada por:

$$R(D) = \frac{1}{2} \log \left(\frac{\sigma^2}{D} \right) \tag{2.1}$$

Esta função está ilustrada na figura 2.1. Podemos observar que:

1. A menor distorção média que pode ser obtida se tentarmos representar qualquer saída X produzida por esta fonte usando zero bits é $D_{max} = \sigma^2$. Neste caso, é fácil ver que o melhor valor para representar qualquer X produzido pela fonte é o valor da média, que é igual a zero.

2. A taxa necessária para representar esta fonte com distorção zero é infinita. Isto é esperado, uma vez que uma saída X desta fonte pode assumir qualquer valor real, ou seja $X \in \mathbb{R}$, requerendo precisão infinita para ser completamente especificado.

Uma sequência infinita de VAs Gaussianas independentes igualmente distribuidas $x(n) = \{\ldots, x_{-1}, x_0, x_1, \ldots\}$ caracteriza uma *fonte Gaussiana sem memória*. Como o número de varáveis aleatórias desta fonte é infinito, o número de bits necessários para representar qualquer saída $X(n) = \{\ldots, X_{-1}, X_0, X_1, \ldots\}$ produzida por ela é infinito, com excessão do ponto $R(D_{max}) = 0$. Assim sendo, define-se uma função $R(D)$ que especifica a taxa média *por símbolo* necessária para representar a fonte com distorção média *por símbolo* menor ou igual a D. Pode-se mostrar que a função $R(D)$ da fonte Gaussiana sem memória é dada pela equação 2.1, ou seja idêntica à função $R(D)$ da fonte Gaussiana.

Uma sequência de varáveis aleatórias Gaussianas correlatadas caracteriza uma *fonte Gaussiana com memória*. A função $R(D)$ de uma fonte Gaussiana com memória de Markov, definida no apêndice B, está ilustrada na figura 2.1. Nota-se que a introdução de memória reduziu o valor da taxa necessária para especificar a fonte a qualquer distorção (exceto para $D = D_{max}$). Este comportamento é típico e ocorre para qualquer tipo de fonte.

Uma fonte pode produzir em sua saída símbolos que pertencem a um alfabeto contável $\mathcal{A}$. Este tipo de fonte é chamada de fonte discreta e é caracterizada por uma varável aleatória discreta. Analogamente ao caso Gaussiano, podemos também definir fontes discretas com e sem memória, caracterizadas por sequências de VAs correlatadas ou não, respectivamente. Quando o alfabeto $\mathcal{A}$ possui um número finito de elementos, a fonte discreta possui a importante propriedade $R(D) < \infty$, ou seja, é possível especificar qualquer saída desta fonte discreta com distorção nula usando um número finito de bits por símbolo. Neste caso é comum designar-se o valor $R(0)$ como *entropia* da fonte.

Detalhes adicionais sobre a função $R(D)$ de fontes Gaussianas e não Gaussianas são dados no apêndice B.

2.2 Soluções clássicas para o problema da compressão

Existem diversos métodos pra comprimir uma fonte discreta com alfabeto $\mathcal{A}$ finito com distorção nula e taxa média arbitrariamente próxima da entropia da fonte. A compressão com distorção igual a zero é dita *sem perdas*. O código de Huffman, o codificador aritmético e o código de Lempel-Ziv são exemplos de métodos de compressão sem perdas. A taxa de tais métodos aproxima a entropia da fonte, pelo menos quando o número de símbolos da fonte processados é grande. O código de Huffman e o codificador aritmético requerem o conhecimento de um modelo estatístico para a fonte para serem implementados. Isto significa que eles são casados a uma fonte particular e só atingem seu desempenho máximo para aquela fonte. Versões adaptativas destes códigos são inicializadas com modelos padrão que são atualizados a medida que a codificação prossegue. O algoritmo de Lempel-Ziv por outro lado, é *universal* no sentido que nenhum modelo prévio para fonte é necessário e a taxa do código tende para a entropia da fonte quando o número de símbolos codificados tende a infinito. Os métodos de compressão sem perdas também são conhecidos como codificadores de entropia.

2.2.1 Compressão com perdas

Um esquema de compressão de dados com distorção diferente de zero é chamado *com perdas*. Ao contrário do que ocorre no caso sem perdas, não se conhece um método de compressão universal com perdas para o qual a taxa aproxime-se arbitrariamente da $\mathsf{R}(\mathsf{D})$ para qualquer fonte. Assim sendo, é bastante comum dividir-se a solução do problema em dois ou três passos.

Uma solução em dois passos, inicialmente mapeia a fonte $\mathfrak{x}(\mathfrak{n})$ em uma outra fonte de menor entropia $\hat{\mathfrak{x}}(\mathfrak{n})$. A fonte original $\mathfrak{x}(\mathfrak{n})$ inclusive poderia ser definida em um alfabeto ilimitado e possuir $\mathsf{R}(0) \to \infty$. Exige-se contudo que para a fonte $\hat{\mathfrak{x}}(\mathfrak{n})$ tenhamos $\mathsf{R}(0) < \infty$. Em seguida, no segundo passo, a fonte $\hat{\mathfrak{x}}(\mathfrak{n})$ é comprimida sem perdas usando um codificador de entropia qualquer. O passo que mapeia $\mathfrak{x}(\mathfrak{n})$ em $\hat{\mathfrak{x}}(\mathfrak{n})$ é chamado *quantização*. A quantização pode ser escalar ou vetorial, conforme definido no apendice B. Enquanto no caso escalar cada símbolo da fonte é indepen-

dentemente quantizado, no caso vetorial mapeia-se grupos de símbolos emitidos por $x(n)$ em grupos de símbolos de $\hat{x}(n)$.

Uma solução em três passos, é composta de uma etapa de transformação seguida dos passos de quantização e codificação de entropia. Esta técnica é muito popular e vem sendo aplicada com sucesso na construção de codificadores específicos para imagems, vídeo, voz e audio. Varias transformações diferentes podem ser usadas na etapa de transformação. Por exemplo, o popular codificador de imagens JPEG [10] utiliza a DCT ("Discrete Cosine Transform") enquanto o codificador JPEG2000 [11] utiliza a DWT ("Discrete Wavelet Transform"). Estas transformações, bem como suas propriedades, estão descritas no apendice B. O desempenho de uma transformação no contexto de um codificador em três passos é em geral dependendente do tipo do sinal de entrada. Por exemplo, a DWT possui vetores base bem localizados espacialmente enquanto a DCT possui vetores base bem localizados no domínio da frequência. Em geral a DWT apresenta bom desempenho com imagens mas para sinais com componentes bem localizadas na freqência pode ser superada pela DCT.

2.3 O algoritmo "Matching Pursuits"

Um algoritmo de particular interesse para este trabalho e que pode ser usado na etapa de transformação é o "Matching Pursuits" (MP). Este algoritmo faz uma expansão de um sinal em um *frame* [17] arbitrário especificado por um dicionário $\mathcal{D}\{\phi_k\}_{k=0}^{K-1}$. O algoritmo faz refinamentos sucessivos da aproximação para o sinal de entrada usando projeções nos elementos de $\mathcal{D}$.

A decomposição começa escolhendo-se o valor k_0 que minimiza o produto interno $< \phi_{k_0}, \mathbf{x} >$. A seguir calcula-se o *resíduo* $R^1\mathbf{x}$ como:

$$\begin{aligned} R^0\mathbf{x} &= \mathbf{x} \\ R^1\mathbf{x} &= R^0\mathbf{x} - < \phi_{k_0}, \mathbf{x} > \phi_{k_0} \end{aligned} \tag{2.2}$$

Este resíduo $R^1\mathbf{x}$ é então expandido de maneira similar à $\mathbf{x}$. No passo n nós teremos:

$$R^{n+1}\mathbf{x} = R^{n}\mathbf{x} - < \phi_{k_n}, \mathbf{x} > \phi_{k_n} \tag{2.3}$$

Depois de N passos, o vetor $\mathbf{x}$ pode ser aproximado como:

$$\hat{\mathbf{x}} = \sum_{i=0}^{N-1} < \phi_{k_i}, \mathbf{x} > \phi_{k_i} \tag{2.4}$$

O dicionário do MP é redundante, ou seja, possui mais elementos do que os necessários para gerar o espaço que contém o sinal de entrada. Assim sendo, em geral nem todos os elementos do dicionário serão usados na expansão de um sinal de entrada particular. por isso, ao contrário do que ocorre com transformações como a DWT e a DCT, a representação no domínio transformado deve especificar não só os valores dos coeficientes $< \phi_{k_i}, \mathbf{x} >$ mas também o índice k_i do elemento correspondente no dicionário.

Uma das vantagens do uso do MP na etapa de transformação é que o dicionário pode ser escolhido de modo a combinar as propriedades desejáveis de outras transformações. Por exemplo, podemos incluir no dicionário algumas funções base bem localizadas espacialmente, como na DWT, e outras bem localizadas no domínio da frequência, como na DCT.

No capítulo 4 iremos descrever um algoritmo de compressao de dados adaptaativo, chamado UMMP, baseado no MP mas que constrói o seu próprio frame para expandir o sinal de entrada enquanto codifica. Este algoritmo agrupa em um único passo as etapas de transformação, quantização e codificação de entropia.

Capítulo 3

Probabilidade de casamento aproximado de padrões Gaussianos

Neste capítulo nós investigamos as propriedades de casamento de vetores Gaussianos. Nosso objetivo é compreender melhor o funcionamento do casamento aproximado de padrões multiescalas, justificando o seu emprego em um sistema de compressão de dados com perdas.

3.1 Casamento com vetores Gaussianos em um dicionário

Seja $\mathbf{x} = (x_1 x_2 \dots x_N)^T$ um vetor Gaussiano de média nula e matriz covariância $\mathbf{K}_x$ e seja $\mathcal{D} = \{\mathbf{y}^1, \mathbf{y}^2, \dots, \mathbf{y}^M\}$ um dicionário onde $\mathbf{y}^i = (y_1^i y_2^i \dots y_N^i)^T$ são vetores Gaussianos de média nula e matriz covariância $\mathbf{K}_{y^i} = \mathbf{K}_x$. Sejam $\mathbf{x}$ e $\mathbf{y}^i, i \in 1, M$ mutuamente independentes [4].

Nós diremos que ocorreu o evento "casamento" quando pelo menos um dos vetores do dicionário puder ser usado para representar $\mathbf{x}$ com distorção menor ou igual a Nd^*. Se o critério de fidelidade é o erro médio quadrático, então ocorre um casamento se:

$$||\mathbf{x} - \mathbf{y}^i||^2 \leq Nd^* \text{ para algum } i \in [1, M] \tag{3.1}$$

A probabilidade de casamento P_m é então:

$$P_m = Pr\{||\mathbf{x} - \mathbf{y}^i||^2 \leq Nd^* \text{ para algum } i \in [1, M]\} \tag{3.2}$$

ou

$$P_m = 1 - Pr\{||\mathbf{x} - \mathbf{y}^i||^2 > Nd^* \text{ para todo } i \in [1, M]\} \tag{3.3}$$

No apêndice C deduz-se as seguintes expressões para a probabilidade de casamento:

$$P_m(N, M, d^*) = \int_{-\infty}^{\infty} \dots \int_{-\infty}^{\infty} P_{m|\mathbf{X}}(N, M, d^*, \mathbf{X}) p_x(\mathbf{X}) d\mathbf{X} \tag{3.4}$$

$$P_{m|\mathbf{X}}(N, M, d^*) = 1 - \left(1 - F_{z_i|\mathbf{X}}(Nd^*, \mathbf{X})\right)^M, \tag{3.5}$$

onde:

$$F_{z_i|\mathbf{X}}(Z_i, \mathbf{X}) = \mathcal{L}^{-1}\left\{\frac{1}{s} M_{z_i}(s)\right\},$$

e

$$M_{z_i}(\mathbf{s}) = |\mathbf{I}_N + 2s_i\mathbf{K}_x|^{-\frac{1}{2}} e^{-s_i\mathbf{X}^T(\mathbf{I}_N + 2s_i\mathbf{K}_x)^{-1}\mathbf{X}}$$

A equação 3.4 fornece o valor da probabilidade de casamento aproximado, com erro médio quadrático de até Nd^*, de um vetor Gaussiano de comprimento N com um dos vetores de um dicionário de M vetores Gaussianos de mesmo comprimento.

A equação 3.5 fornece o valor da probabilidade de casamento aproximado condicionada a uma particular realização $\mathbf{X}$ do vetor aleatório Gaussiano $\mathbf{x}$.

Caso o vetor Gaussiano possua componentes independentes e identicamente distribuídas, a matriz covariância de $\mathbf{x}$ será dada por $\mathbf{K}_x = \sigma_x^2\mathbf{I}_N$. Neste caso, as expressões para a probabilidade de casamento 3.4 e para a probabilidade de casamento condicionada 3.5 reduzem-se a (ver apêndice C para detalhes):

$$Pm(N, M, d^*) = \int_0^{\infty} P_{m|\lambda}(N, M, d^*, \lambda) \frac{\lambda^{\frac{N}{2}-1} e^{-\frac{\lambda}{2\sigma_x^2}}}{(2\sigma_x^2)^{\frac{N}{2}}\Gamma(\frac{N}{2})} d\lambda \tag{3.8}$$

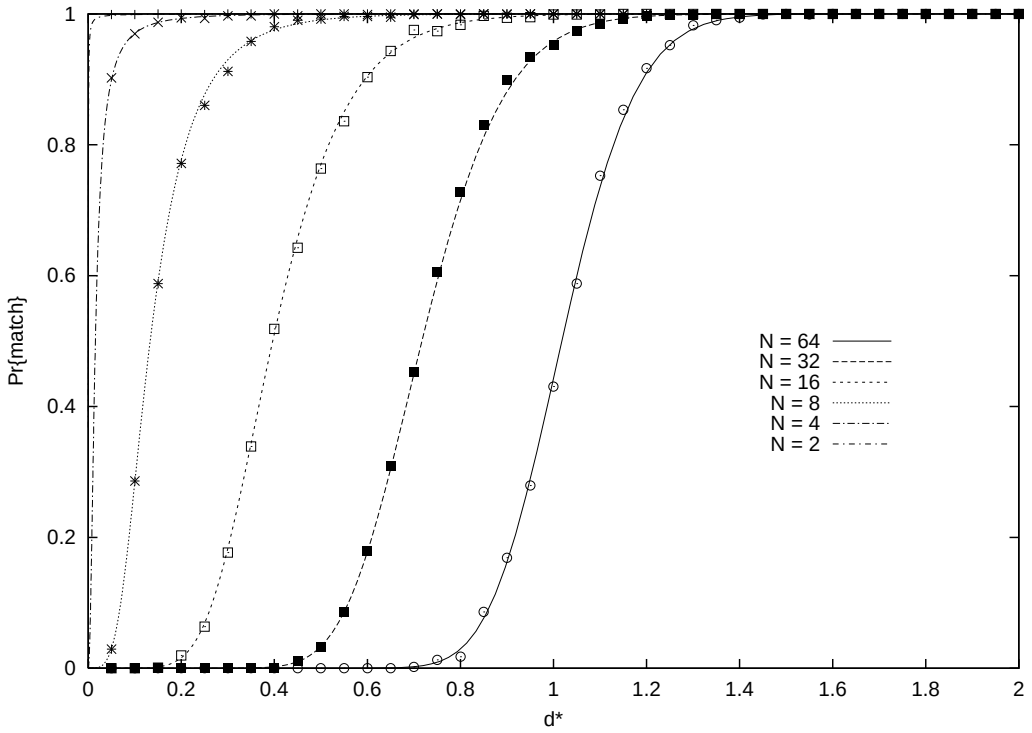


Figura 3.1: Probabilidade de casamento $Pm \times$ distorção alvo d^*, dicionário Gaussiano aleatório.

$$P_{m|\|\mathbf{X}\|^2}(N, M, d^*, \|\mathbf{X}\|^2) = 1 - e^{-\frac{M(Nd^* + \|\mathbf{X}\|^2)}{2\sigma_x^2}} \left(\sum_{k=0}^{\infty} \frac{1}{k!} \left(\frac{\|\mathbf{X}\|^2}{2\sigma_x^2} \right)^k \sum_{l=0}^{k+\frac{N}{2}-1} \frac{1}{l!} \left(\frac{Nd^*}{2\sigma_x^2} \right)^l \right)^M \tag{3.9}$$

Observa-se que, no caso sem memória, a probabilidade de casamento condicionada não depende da forma do vetor $\mathbf{X}$, mas apenas da sua norma ao quadrado $\|\mathbf{X}\|^2$.

A figura 3.1 ilustra as curvas teóricas de probabilidade de casamento dadas pela equação 3.9 para $N = 64, 32, \ldots, 2$ e $M = 8192$ comparadas a resultados experimentais (pontos sobre as curvas). Os experimentos foram feitos em MatLab com 1024 repetições para cada ponto.

Conforme podemos ver na figura 3.1, a probabilidade de casamento diminui (para um dado valor de d^*) a medida que o comprimento N dos vetores aumenta, considerando-se que o número M de vetores do dicionário é constante. Isto é esperado. Se M é fixo, gastaremos no máximo $\log_2(M)$ bits para representar um vetor. A medida que o comprimento N aumenta, usa-se cada vez menos bits em média para

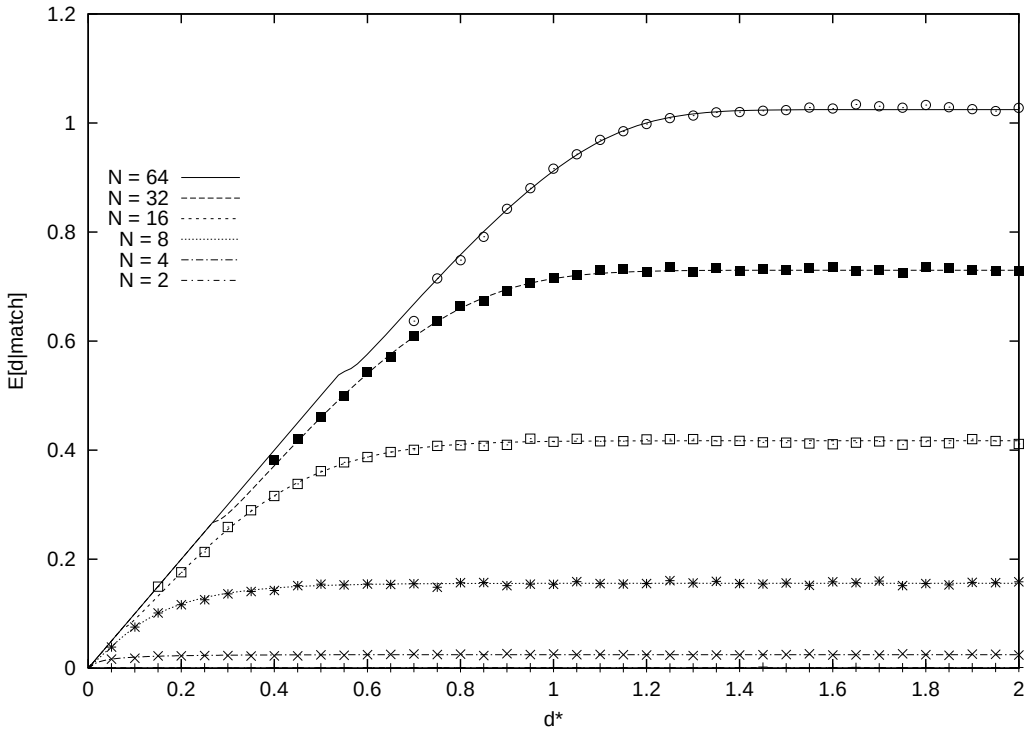


Figura 3.2: Distorção média D_m × distorção alvo d^*, dicionário Gaussiano aleatório.

cada componente, ou seja $\log_2(M)/N$ bits/componente. De acordo com a função $R(D)$ da fonte Gaussiana, definida pela equação 2.1, quanto menos bits se gasta, maior deve ser a distorção. Portanto a probabilidade de casamento deve diminuir a medida que N cresce para que a distorção média aumente conforme previsto pela $R(D)$. De fato, no apêndice C determina-se o valor esperado da distorção dado que ocorreu um casamento como:

$$D_m(N, M, d^*) = E\{d|match\} = d^* - \frac{1}{F_d(d^*)} \int_0^{d^*} P_m(N, M, \xi) d\xi \qquad (3.10)$$

A figura 3.2 mostra as curvas do valor esperado da distorção dado pela equação 3.10 para $N = 64, 32, \ldots, 2$ e $M = 8192$ comparadas a resultados experimentais.

Conforme pode ser observado na figura 3.2, a distorção aumenta a medida que N aumenta, mantendo-se o número M de vetores no dicionário fixo. É interessante observar que para $N = 64$, a distorção ultrapassa $D_{max} = \sigma_x^2$, o valor máximo previsto pela $R(D)$ da fonte Gaussiana. Um sistema de codificação ótimo nunca ultrapassaria esta distorção que só seria atingida com taxa $R = 0$. Isto ocorre porque

não existe nenhuma garantia de que um dicionário Gaussiano aleatório contenha o vetor nulo, que é a solução ótima para este problema de compressão quando $R = 0$. Assim sendo, vale a pena investigar o comportamento do sistema se substituírmos um dos vetores do dicionário pelo vetor nulo. No apêndice C, verifica-se que a probabilidade de casamento condicionada é dada neste caso por:

$$P_{m|\|\mathbf{X}\|^2}(N, M, d^*, \|\mathbf{X}\|^2) = 1 - e^{-\frac{(M-1)(Nd^* + \|\mathbf{X}\|^2)}{2\sigma_x^2}} \cdot \left(\sum_{k=0}^{\infty} \frac{1}{k!} \left(\frac{\|\mathbf{X}\|^2}{2\sigma_x^2} \right)^k \sum_{l=0}^{k+\frac{N}{2}-1} \frac{1}{l!} \left(\frac{Nd^*}{2\sigma_x^2} \right)^l \right)^{M-1} S(\|\mathbf{X}\|^2 - Nd^*) \quad (3.11)$$

A probabilidade de casamento e o valor esperado da distorção dados pelas equações 3.4 e 3.10 continuam válidas neste caso, desde que se substitua o valor da probabilidade de casamento condicionada pela expressão em 3.11.

As figuras 3.3 e 3.4 mostram o comportamento da probabilidade de casamento e do valor esperado da distorção dado um casamento quando o vetor zero substitui um dos vetores do dicionário. A inclusáo do vetor zero faz com que a distorção seja sempre inferior à D_{max}.

Se nós escolhermos a distorção alvo muito grande, $d^* \to \infty$, o casamento ocorrerá com probabilidade 1 e nós sempre usaremos um vetor do dicionário para representar o vetor de entrada. Este é exatamente o funcionamento de um quantizador vetorial. Portanto a expressão do valor esperado da distorção dado casamento, equação 3.10, pode ser usada para calcular o desempenho de um quantizador vetorial de M vetores de dimensão N se fizermos $d^* \to \infty$.

A figura 3.5 mostra o desempenho de um quantizador vetorial com M vetores Gaussianos bem como o desempenho de um QV com $M-1$ vetores Gaussianos mais o vetor zero. A diferença de desempenho entre os dois casos não é grande (0.22 dB a 0.2031 bits/componente) para este valor de $M = 8192$, mas aumenta bastante quando o número de vetores no dicionário diminui. Consideramos a taxa dada por $R = \log_2(M)/N$ bits/componente.

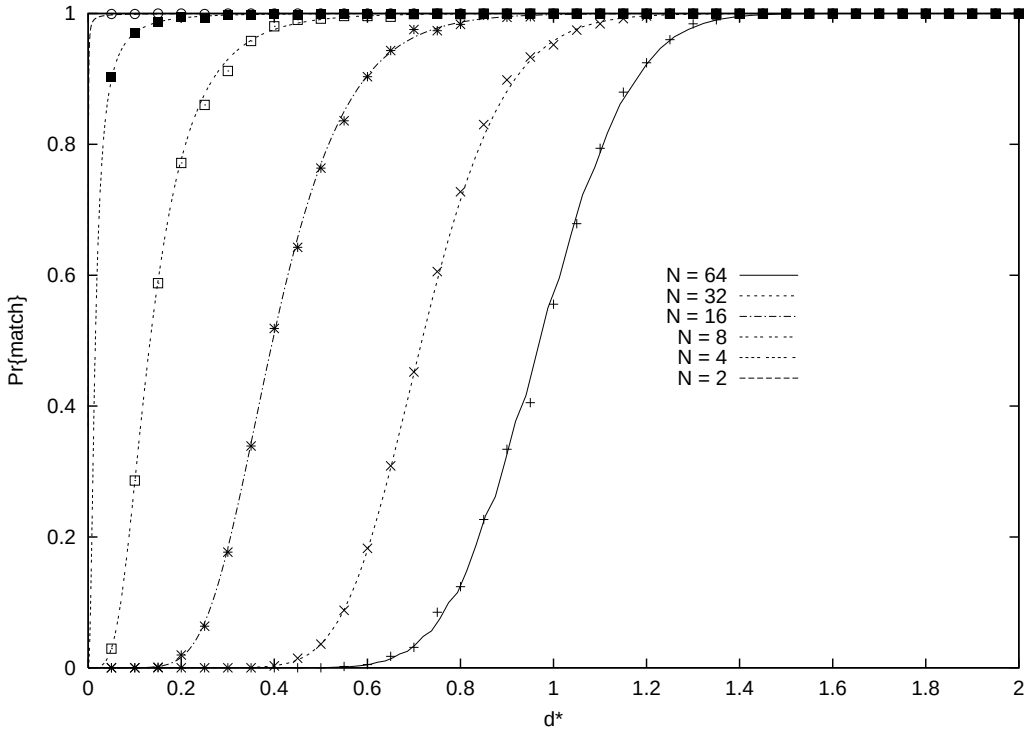


Figura 3.3: Probabilidade de casamento Pm × distorção alvo d^*, dicionário Gaussiano com zero.

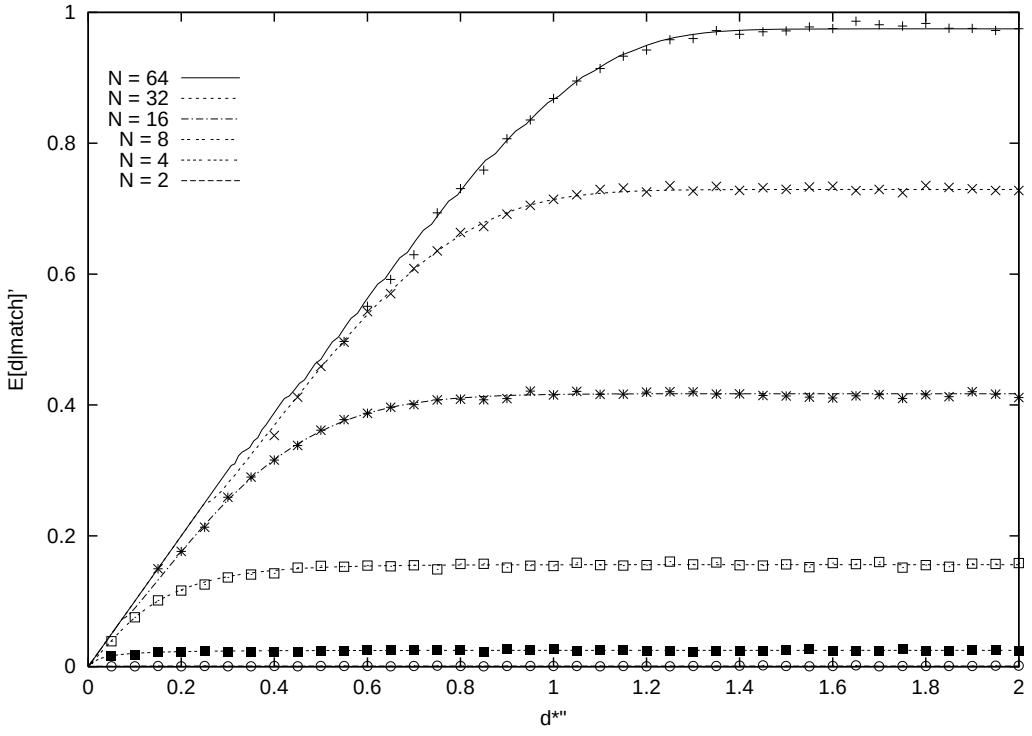


Figura 3.4: Distorção média Dm × distorção alvo d^*, dicionário Gaussiano com zero..

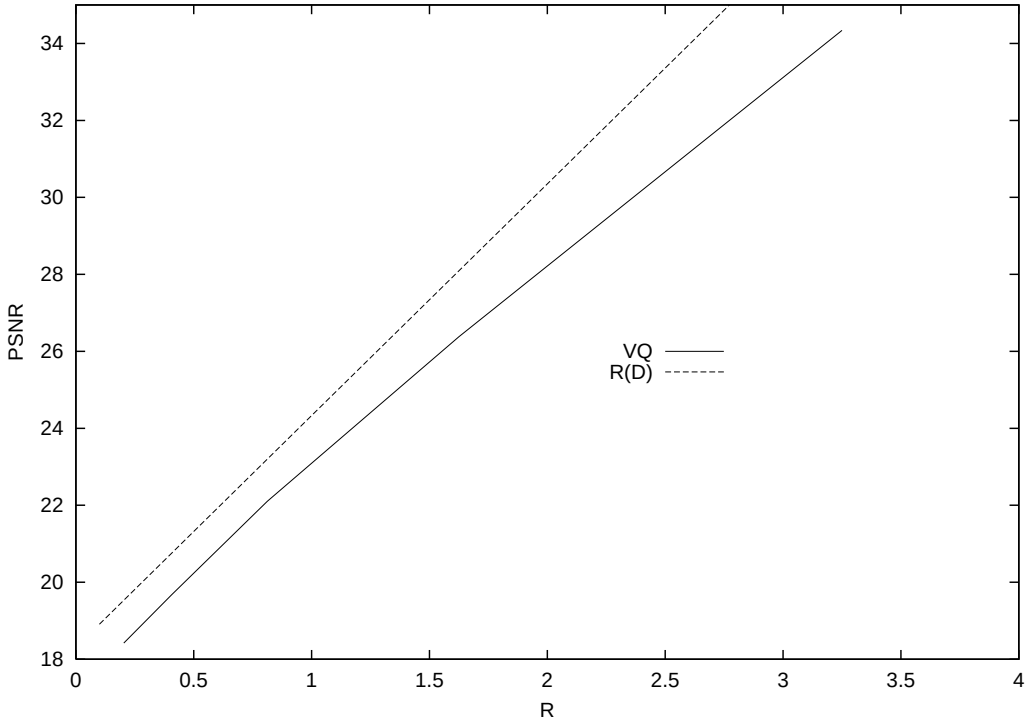


Figura 3.5: Desempenho taxa-distorção de um QV com dicionário de 8192 vetores Gaussianos.

3.2 Casamento aproximado de vetores Gaussianos em escalas diferentes

Uma característica importante deste trabalho é o uso do casamento aprximado de padrões usando escalas. Neste caso estamos interessados em saber como se comporta a probabilidade de casamento se tentarmos aproximar um vetor Gaussiano de comprimento N usando um dicionário de M vetores Gaussianos de comprimento N'. No apêndice C, determina-se a expressão para a probabilidade de casamento como:

$$Pm^s(N, N', M, d^*) = \int_0^{Nd^*} Pm\left(N', M, \frac{Nd^* - \lambda}{N'}\right) \frac{\lambda^{\frac{N-N'}{2}-1} e^{-\frac{\lambda}{2\sigma_x^2}}}{(2\sigma_x^2)^{\frac{N-N'}{2}} \Gamma\left(\frac{N-N'}{2}\right)} d\lambda \quad (3.12)$$

A probabilidade de casamento com escalas está ilustrada na figura 3.6. As curvas ilustram a probabilidade de se obter um casamento de um vetor Gaussiano de comprimento $N = 64$ com algum vetor Gaussiano de comprimento $N' = 2, 4, \ldots, 64$ pertencente a um dicionário de $M = 8192$ vetores. Como N é fixo, o sistema

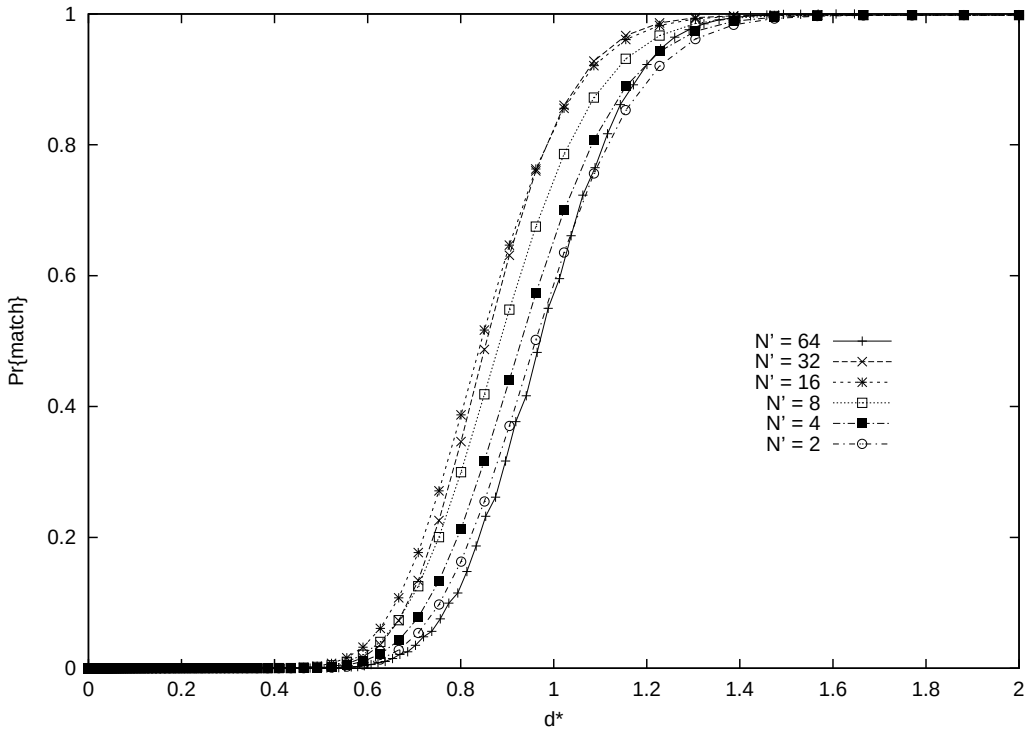


Figura 3.6: Probabilidade de casamento com escalas. $M = 8192$.

está sempre operando a uma taxa de $R = \log_2(M)/N = 0.2031$ bits/componente. É interessante observar que neste caso, a probabilidade de casamento com escalas pode ser *maior* do que a probabilidade de casamento sem escalas. No apêndice C discute-se qualitativamente este comportamento e conclui-se que *o uso de escalas pode melhorar o desempenho de um sistema baseado em casamento aproximado de padrões*. A figura 3.7 ilustra o comportamento do valor esperado da distorção dado casamento com escalas.

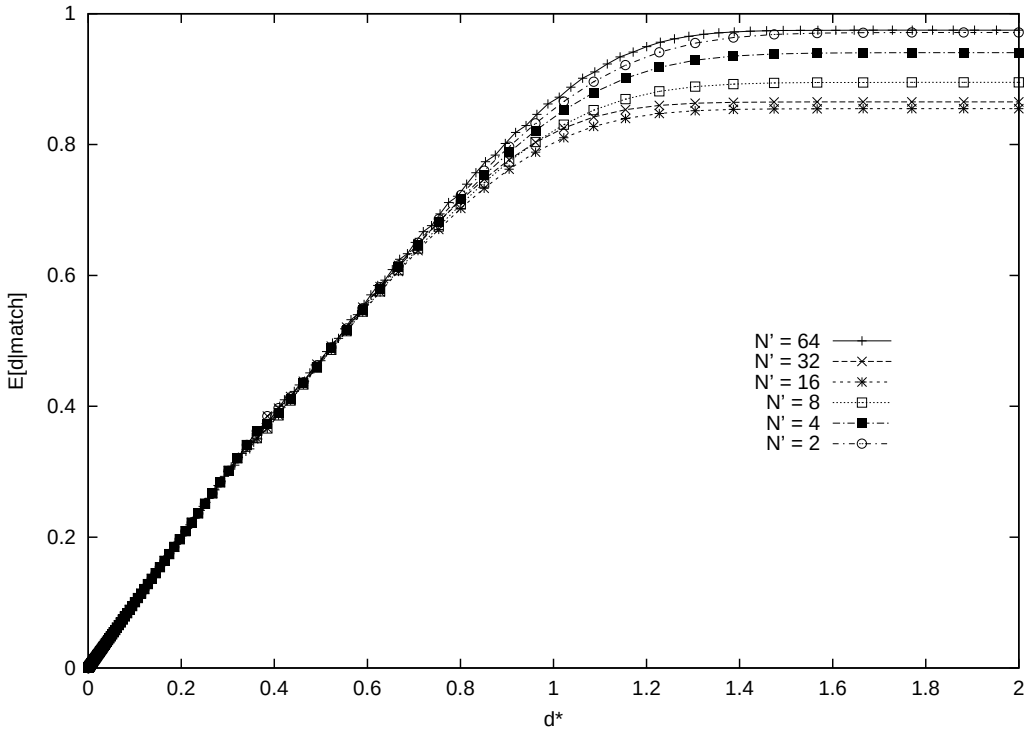


Figura 3.7: Distorção em um casamento com escalas $M = 8192$.

Capítulo 4

UMMP

Neste capítulo nós propomos um algoritmo de compressão de dados chamado UMMP (Universal Multiscale Matching Pursuits). Ele é baseado em *casamento aproximado de padrões recorrentes com escalas*. O algoritmo UMMP constrói um dicionário $\mathcal{D}$ contendo concatenações de versões *contraídas/dilatadas* de padrões previamente codificados enquanto codifica o sinal de entrada. O UMMP usa o dicionário como uma base para decompor o dado de entrada, ou seja, o UMMP procura representar o sinal de entrada como uma combinação linear de elementos do seu dicionário.

4.1 Descrição do UMMP

No capítulo 2 nós apresentamos algumas técnicas para resolver o problema da compressão. Uma das abordagens consistia em dividir a solução em três passos: transformação, quantização e codificação de entropia. O algoritmo "matching pursuits", MP, descrito na seção 2.3, é uma das opções que podem ser usadas na etapa de transformação. Ele opera com um dicionário fixo $\mathcal{D}$ que é totalmente conhecido antes da expansão começar. O dicionário possui redundância, ou seja, contém mais vetores do que os necessários para gerar todo o espaço aonde o sinal de entrada é definido. Num certo sentido a expansão é adaptativa porque o MP escolhe o subconjunto $\mathcal{D}_{\text{opt}}$ do dicionário $\mathcal{D}$ que melhor se presta à expansao do sinal de entrada. No entanto, o nível de adaptação disponivel depende do grau de redundância do dicionário. Se o dicionário é muito redundante, a expansão é muito adaptável mas

o custo de codificar os índices do dicionário também é alto. Seria interessante investigar um método que, ao invés de apenas usar um subconjunto $\mathcal{D}_{\mathrm{opt}}$ do dicionário, procurasse construir $\mathcal{D}_{\mathrm{opt}}$ diretamente enquanto codificasse a fonte. Uma possibilidade é adaptar a técnica de construção de dicionário do algoritmo de compressão de dados sem perdas Lempel-Ziv (LZ) [5].

O algoritmo LZ emprega vetores de comprimento variável pertencentes a um dicionário para codificar blocos de dados extraídos da fonte. O dicionário é construído a medida que os dados vão sendo codificados. Novos vetores para o dicionário são criados pela concatenção de vetores já presentes no dicionário, reproduzindo a ordem de sua ocorrência na sequência de entrada. Este método tende a produzir um dicionário que contém sequências típicas [2] que são quase equiprováveis com $\mathsf{Pr} \approx 2^{-\mathsf{H}(\mathsf{x})}$, onde $\mathsf{H}(\mathsf{x})$ é a entropia da fonte. Deste modo, os índices do dicionário produzem um código com entropia próxima da entropia da fonte.

O algoritmo matching pursuits procura expandir o vetor de entrada $\mathbf{X}$ usando vetores do dicionário $\mathcal{D}$. A expansáo é feita através de refinamentos sucessivos, onde apenas um vetor de $\mathcal{D}$ é usado de cada vez. O MP inicialmente projeta $\mathbf{X}$ nos vetores do dicionário e seleciona aquele que produz o menor resíduo $\mathbf{R}^1$, que é a diferença entre o vetor $\mathbf{X}$ e o resultado da projeção . Caso o resíduo seja considerado suficientemente pequeno, segundo algum critério de fidelidade, a expansão termina. Caso contrário, o resíduo $\mathbf{R}^1$ é projetado em $\mathcal{D}$ e um novo resíduo $\mathbf{R}^2$ é gerado. O procedimento prossegue até que para algum resíduo $\mathbf{R}^{\mathsf{L}}$ o critério de fidelidade seja atingido. Nós propomos modificar o MP, criando um algoritmo de compressão de dados com perdas universal que chamaremos UMMP ("Universal Multiscale Matching Pursuits"), do seguinte modo: inicialmente, o vetor de entrada $\mathbf{X}$ é projetado em $\mathcal{D}$ e um resíduo $\mathbf{R}^1$ calculado como no MP. Se o critério de fidelidade for atingido, a expansão termina. Caso contário, o vetor $\mathbf{R}^1$ será dividido em dois segmentos, $\mathbf{R}^{1,1}$ e $\mathbf{R}^{1,2}$, tais que a concatenação de ambos resulta no resíduo original $\mathbf{R}^1$. Em seguida cada um dos dois segmentos, $\mathbf{R}^{1,1}$ e $\mathbf{R}^{1,2}$, será projetado em $\mathcal{D}$ e assim por diante. O fato de o tamanho dos vetores agora ser variável poderia inviabilizar o cálculo das projeções em $\mathcal{D}$ (como projetar em vetores de tamanhos diferentes?), porém esta dificuldade é facilmente contornável se utilizarmos escalas. Para isso introduzimos uma transformação de escala $\mathsf{T}_{\mathsf{N}}^{\mathsf{M}} : \mathbb{R}^{\mathsf{M}} \rightarrow \mathbb{R}^{\mathsf{N}}$ que mapeia um vetor de comprimento

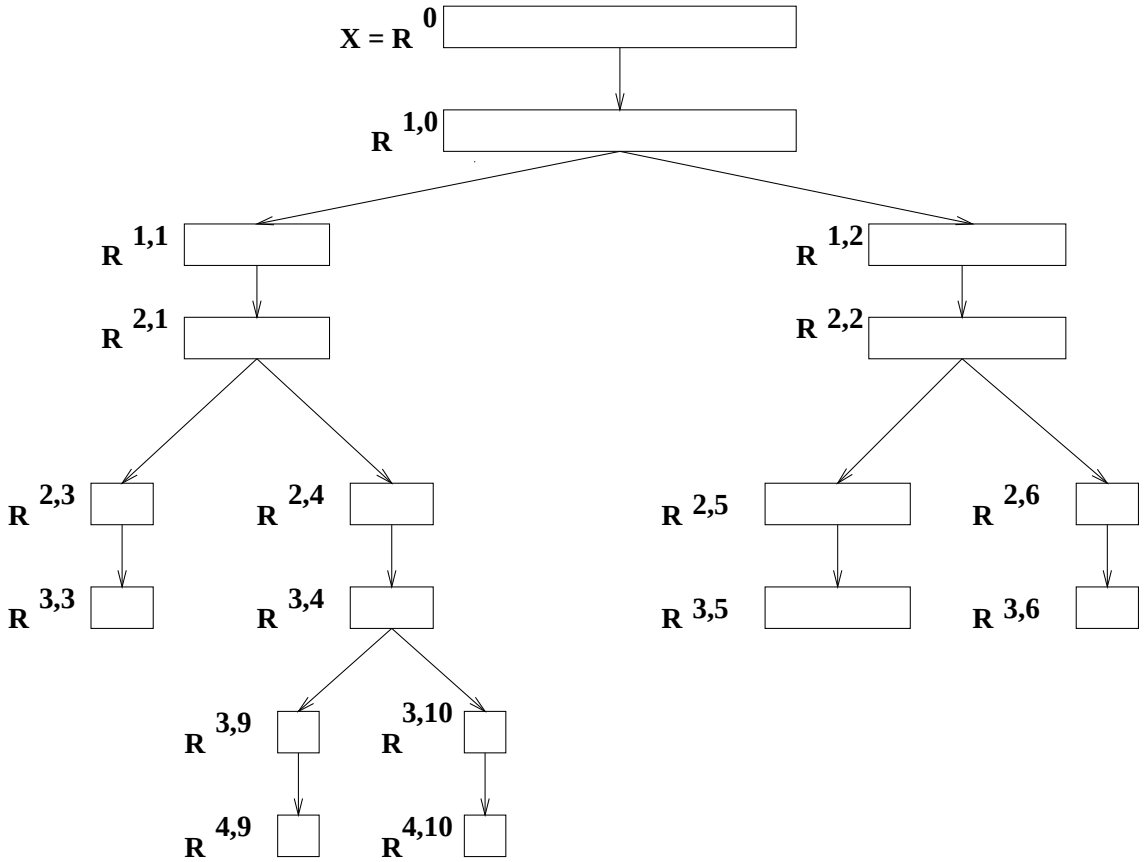


Figura 4.1: Árvore de segmentação do UMMP.

M em um vetor de comprimento N. Para cada novo resíduo $\mathbf{R}^{i,j}$ criado, toma-se a decisão de continuar a expansão, criando dois novos segmentos $\mathbf{R}^{i,2j+1}$ e $\mathbf{R}^{i,2j+2}$ a partir deste resíduo, ou parar. A decisão de parar ou segmentar é tomada de forma independente para cada resíduo. Este procedimento cria uma segmentação do vetor de entrada em vetores de comprimento variável. Esta segmentação pode ser representada por uma árvore binária, como na figura 4.1.

O fato de agora contarmos com vetores de comprimento variável permite que usemos a concatenação de vetores previamente codificados para atualizar o dicionário. Por exemplo, suponha que em um dado instante o algoritmo já possua representações para os dois segmentos $\mathbf{R}^{i+1,2j+1}$ e $\mathbf{R}^{i+1,2j+2}$, denotadas respectivamente por $\hat{\mathbf{R}}^{i+1,2j+1}$ e $\hat{\mathbf{R}}^{i+1,2j+2}$. Então o algoritmo UMMP pode reconstruir uma aproximação $\hat{\mathbf{R}}^{i+1,j}$ para $\mathbf{X}^{i+1,j}$ concatenando os dois segmentos $\hat{\mathbf{R}}^{i+1,2j+1}$ e $\hat{\mathbf{R}}^{i+1,2j+2}$. Este processo está ilustrado na figura 4.2.

A partir de $\hat{\mathbf{R}}^{i+1,j}$, o UMMP pode reconstruir $\hat{\mathbf{R}}^{i,j} = \hat{\mathbf{R}}^{i+1,j} + < \mathbf{R}^{i,j}, \mathbf{S}^{s}_{k_{i,j}} > \mathbf{S}^{s}_{k_{i,j}}$, onde $\mathbf{S}_{k_{i,j}}$ é o vetor do dicionário, escalado para o comprimento de $\hat{\mathbf{R}}^{i,j}$, que

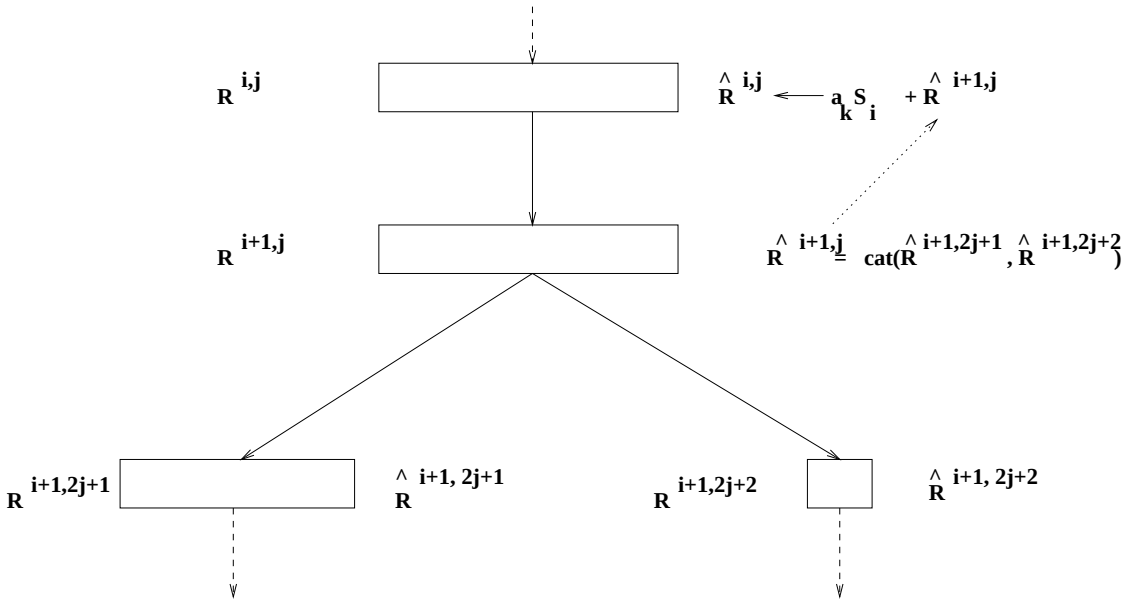


Figura 4.2: Atualização do dicionário do UMMP.

tinha sido anteriormente escolhido como a melhor projeção de $\hat{\mathbf{R}}^{i,j}$ em $\mathcal{D}$.

O algoritmo completo, considerando a distorção medida pelo erro médio quadrático, está descrito a seguir. Sejam:

1) Um dicionário $\mathcal{D}$ de vetores, inicializados com $\mathcal{D}_0 = \{\mathbf{S}_0, \ldots, \mathbf{S}_{M-1}\}$,

2) Um quantizador escalar definido pelo conjunto de valores de reprodução $\mathcal{Q} = \{\alpha_0, \ldots, \alpha_{L-1}\}$,

3) Uma transformação de escala $T_N^M : \mathbb{R}^M \rightarrow \mathbb{R}^N$,

4) Uma distorção alvo d^*.

5) Um vetor de entrada $\mathbf{X}$, usado para inicializar $\mathbf{R}^{0,0} = \mathbf{X}$.

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

passo 1 escale todos os vetores do dicionário para o tamanho de $\mathbf{R}^{i,j}$, denotado por $\ell(\mathbf{R}^{i,j})$,
ou seja: $\mathbf{S}^s_k = T^{\ell(\mathbf{S}_k)}_{\ell(\mathbf{R}^{i,j})}[\mathbf{S}_k]$ para todo k, onde $\ell(\mathbf{S}_k)$ é o comprimento de $\mathbf{S}_k$.

passo 2 encontre $k_{i,j}$ tal que $\frac{\mathbf{S}^T_{k_{i,j}}\mathbf{R}^{i,j}}{\|\mathbf{S}^s_{k_{i,j}}\|}$ é máximo

passo 3 encontre $l_{i,j}$ tal que $\|\mathbf{R}^{i,j} - \alpha_{l_{i,j}}\mathbf{S}^s_{k_{i,j}}\|$ é mínimo e faça $\hat{\mathbf{R}}^{i,j} = \alpha_{l_{i,j}}\mathbf{S}^s_{k_{i,j}}$.

passo 4 escreva na saída $k_{i,j}, l_{i,j}$

passo 5 `se` $\|\mathbf{R}^{i,j} - \hat{\mathbf{R}}^{i,j}\|^2 \leq n d^*$ `então` escreva na saída flag '1' e retorne $\hat{\mathbf{R}}^{i,j}$.
`senão` vá ao passo 6.

passo 6 calcule o resíduo $\mathbf{R}^{i+1,j} = \mathbf{R}^{i,j} - \hat{\mathbf{R}}^{i,j}$

passo 7 escolha o valor de $p \in [0, \ell(\mathbf{R}^{i,j}) - 2]$ que minimize $\|\mathbf{R}^{i+1,2j+1} - \hat{\mathbf{R}}^{i+1,2j+1}\| + \|\mathbf{R}^{i+1,2j+2} - \hat{\mathbf{R}}^{i+1,2j+2}\|$ e então
segmente $\mathbf{R}^{i+1,j} = \begin{pmatrix} \mathbf{R}^{i+1,2j+1} & \mathbf{R}^{i+1,2j+2} \end{pmatrix}$

passo 8 escreva na saída flag '0' e o valor de p.

passo 9 calcule $\hat{\mathbf{R}}^{i+1,2j+1} = \text{codifica}(\mathbf{R}^{i+1,2j+1}, \mathcal{D}, d^*)$ e
$\hat{\mathbf{R}}^{i+1,2j+2} = \text{codifica}(\mathbf{R}^{i+1,2j+2}, \mathcal{D}, d^*)$

passo 10 faça $\hat{\mathbf{R}}^{i+1,j} = \begin{pmatrix} \hat{\mathbf{R}}^{i+1,2j+1} & \hat{\mathbf{R}}^{i+1,2j+2} \end{pmatrix}$

passo 11 atualize $\hat{\mathbf{R}}^{i,j} = \alpha_{l_{i,j}}\mathbf{S}^s_{k_{i,j}} + \hat{\mathbf{R}}^{i+1,j}$

passo 12 atualize $\mathcal{D} = \mathcal{D} \cup (\hat{\mathbf{R}}^{i,j}) \cup (\hat{\mathbf{R}}^{i+1,j})$

passo 13 retorne $\hat{\mathbf{R}}^{i,j}$

Após algumas simulações em computador, verificamos que o UMMP funcionava melhor, do ponto de vista do desempenho taxa-distorção, se os resíduos fossem segmentados em dois vetores de igual comprimento, ou seja, sempre que um resíduo fosse ser segmentado, escolhia-se $p = N/2$, onde N é o comprimento do resíduo. Também verificamos que o algoritmo funcionava melhor se o quantizador

escalar possuísse apenas um nível, $\mathcal{Q} = \{1\}$. Estas observações, bem como outras variações do algoritmo estão detalhadas no apêndice D.

Neste trabalho estamos interessados em estudar o desempenho do UMMP na codificação de imagens. Uma forma de aplicar o UMMP em uma imagem seria converter a matriz de dados bidimensional que representa a imagem em um vetor por meio de algum tipo de varredura. Contudo, isso não é necessário porque o UMMP é trivialmente extensível para dimensões superiores. De fato, se consideramos a entrada como uma matriz $\mathbf{X}$, basta que o ponto de segmentação seja um ponto no espaço bidimensional, com duas coordenadas ou seja, $\mathbf{p} = \begin{pmatrix} p_l & p_c \end{pmatrix}$. Neste caso, um resíduo $\mathbf{R}^{i,j,q}$ seria segmentado em 4 partes, como:

$$\mathbf{R}^{r,j,q} = \begin{pmatrix} \mathbf{R}^{r,2j+1,2q+1} & \mathbf{R}^{r,2j+1,2q+2} \\ \mathbf{R}^{r,2j+2,2q+1} & \mathbf{R}^{r,2j+2,2q+2} \end{pmatrix} \tag{4.1}$$

Assim como no caso unidimensional, uma versão de UMMP bidimensional com segmentação bidimensional fixa funcionou melhor do que a versão com a segmentação genérica dada pela equação 4.1. Detalhes destas implementaçõs do algoritmo bidimensional, 2D-UMMP, estão no apêndice D.

No capítulo 5 apresentaremos uma variação do UMMP que possui melhor desempenho no sentido taxa-distorção, o algoritmo MMP. Neste capítulo, também comparamos o desempenho do UMMP e do MMP ao desempenho de outros algoritmos de compressão com perdas.

Capítulo 5

MMP

Neste capítulo nós propomos um algoritmo de compressão com perdas que nós chamamos MMP. De modo semelhante ao algoritmo UMMP descrito no capitulo 4, ele é baseado no *casmento aproximado de padrões recorrentes multiescalas*. O MMP também constrói um dicionário $\mathcal{D}$ e o atualiza usando concatenações de versões *contraídas/dilatadas* de padrões previamente codificados, enquanto codifica o sinal de entrada. A diferença básica entre o UMMP e o MMP é que o UMMP procura aproximar um segmento extraído do vetor de entrada usando uma *combinação linear* de vetores do dicionário, enquanto o MMP usa um *único vetor* do dicionário para representar o mesmo segmento.

5.1 Descrição do MMP

O algoritmo de compressão com perdas MMP é baseado em um procedimento de segmentação semelhante ao do UMMP, descrito no capítulo 4. Ele utiliza um dicioário de vetores $\mathcal{D}$ e procura aproximar um vetor de entrada $\mathbf{X}$ do seguinte modo: inicialmente o vetor $\mathbf{X}$ é projetado nos vetores do dicionário e o melhor deles, segundo alguma medida de fidelidade, é selecionado para representar $\mathbf{X}$. Caso a aproximação assim obtida seja considerada suficientemente precisa, o procedimento termina. Caso contrário, esta aproximação é *descartada* e o vetor de entrada $\mathbf{X}$ é segmentado em duas partes, $\mathbf{X}^1$ e $\mathbf{X}^2$. Isto diferencia o MMP do UMMP, uma vez que o UMMP sempre utiliza a melhor aproximação para calcular um vetor resíduo. Em seguida cada um dos dois segmentos, $\mathbf{X}^1$ e $\mathbf{X}^2$, será projetado em $\mathcal{D}$ e assim por

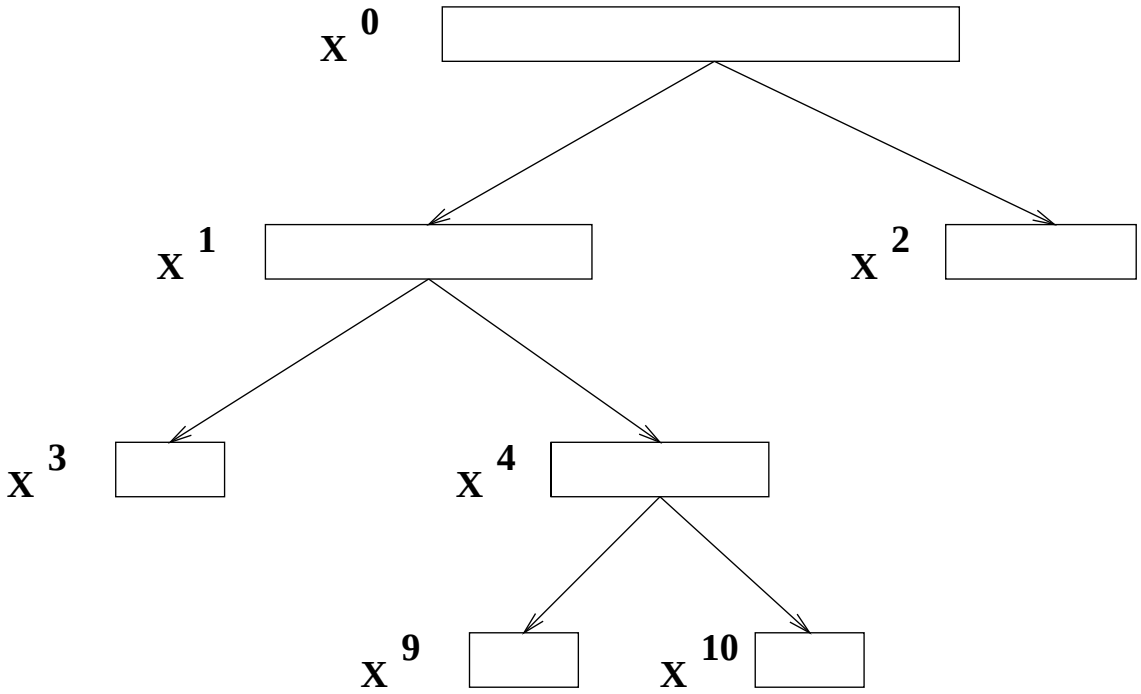


Figura 5.1: Árvore de segmentação do MMP.

diante. Assim como no UMMP, usa-se uma transformação de escala $T_N^M : \mathbb{R}^M \rightarrow \mathbb{R}^N$ para possibilitar a projeção de vetores de comprimentos diferentes nos vetores do dicionário. Para cada novo segmento $\mathbf{X}^j$ criado, toma-se a decisão de continuar a segmentação, criando dois novos segmentos $\mathbf{X}^{2j+1}$ e $\mathbf{X}^{2j+2}$, ou parar. A decisão de parar ou segmentar é tomada de forma independente para cada segmento $\mathbf{X}^j$. Este procedimento cria uma segmentação do vetor de entrada em vetores de comprimento variável. Esta segmentação pode ser representada por uma árvore binária, como na figura 5.1.

A árvore de segmentação da figura 5.1 corresponde à divisão do vetor de entrada em quatro segmentos, $\left(\begin{array}{cccc} \mathbf{X}^3 & \mathbf{X}^9 & \mathbf{X}^{10} & \mathbf{X}^2 \end{array} \right)$. Cada segmento será representado por um vetor no dicionário escalado, obtendo-se $\hat{\mathbf{X}} = \left(\begin{array}{cccc} \hat{\mathbf{X}}^3 & \hat{\mathbf{X}}^9 & \hat{\mathbf{X}}^{10} & \hat{\mathbf{X}}^2 \end{array} \right)$.

O dicionário do MMP pode ser atualizado de modo semelhante ao do UMMP: sempre que as aproximações para os dois segmentos $\mathbf{X}^{2j+1}$ and $\mathbf{X}^{2j+2}$ estiverem disponíveis, o MMP forma uma aproximação para $\mathbf{X}^j$ usando concatenação, ou seja, $\hat{\mathbf{X}}^j = \left(\begin{array}{cc} \hat{\mathbf{X}}^{2j+1} & \hat{\mathbf{X}}^{2j+2} \end{array} \right)$. No exemplo da figura 5.1, quando $\hat{\mathbf{X}}^9$ e $\hat{\mathbf{X}}^{10}$ estão disponíveis, o MMP pode concatena-los para formar $\hat{\mathbf{X}}^4$. Esta nova aproximação pode então ser incluída no dicionário.

O MMP produz em sua saída um sequência de inteiros correspondendo aos

índices dos elementos do dicionário utilizados para representar cada segmento extraído do vetor de entrada, bem como uma sequência de flags binários usados para descrever a árvore de segmentação. Uma descrição completa do MMP é apresentada a seguir.

Sejam:

1) Um dicionário $\mathcal{D}$ de vetores, inicializado com $\mathcal{D}_0 = \{\mathbf{S}_0, \ldots, \mathbf{S}_{M-1}\}$,

2) Um quantizador escalar definido pelo conjunto de valores de reprodução $\mathcal{Q} = \{\alpha_0, \ldots, \alpha_{K-1}\}$,

3) Uma transformação de escala $T_N^M : \mathbb{R}^M \to \mathbb{R}^N$,

4) Um vetor de entrada $\mathbf{X}$ usado para inicializar $\mathbf{X}^0$.

5) A target distortion d^*.

Procedure $\hat{\mathbf{X}}^j = \text{codifica}(\mathbf{X}^j, \mathcal{D}, d^*)$:

passo 1 escale todos os vetores do dicionário para o tamanho $\ell(\mathbf{X}^j)$ de $\mathbf{X}^j$,
ou seja: $\mathbf{S}_i^s = T_{\ell(\mathbf{X}^j)}^{\ell(\mathbf{S}_i)}[\mathbf{S}_i]$ para todo i.

passo 2 encontre i_j, k_j tais que $\|\mathbf{X}^j - \alpha_{k_j}\mathbf{S}_{i_j}^s\|$ é mínimo e faça $\hat{\mathbf{X}}^j = \alpha_{k_j}\mathbf{S}_{i_j}^s$.

passo 3 `se` $\|\mathbf{X}^j - \hat{\mathbf{X}}^j\|^2 \leq nd^*$ `então` escreva na saída o flag '1', os índices i_j, k_j e retorne $\hat{\mathbf{X}}^j$.
`senão` vá ao passo 4.

passo 4 escolha $p \in [0, \ell(\mathbf{X}^j) - 2]$ para minimizar
$\|\mathbf{X}^{2j+1} - \hat{\mathbf{X}}^{2j+1}\|^2 + \|\mathbf{X}^{2j+2} - \hat{\mathbf{X}}^{2j+2}\|^2$ e então
segmente $\mathbf{X}^j = \begin{pmatrix} \mathbf{X}^{2j+1} & \mathbf{X}^{2j+2} \end{pmatrix}$

passo 5 escreva na saída o flag '0' e o valor de p.

passo 6 calcule $\hat{\mathbf{X}}^{2j+1} = \text{codifica}(\mathbf{X}^{2j+1}, \mathcal{D}, d^*)$ e $\hat{\mathbf{X}}^{2j+2} = \text{codifica}(\mathbf{X}^{2j+2}, \mathcal{D}, d^*)$

passo 7 faça $\hat{\mathbf{X}}^j = \begin{pmatrix} \hat{\mathbf{X}}^{2j+1} & \hat{\mathbf{X}}^{2j+2} \end{pmatrix}$

passo 8 atualize $\mathcal{D} = \mathcal{D} \cup (\hat{\mathbf{X}}^j)$

passo 9 retorne $\hat{\mathbf{X}}^j$

Assim como o UMMP, o MMP é trivialmente extensível para dimensões superiores. Se consideramos a entrada como uma matriz $\mathbf{X}$, basta que o ponto de segmentação seja um ponto no espaço bidimensional, com duas coordenadas ou seja, $\mathbf{p} = \begin{pmatrix} p_l & p_c \end{pmatrix}$. Neste caso, um semento $\mathbf{X}^{j,q}$ seria segmentado em 4 partes, como:

$$\mathbf{X}^{j,q} = \begin{pmatrix} \mathbf{X}^{2j+1,2q+1} & \mathbf{X}^{2j+1,2q+2} \\ \mathbf{X}^{2j+2,2q+1} & \mathbf{X}^{2j+2,2q+2} \end{pmatrix} \tag{5.1}$$

Contudo, assim como ocorreu para o UMMP, uma versão de MMP bidimensional com segmentação bidimensional fixa funcionou melhor do que a versão com a segmentação genérica dada pela equação 5.1. Detalhes desta implementação do algoritmo bidimensional, 2D-MMMP, estão no apêndice E.

5.2 Resultados experimentais

Nesta seção apresentamos resultados experimentais de simulações em computador do MMP e do UMMP. O algoritmo bidimensional 2D-MMP, descrito em detalhes no apêndice E foi implementado e testado usando uma série de imagens digitalizadas como entrada. Mostraremos aqui alguns resultados obtidos para as imagens LENA, PP1209 e PP1205, todas de tamanho 512×512. Estas imagens estaão ilustradas nas figuras J.1, J.9 e J.8 respectivamente. A imagem PP1205 é composta de texto e formulas, obtidos da página 1205 da revista *IEEE Transactions on Image Processing*, Volume 9, number 7, July 2000, capturada por um scanner. A imagem PP1209 é composta de texto, gráficos e duas versões da imagem LENA. Resultados obtidos com outras imagens estão no apêndice E. Também mostramos resultados para os algoritmos bidimensionais 2D-UMMP e 2D-UMMPG, descritos em detalhes no apêndice D. A figura 5.2 contém a curva taxa $\times$ distorção obtida para imagem LENA. Incluimos na figura, para comparação, resultados para o algoritmo SPIHT, que ilustra o desempenho dos codificadores de imagens modernos baseados em wavelets, e para o popular algoritmo JPEG. Também incluímos o desempenho do LLZ como exemplo de um algoritmo de compressão com perdas que

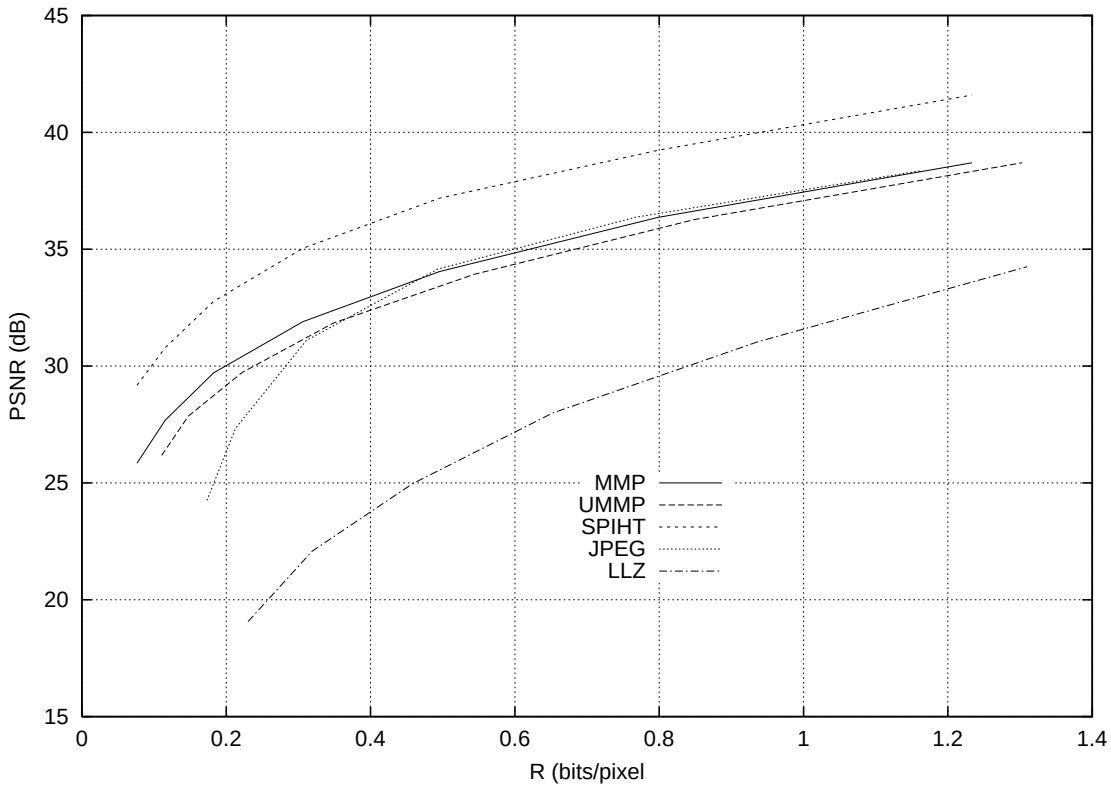


Figura 5.2: Taxa $\times$ distorção do 2D-UMMP e do 2D-MMP para LENA 512×512.

utiliza casamento aproximado de padrões, sem contudo utilizar o conceito de casamento multiescalas. As figuras 5.3 e 5.4 apresentam os resultados para as imagens PP1205 e PP1209 respectivamente.

Pode-se ver da figura 5.2, que o melhor desempenho é do algoritmo SPIHT, seguido do MMP e do JPEG. O algoritmo UMMP apresentou um desempenho um pouco inferior. Por outro lado, para as images PP1205 e PP1209 o MMP superou todos os demais algoritmos. De fato, o desempenho do MMP com imagems derivadas de documentos contendo apenas texto ou texto misturado a imagens é muito bom. No apêndice E mostra-se o desempenho do MMP com outras imagens e curiosamente o pior desempenho relativo, quando comparado ao SPIHT e ao JPEG, ocorre justamente com LENA e BARBARA, duas das imagens mais populares na literatura de compressão de imagens. O desempenho inferior do UMMP pode ser em parte compreendido considerando-se o seguinte exemplo: Suponhamos que o UMMP seja empregado para codificar o vetor de entrada $\mathbf{X} = \begin{pmatrix} 0 & 0 & 2 & 2 \end{pmatrix}$ usando o dicionário $\mathcal{D} = \{-2, -1, 0, 1, 2\}$, quantizador escalar $\mathcal{Q} = \{1\}$, distorção alvo $d^* = 0$ e segmentação fixa $p = \ell(\mathbf{X}^j)/2$. No primeiro passo, o UMMP usará o segundo vetor

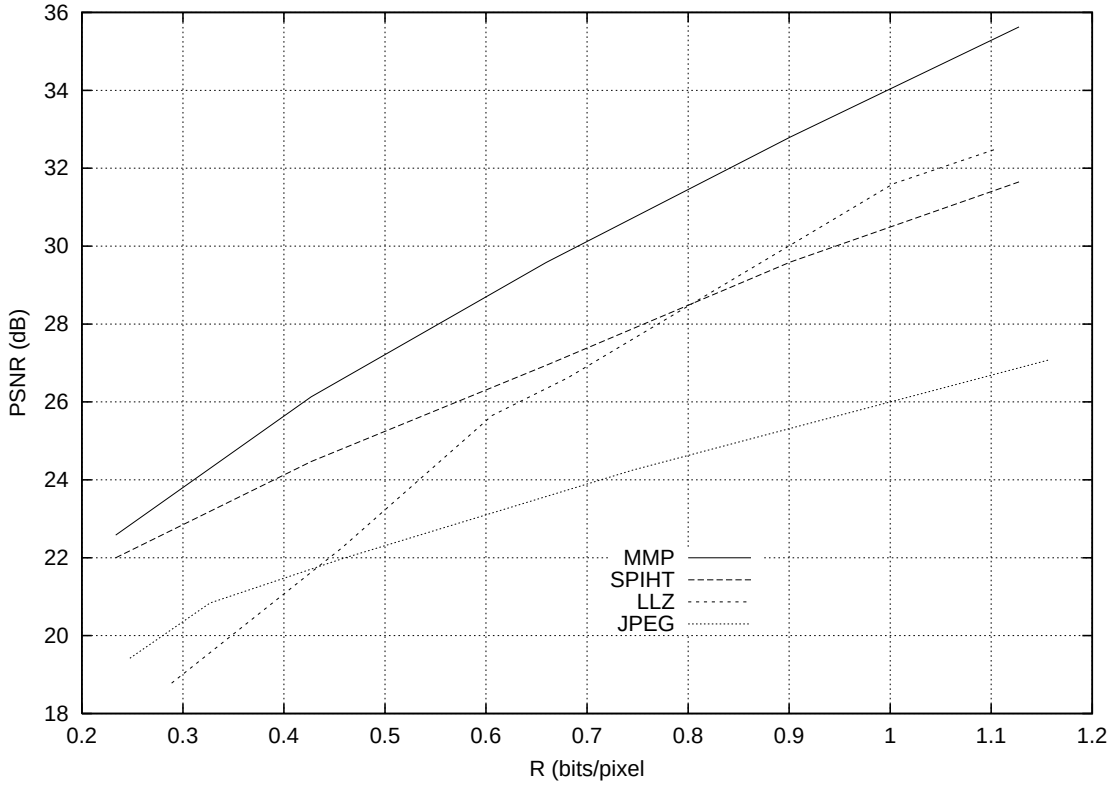


Figura 5.3: Taxa × distorção do 2D-UMMP e do 2D-MMP para PP1205 512×512.

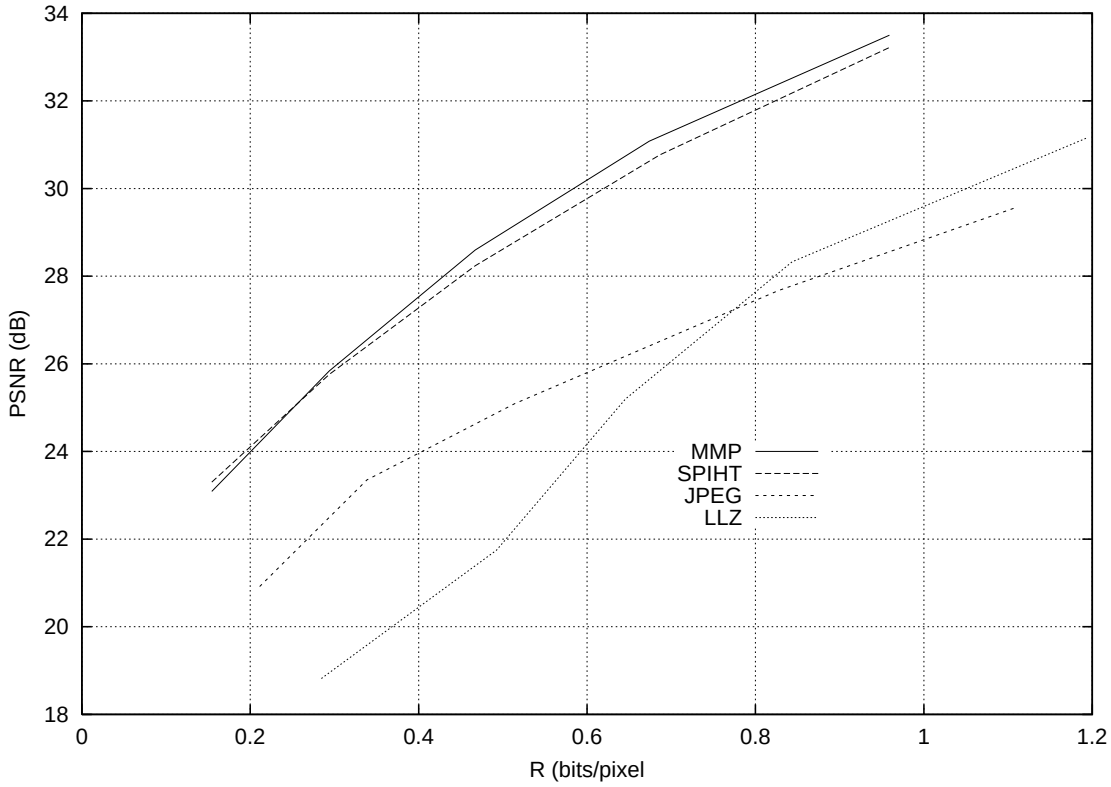


Figura 5.4: Taxa × distorção do 2D-UMMP e do 2D-MMP para PP1209 512×512.

do dicionário, escalado para comprimento 4 ou seja $\mathbf{S}_3^s = \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix}$, para aproximar $\mathbf{X}$. O resíduo neste caso será $\mathbf{R}^{1,0} = \begin{pmatrix} -1 & -1 & 1 & 1 \end{pmatrix}$. Como a norma do resíduo é maior do que $\mathsf{Nd}^* = 0$, o residuo será dividido em dois segmentos, neste caso $\mathbf{R}^{1,1} = \begin{pmatrix} -1 & -1 \end{pmatrix}$ e $\mathbf{R}^{1,2} = \begin{pmatrix} 1 & 1 \end{pmatrix}$. Ambos segmentos podem ser representados com distorção nula pelo dicionário, portanto o procedimento irá gerar a seguinte sequência para representar a entrada: 0, i_3, 1, i_1, 1, i_3 (neste exemplo não é necessário transmitir índices para o quantizador porque ele só possui um elemento). O MMP, usando o mesmo dicionário inicial, o mesmo quantizador e a mesma distorção alvo, faria a aproximação no mesmo número de passos, porém não precisaria transmitir o índice correspondente à primeira aproximação pois ela é descartada. Assim sendo o MMP iria gerar em sua saída a seguinte sequência: 0, 1, i_2, 1, i_4. Neste exemplo, o UMMP removeu a média do sinal $\mathbf{X}$ ao calcular o primeiro resíduo $\mathbf{R}^{1,0}$. Contudo este passo foi inútil uma vez que é possível, *com este dicionário inicial*, representar os dois segmentos $\mathbf{R}^{1,1}$ e $\mathbf{R}^{1,2}$ com distorção nula, independentemente da média ter sido removida ou não. Esta discussão é aprofundada no apêndice E.

Os algoritmos UMMP e MMP descritos até agora, criam suas árvores de segmentação baseados em decisões locais, ou seja, um vetor é segmentado ou não apenas em função da distorção na representação daquele vetor. Não existe garantia de que esta técnica leva ao melhor desempenho, do ponto de vista de taxa $\times$ distorção. No capítulo 6 descreveremos uma versão do MMP com otimização taxa-distorção.

Capítulo 6

Otimização taxa-distorção no MMP

O algoritmo MMP, proposto no capítulo 5, era controlado por um parâmetro de distorção alvo d^*. A árvore de segmentação construída dependia de decisões baseadas em valores locais de distorção. Especificamente, a subdivisão de um segmento era interrompida quando a distorção na aproximação do segmento era menor ou igual a d^* vezes o comprimento do segmento. Nós podemos esperar uma melhoria no desempenho se as decisões de segmentação forem baseadas em um critério global e não local. Neste capítulo nós descrevemos uma versão do algoritmo MMP com segmentação otimizada segundo um critério de taxa $\times$ distorção.

6.1 Otimização da árvore de segmentação

O MMP é um algoritmo de compressão de dados com perdas que se utiliza de um dicionário de vetores $\mathcal{D}$ e de um procedimento de segmentação para codificar um vetor de entrada $\mathbf{X}$. O vetor $\mathbf{X}$ é subdividido em L segmentos $\mathbf{X} = \left(\begin{array}{ccc} \mathbf{X}^{l_0} & \ldots & \mathbf{X}^{l_{L-1}} \end{array} \right)$ de comprimento $\ell(\mathbf{X}^{l_m})$, $m = 0, 1, \ldots, L-1$. Cada segmento é codificado usando-se um elemento do dicionário $\mathcal{D}$ escalado para o tamanho apropriado usando uma transformação de escala, conforme descrito no capítulo 5. Obtém-se assim uma aproximação para $\mathbf{X}$ como $\hat{\mathbf{X}} = \left(\begin{array}{ccc} \mathbf{S}^s_{i_0} & \ldots & \mathbf{S}^s_{i_{L-1}} \end{array} \right)$. A segmentação pode ser representada por uma árvore de segmentação, conforme ilustrado na figura 6.1.

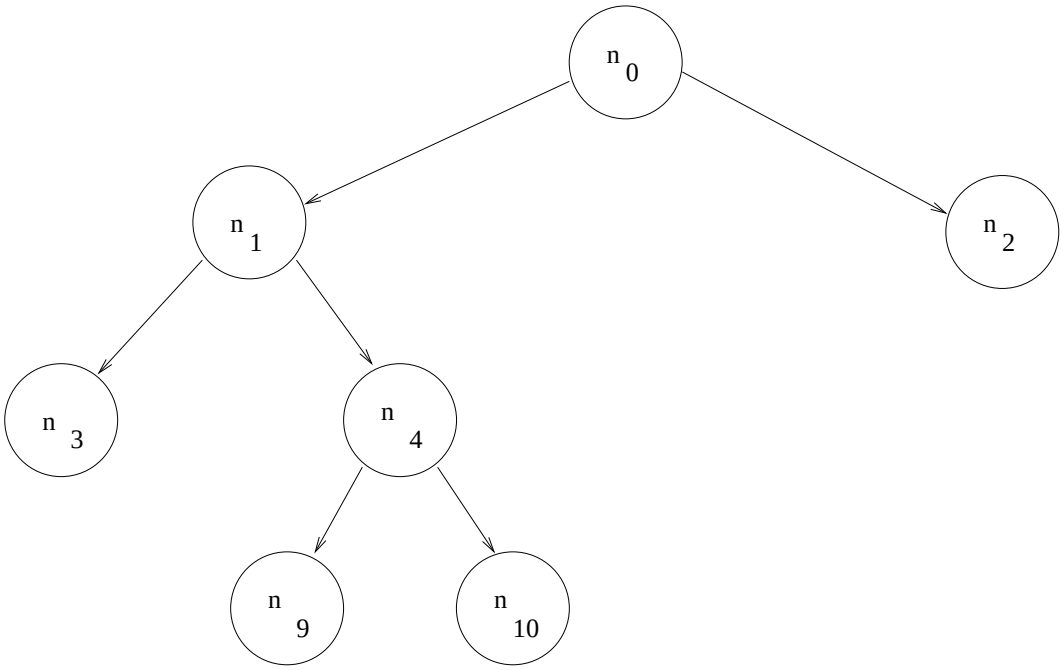


Figura 6.1: Uma árvore de segmentação.

Cada nodo folha da árvore, isto é, cada nodo sem descendentes, está associado a um segmento do vetor de entrada. Por exemplo, a árvore da figura 6.1 representa a subdivisão do vetor de entrada em 4 segmentos, ou seja $\mathbf{X} = \begin{pmatrix} \mathbf{X}^3 & \mathbf{X}^9 & \mathbf{X}^{10} & \mathbf{X}^2 \end{pmatrix}$. Cada segmento é aproximado por um vetor do dicionário escalado, com alguma distorção, e nós podemos associar ao nodo folha correspondente da árvore este valor de distorção. De um modo geral, todos os nodos da árvore estão associados a algum segmento do vetor $\mathbf{X}$, mesmo que este segmento não seja utilizado na representação final. Por exemplo, na árvore da figura 6.1, o nodo n_1 está associado a um segmento $\mathbf{X}^1$ que não é utilizado porque foi segmentado como $\mathbf{X}^1 = \begin{pmatrix} \mathbf{X}^3 & \mathbf{X}^9 & \mathbf{X}^{10} \end{pmatrix}$. Mesmo assim podemos associar uma distorção a este nodo, que é dada pela melhor aproximação que podemos fazer para $\mathbf{X}^1$ usando o dicionário. De modo geral, definimos a distorção $D(n_l)$ associada ao nodo n_l como:

$$D(n_l) = d\left(\mathbf{X}^l, \mathbf{S}_{i_l}^{(s)}\right) \tag{6.1}$$

Também definiremos $R(n_l)$ como a taxa necessária para especificar o índice i_l, ou seja:

$$R(n_l) = -\log_2(Pr(i_l)) \tag{6.2}$$

onde $Pr(i_l)$ é a probabilidade de ocorrência do índice i_l.

Também definimos o custo Lagrangeano como:

$$J(n_l) = D(n_l) + \lambda R(n_l) \tag{6.3}$$

No apêndice F nós deduzimos um algoritmo que escolhe a melhor árvore de segmentação no sentido taxa-distorção, ou seja, a uma dada taxa o algoritmo encontra a árvore que leva à menor distorção ou alternativamente, a menor taxa à uma dada distorção. Este algoritmo está reproduzido abaixo.

passo 1 Inicialize $\mathcal{S}$ como a árvore binária completa de profundidade $\log_2(M) + 1$.

passo 2 Faça $J_i = \infty, i = M-1, M, \ldots, 2M-2$.

passo 3 Faça $p = \log_2(M)$.

passo 4 Para cada nodo $n_l \in \mathcal{S}$ na profundidade p, ou seja $l \in [2^{p-1}-1, 2^p-2]$, calcule $J_l = J(n_l) + \lambda R_{1_l}$ onde $J(n_l)$ é o custo para representar o segmento de entrada associado ao nodo n_l e R_{1_l} é a taxa necessária para indicar que o nodo n_l é uma folha. Avalie $\Delta J_l = \sum_{n_r \in \mathcal{S}-\mathcal{S}(n_l)} J(n_r) - \sum_{n_r \in \mathcal{S}-\mathcal{S}(n_l)} J'(n_r)$, onde $J'(n_l)$ é computado usando o dicionário do qual o vetor $\hat{\mathbf{X}}^l = \left(\begin{array}{cc} \hat{\mathbf{X}}^{2j+1} & \hat{\mathbf{X}}^{2j+2} \end{array} \right)$ foi removido. Se $J_l - J_{2l+1} - J_{2l+2} - \lambda R_{0_l} \leq \Delta J_l$ então poda-se os nodos n_{2l+1} e n_{2l+2} (R_{0_l} é a taxa usada para indicar segmentação, e J_{2l+1}, J_{2l+2} foram ambos calculados na iteração anterior com $p+1$). Caso contrário, o custo do nodo n_l é atualizado com $J_{2l+1} + J_{2l+2} + \lambda R_{0_l}$. Faça $p = p - 1$.

passo 5 Repita o passo 4 até que $p = 0$.

passo 6 Se $\mathcal{S}_0 = \mathcal{S}$ então a otimização está terminada. Caso contrário, faça $\mathcal{S}_0 = \mathcal{S}$ e volte ao passo 3.

No algoritmo acima, usamos a notação $\mathcal{S} - \mathcal{S}(n_l)$ para denotar a árvore que contém todos os nodos de $\mathcal{S}$ exceto n_l e todos os seus descendentes.

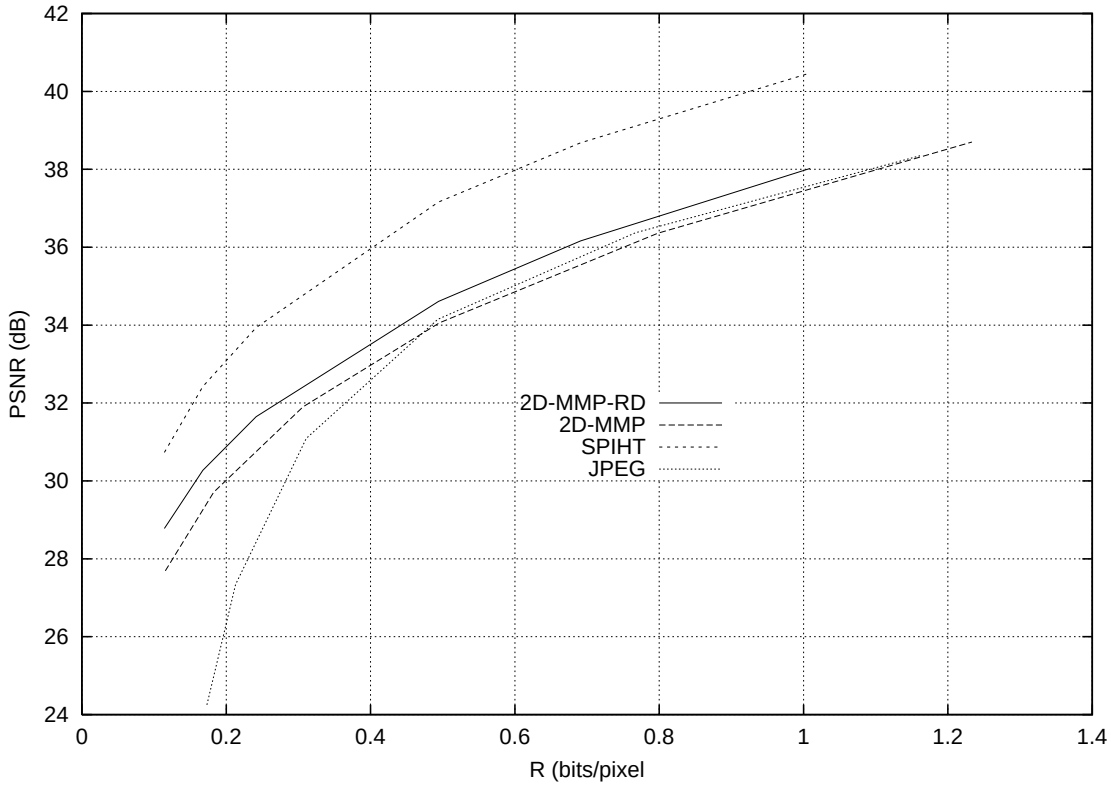


Figura 6.2: Taxa $\times$ distorção do 2D-MMP-RD para LENA 512×512.

6.2 Resultados experimentais

Nós utilizamos o algoritmo de otimização acima em conjunto com o 2D-MMP descrito no capítulo 5 para comprimir uma série de imagens de teste. Nós chamamos o algoritmo otimizado de 2D-MMP-RD e detalhes de sua implementação estão no apêndice F.

Conforme esperado, os resultados do 2D-MMP-RD foram bem superiores aos do MMP, para todas as imagens de teste utilizadas. Nas figuras 6.2, 6.3 e 6.4 apresentamos os resultados para as imagens LENA, PP1205 e PP1209 respectivamente.

No apêndice F, apresentamos resultados de simulações com outras imagens de teste.

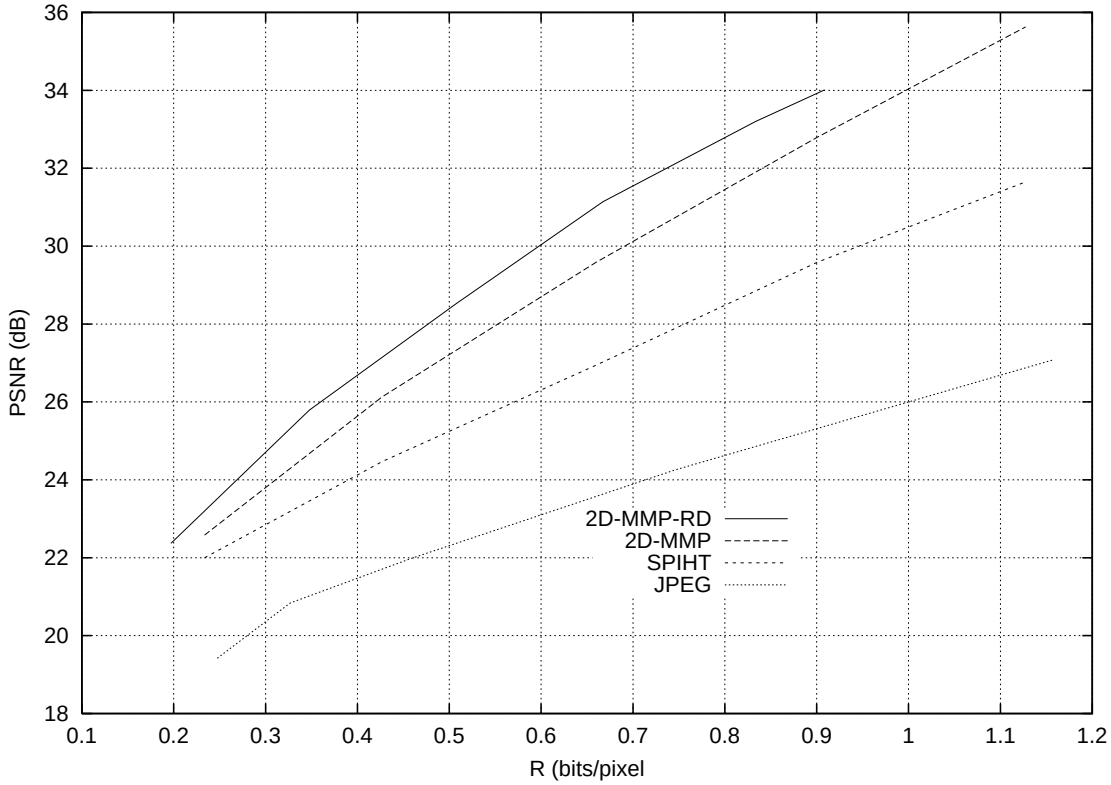


Figura 6.3: Taxa $\times$ distorção do 2D-MMP-RD para PP1205 512×512.

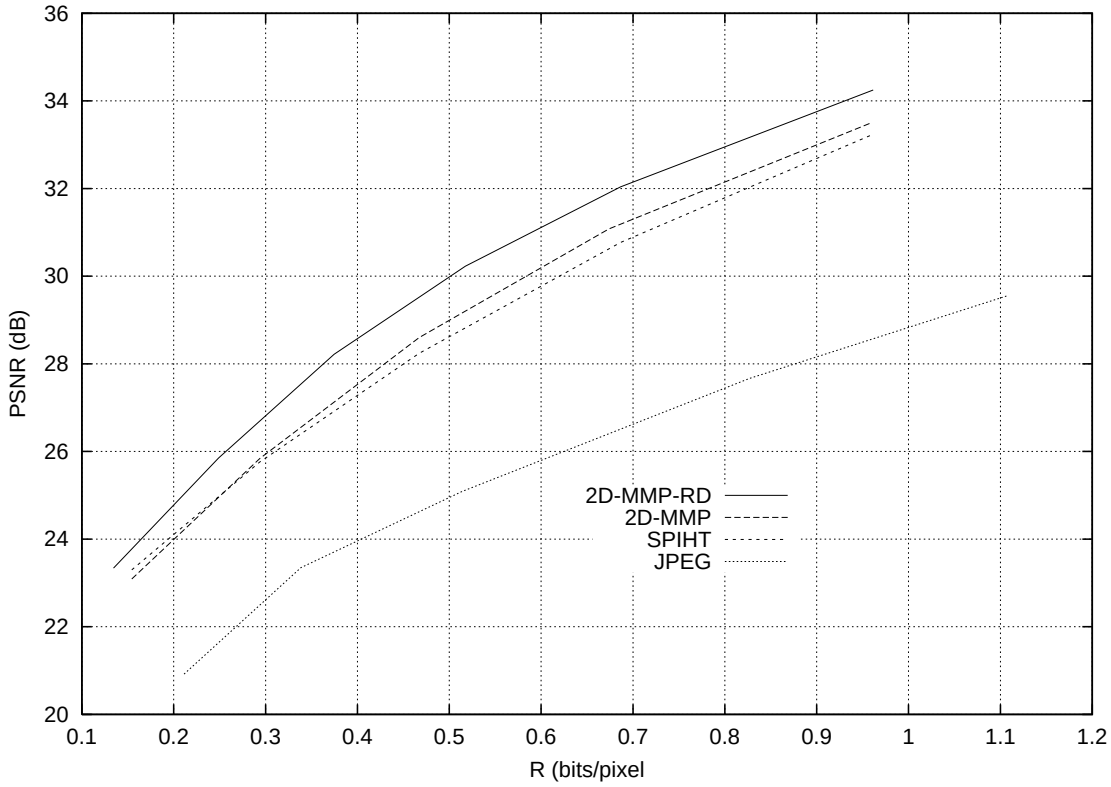


Figura 6.4: Taxa $\times$ distorção do 2D-MMMP-RD para PP1209 512×512.

Capítulo 7

O efeito de blocagem no MMP

Neste capítulo nós pesquisamos formas de controlar o efeito de blocagem no MMP. O efeito de blocagem é um fenômeno peculiar que os codificadores em blocos, como o MMP, tendem a apresentar. Este efeito pode ser bastante desagradável visualmente em aplicações de compressão de imagens. Ele é causado por descontinuidades artificialmente criadas pelo codificador nas fronteiras dos blocos, uma conseqüência comum do processamento independente dos blocos. O efeito tende a ser mais pronunciado em baixas taxas devido à maior distorção na reprodução. De fato, em taxas baixas e maiores distorções, maiores afastamentos dos valores originais são tolerados favorecendo o aparecimento de descontinuidades entre os blocos.

A figura G.2 no apêndice G ilustra a imagem LENA 512×512 codificada à 0.2415 bits/pixel pelo 2D-MMP-RD. O efeito de blocagem é bem aparente.

7.1 Controle do efeito de blocagem no MMP

O procedimento de segmentação usado no MMP divide o vetor de entrada $\mathbf{X}$ em L segmentos $\mathbf{X} = \left(\begin{array}{ccc} \mathbf{X}^{l_0} & \ldots & \mathbf{X}^{l_{L-1}} \end{array} \right)$. Cada segmento é independentemente aproximado usando vetores pertencentes a um dicionário apropriadamente escalados. Não existe em princípio nenhuma restrição à escolha dos vetores de reprodução que leve em conta a continuidade no ponto de segmentação. Uma forma de controlar o efeito de blocagem é alterar o critério de fidelidade de modo a considerar explicitamente a descontinuidade no ponto de segmentação. Uma outra possibilidade é utilizar blocos com superposição. Nesta abordagem, a soma dos comprimentos dos

L segmentos é maior que o comprimento total do vetor de entrada N. Dois segmentos adjacentes não são apenas concatenados, mas somados com superposição de algumas componentes. Contudo, quando se utiliza superposição no codificador, torna-se mais difícil selecionar a melhor árvore de segmentação porque a distorção associada a um nodo é dependente dos vetores de reprodução associados aos segmentos adjacentes. No apêndice G propomos uma solução, baseada em uma técnica de pós-filtragem adaptativa, que implementa superposição no decodificador apenas.

7.2 Resultados experimentais

O algoritmo de controle de blocagem foi implementado no decodificador do 2D-MMP-RD e aplicado às imagens de teste. A figura G.3, no apêndice G, mostra o resultado obtido para a imagem LENA codificada pelo 2D-MMP-RD à 0.2415 bits/pixel. Uma comparação com a figura G.2 deixa evidente a eficácia do metodo em reduzir o efeito de blocagem. O desempenho objetivo, medido pela curva taxa $\times$ distorção, piorou cerca 0.20 dB ao longo de toda curva com a pós filtragem, conforme pode ser visto na figura 7.1.

Não fizemos nehuma tentativa de otimizar a resposta ao inpulso dos filtros usados no decodificador. Contudo, um dos filtros experimentados, de resposta aproximadamente Gaussiana e definido no apêndice G, melhorou o desempenho objetivo aumentando a PSNR para todas as imagens de teste, exceto as que continham texto, ou seja, PP1205 e PP1209. A melhoria para a imagem LENA pode ser vista no gráfico da figura 7.1.

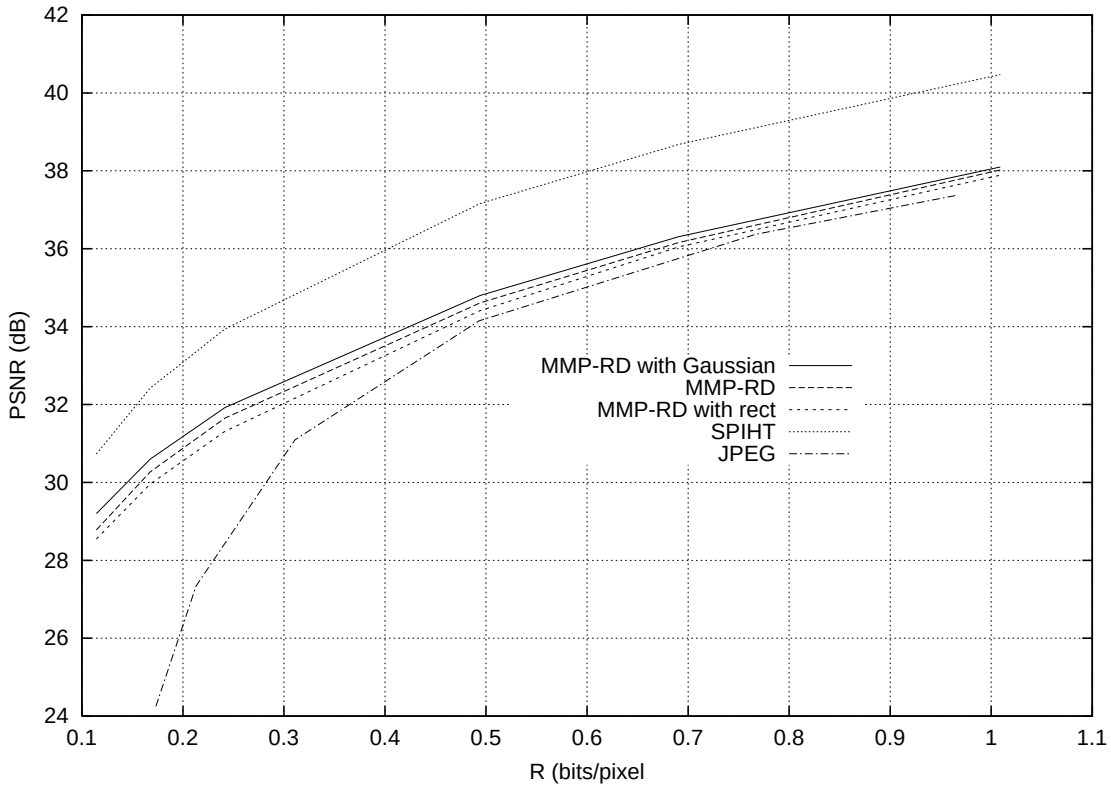


Figura 7.1: Desempenho objetivo do decodificador mofificado. Imagem LENA

Capítulo 8

MMP aplicado a imagens transformadas por DWT

Neste capítulo nós usamos o MMP para comprimir imagens previamente transformadas por uma DWT ("Discrete Wavelet Transform"). Esta transformada está descrita no apêndice B, e é utilizada em diversos codificadores de imagens de alto desempenho. Para imagens com alto conteúdo de baixas frequências a DWT consegue compactar a maior parte da energia em poucos coeficientes de baixa frequência. Se a imagen possui arestas delimitando objetos em uma cena, a energia correspondente é concentrada em alguns coeficientes de alta frequência espacialmente agrupados. Regiões de textura complexa por outro lado tendem a contribuir com energia para todas as bandas da DWT igualmente. Muitos algoritmos de compressão baseados em wavelet e de alto desempenho como o EZW [12], o SPIHT [13] o MGE [14], o SWEET [15] e o JPEG2000 [11] utilizam uma estratégia simples de quantização escalar dos coeficientes associada a métodos eficientes para localizar os coeficientes de maior energia e codifica-los. A eficiência destes métodos de localização depende do tipo de imagem e pressupõe que a maior parte da energia está concentrada nas baixas frequências enquanto o restante da energia, de alta frequência, está concentrada nas arestas e não em texturas. O algoritmo MMP também pode explorar as características deste tipo de imagem transformada para codificar mais eficientemente. De fato, mesmo o MMP possuindo um carater universal, o conhecimento prévio da localização aproximada dos coeficientes mais importantes em uma imagem funciona como *informação lateral* que ajuda o MMP a comprimir mel-

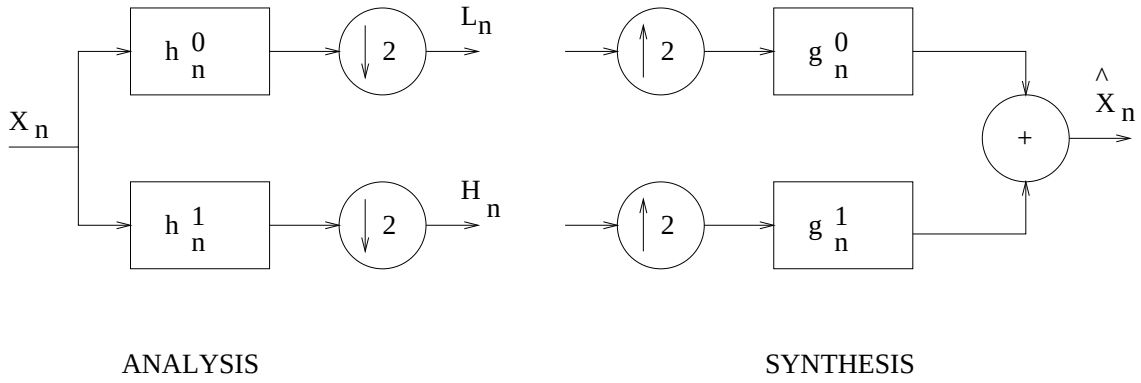


Figura 8.1: Banco de filtros de 2 canais.

hor. Outra vantagem é que ao usar a DWT elimina-se o efeito de blocagem, uma das distorções mais desagradáveis produzidas pelo MMP quando aplicado diretamente as imagens em taxas baixas.

8.1 A DWT

A DWT, descrita no apêndice B, pode ser implementada usando-se bancos de filtros iterados. O bloco fundamental desta construção é o banco de filtros de dois canais com reconstrução perfeita. Este banco de filtros está ilustrado na figura 8.1.

O banco de filtros de duas bandas é composto por um par de filtros de análise, com respostas ao impulso h_n^0 e h_n^1, e um par de filtros de síntese, com respostas ao impulso g_n^0 e g_n^1. Os filtros são tais que a saída de h_n^0 posui informação sobre as componentes de baixa frequência do sinal de entrada enquanto a saída de h_n^1 possui informação sobre as componentes de alta frequência. As saídas dos filtros de análise são subamostradas por um fator 2. Como os filtros não são ideais, os dois canais L_n e H_n possuirão componentes de "aliasing" [20]. Contudo, os quatro filtros do banco de filtros são projetados de modo a produzir o cancelamento do aliasing após a reconstrução.

O banco de dois canais pode ser usado de forma iterada, decompondo-se sucessivamente as saídas, para criar bancos com maior número de canais. A DWT pode ser obtida desta forma decompondo-se sucessivamente o canal de mais baixa frequência. A figura 8.2 ilustra a seção de análise de um banco para DWT de quatro canais.

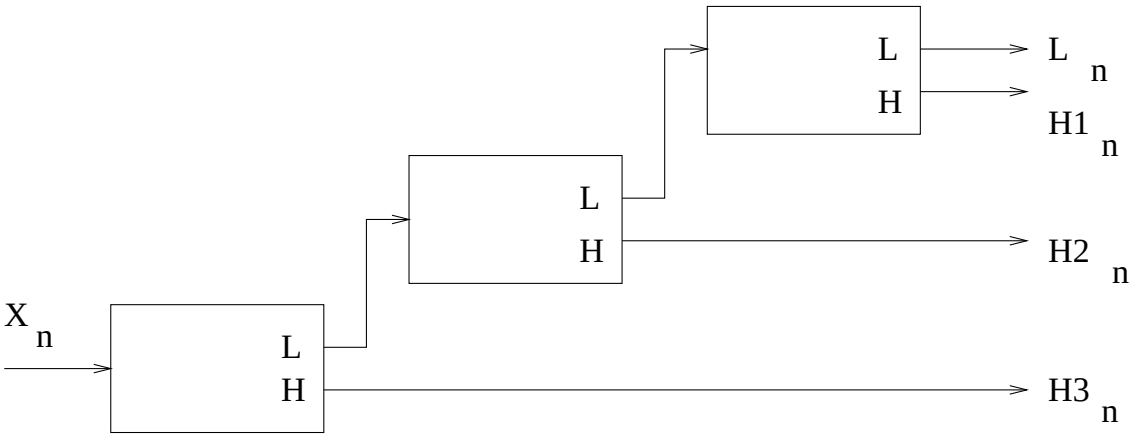


Figura 8.2: Banco de filtros iterado de 4 canais.

8.2 Resultados experimentais

O banco de filtros iterado da figura 8.2 representa uma decomposição para um sinal unidimensional. Para aplicação em imagens, usa-se a versão bidimensional desta decomposição. No apêndice H nós descrevemos em detalhes uma implementação de DWT bidimensional. Nós implementamos um banco de filtros bidimensional iterado de seis estágios usando o conjunto de filtros biortogonais de fase linear 9-7 descritos em [30].

Uma característica importante do sinal no domínio transformado é o aumento da faixa dinâmica. Por exemplo, em uma decomposição de 6 estágios, uma imagem com 256 níveis de cinza pode ter sua faixa diâmica aumentada para 8192 níveis. Nos capítulos anteriores, nós usamos um dicionário inicial de tamanho proporcional à faixa dinâmica. Se tentássemos isso agora, o dicionário inicial seria grande demais. Assim usamos um dicionário inicial que só contém o elemento zero. Por isso fomos obrigados a fazer uma pequena modificação no algoritmo MMP, para que ele pudesse aprender os outros níveis quando necessário. Esta modificação está descrita em detalhes no apêndice H.

A figura 8.3 mostra o desempenho com a imagem LENA. Nota-se que o uso da DWT melhorou bastante o desempenho do MMP, tornando-o bem mais próximo do desempenho do algoritmo SPIHT neste caso. O mesmo ocorreu com a imagem BARBARA. Para as demais imagens de teste do tipo tons de cinza, ocorreu uma melhoria de desempenho embora não tão grande. Contudo, como pode ser visto nas figuras 8.4 e 8.5, o desempenho com as duas imagens que contém texto piorou. Este

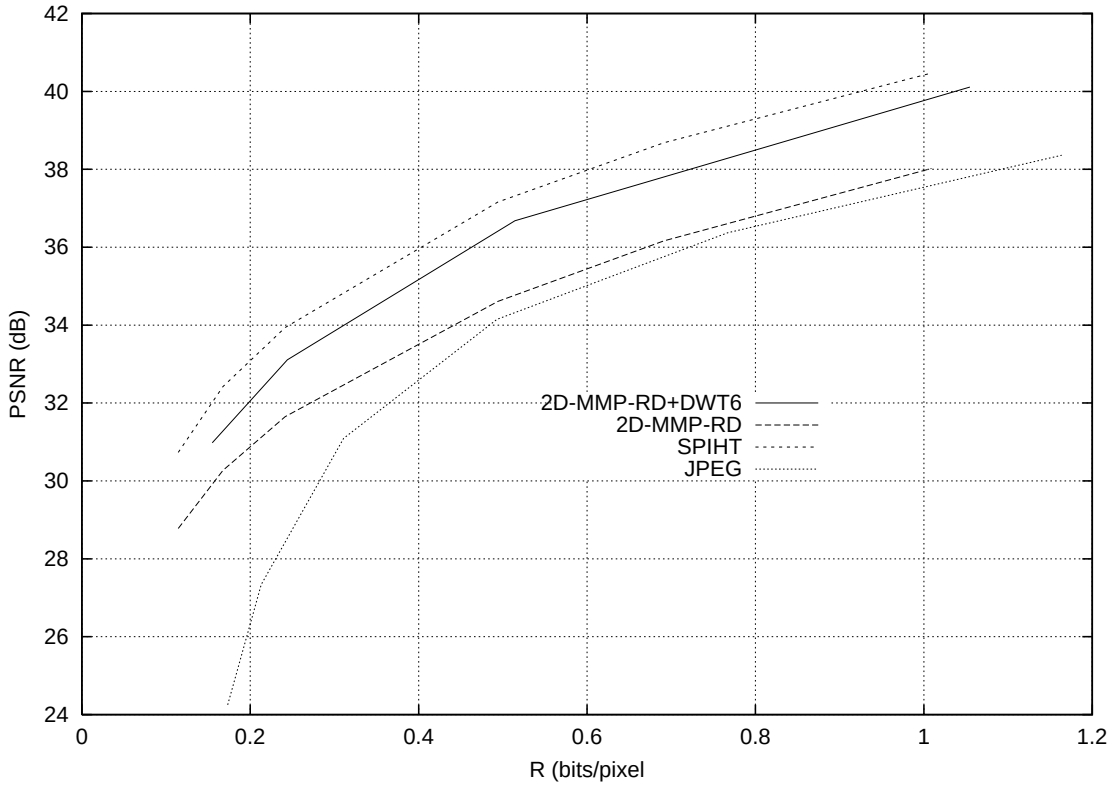


Figura 8.3: Desempenho com LENA 512×512.

comportamento é discutido em detalhes no apêndice H.

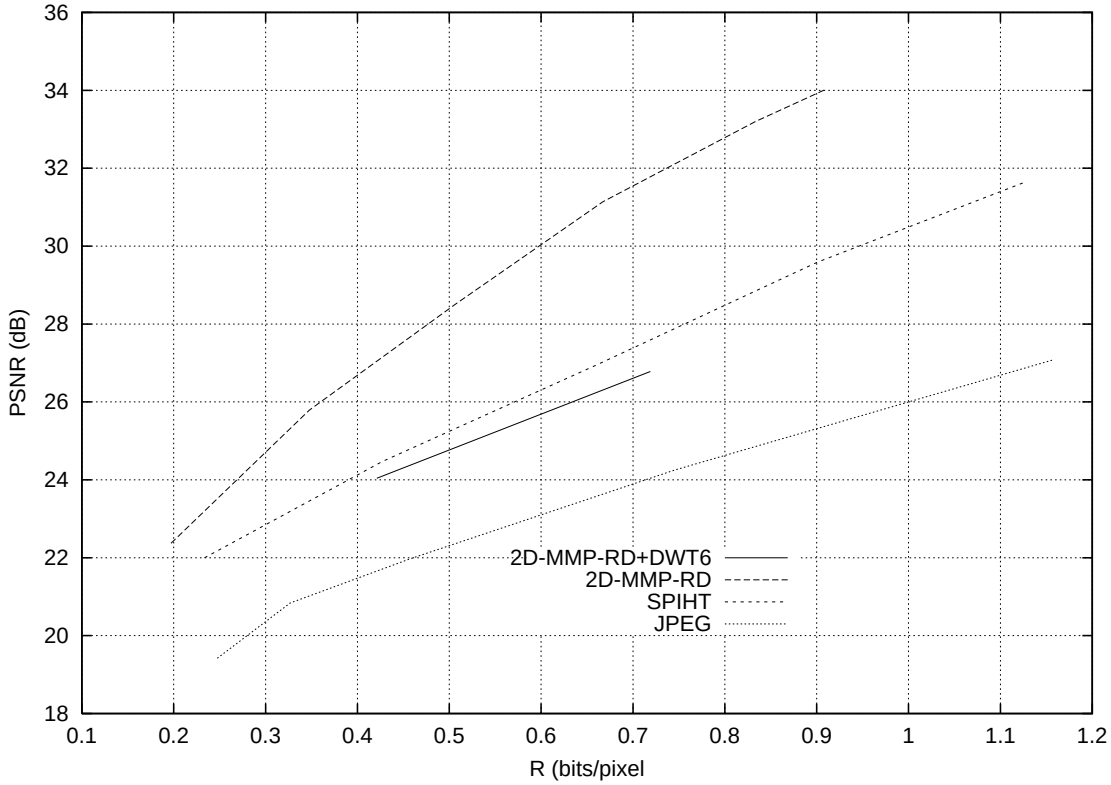


Figura 8.4: Desempenho com PP1205 512×512.

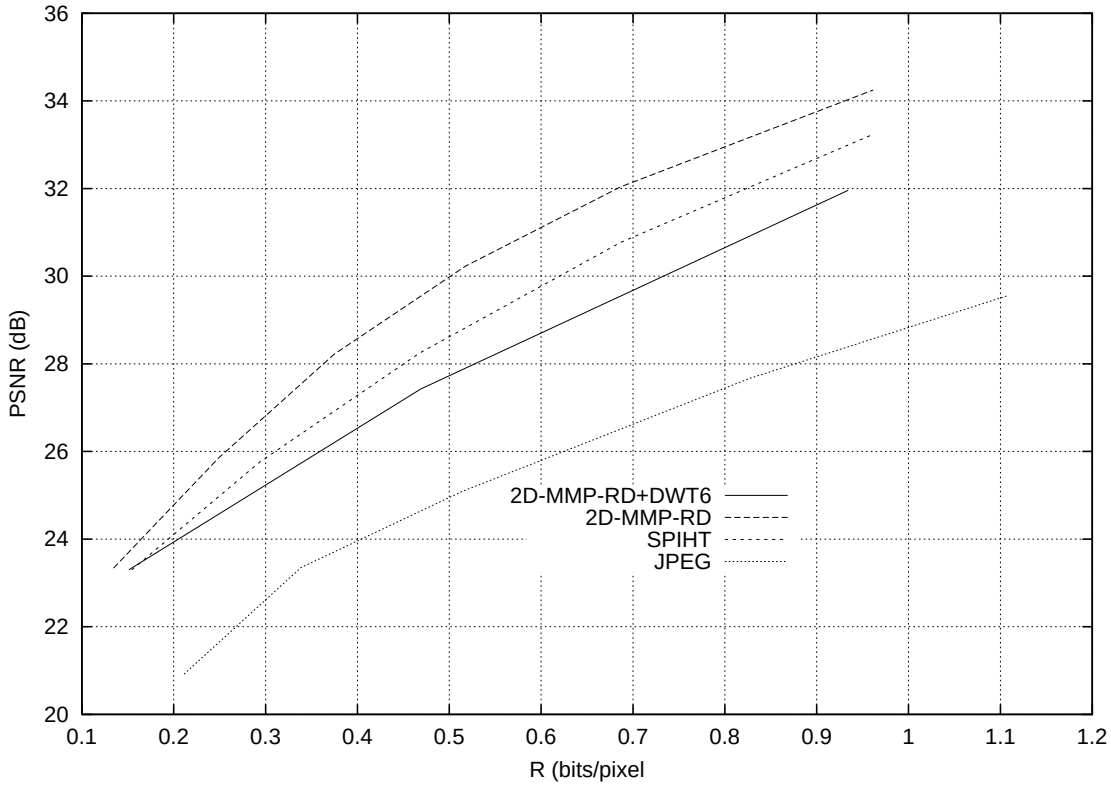


Figura 8.5: Desempenho com PP1209 512×512.

Capítulo 9

Conclusão

Nós apresentamos uma nova classe de algoritmos para compressão de dados com perdas baseados em *casamento aproximado de padrões recorrentes usando escalas*. Nós pudemos mostrar que, mesmo não sendo ótimo para comprimentos de bloco tendendo ao infinito, o desempenho do casamento com escalas pode superar o desempenho do casamento sem escalas para blocos finitos e taxas baixas.

Nós propusemos dois tipos de algoritmos: O MMP e o UMMP. Ambos fazem uma segmentação do vetor de entrada e procuram aproximar cada segmento usando vetores pertencentes a um dicionário. A diferença básica entre os dois é que o UMMP usa uma combinação linear de vetores do dicionário escalados para representar um segmento, enquanto o MMP usa apenas um vetor de cada vez. Em nossas simulações, o desempenho do MMP foi sempre superior mas nós suspeitamos que o desempenho do UMMP pode ser melhorado se considerarmos algumas de suas características. Por exemplo, Os vetores no dicionário, em escalas diferentes, tendem a ter potências diferentes, com a potência diminuindo a medida que a escala diminui. Isto ocorre porque as menores escalas correspondem a vetores resíduo que foram projetados mais vezes no dicionário e por isso tiveram mais energia subtraída. Assim, a função de escala deveria ser modificada para levar em conta que a potência média em escalas diferentes é diferente, fazendo uma correção de potência além da mudança de escala. O UMMP faz uma expansão do sinal de entrada em um frame. Sabe-se que a recontrução ótima deveria ser feita com o *frame dual* [17]. Portanto o decodificador do UMMP deveria ser modificado de modo a reconstruir usando o frame dual que é usualmente diferente daquele usado pelo codificador para analisar

o sinal.

Nós também desenvolvemos uma versão otimizada para o MMP , segundo um critério de taxa-distorção. Ela usa técnicas de programação dinâmica para otimizar a árvore de segmentação do MMP, para um dado vetor de entrada e um dado valor do parâmetro de controle λ. A versão otimizada do MMP, que nós chamamos de MMP-RD, mostrou uma melhoria de desempenho significativa quando comparada ao MMP sem otimização.

Como muitos codificadores em bloco, o MMP produz no sinal reconstruído uma distorção conhecida como blocagem. Em algumas aplicações, como por exemplo codificação de imagens, esta distorção é bastante incômoda. Nós desenvolvemos um método para redução de blocagem no MMP baseado em pós-filtragem adaptativa. A qualidade subjetiva das imagens reconstruídas aumentou nitidamente com o nosso método. O desempenho objetivo, medido pelas curvas de PSNR em função da taxa, também aumentou para todas as imagens de níveis de cinza e para todas as taxas testadas, quando usamos um filtro Gaussiano particular. Isto sugere que o desempenho do MMP para este tipo de fonte pode ser melhorado se nós adicionarmos vetores ao dicionário e que atendam a algum critério de suavidade. Para fazer isso, nós acreditamos ser necessário generalizar o codificador MMP de modo a operar com vetores com superposição.

Nós também empregamos o MMP para comprimir imagens no domínio da DWT. O algoritmo foi ligeiramente modificado de modo a aprender novos símbolos na escala 1×1 sempre que fosse necessário. Isso permitiu manter o dicionário inicial pequeno, apesar da faixa dinâmica muito elevada do sinal no domínio transformado.

Alguns temas interessantes para futuros trabalhos incluem:

1. Os decodificadores padrão do MMP e do UMMP não usam toda informação disponível para reconstruir o sinal. De fato, quando o codificador escolhe um índice do dicionário, ele implicitamente define uma série de restrições que posicionam o vetor de entrada de tamanho N dentro de uma região no espaço N-dimensional, ao invés de simplesmente especificar um ponto neste espaço. Esta informação pode possivelpente ser usada para fazer uma estimativa consistente do vetor de entrada original e que ao mesmo tempo atenda a algum critério de suavidade, por exemplo controlando o efeito de blocos.

2. O UMMP faz uma expansão em um frame. Sabe-se que o melhor frame para reconstrução é o frame dual, que normalmente é diferente do frame usado para analisar o sinal. O decodificador do UMMP poderia ser modificado de modo a executar *computações no frame inverso* conforme descrito em [28].

3. Generalização do MMP para usar vetores com superposição no codificador.

4. A regra de atualização do dicionário do MMP e do UMMP procura incluir no dicionário vetores que são semelhantes aos vetores de entrada. Esta regra poderia ser modificada de modo a construir um dicionário ao mesmo tempo bom para codificar o sinal de entrada mas possuindo alguma estrutura, permitindo assim a implementação de versões rápidas do MMP.

Apêndice A

Introduction

Today, there is a trend towards signal digitalization. Digital audio replaced previous analog formats in almost all applications. Digital video processing methods are used in DVD and HDTV. Decreasing VLSI chips costs and the development of high performance digital signal processing techniques suggest an even greater use of digitalization in the future. Therefore, it is important to develop efficient digital signal representations. Digital images are used in different fields such as medicine, geological prospection, entertainment and multimedia. Due to the large amount of data in a typical digital image, compression algorithms are of key importance in applications involving digital images. Although research in image compression has been done for over 30 years, it is still an open problem.

In this work, we propose a new class of compression algorithms based on *approximate matching of recurrent patterns using scales*, that is, the algorithms try to represent a block of input data using dilated or contracted versions of previously occurred blocks. Two major types of algorithms are studied: The MMP ("Multi-dimensional Multiscale Parser") and the UMMP (Universal Multiscale Matching Pursuits"). The MMP algorithm uses dilated and contracted versions of vectors in a dictionary to represent a block of input data. It builds the dictionary using concatenations of previously encoded blocks. The UMMP algorithm is similar to MMP but attempts to represent a block of input data using a *linear combination* of dilated or contracted versions of the vectors in the dictionary. Therefore UMMP performs a decomposition of the input data on a basis that is built while the data is encoded.

This work is organized as follows: the first part is written in portuguese and contains the introduction, the main text in chapters 2 through chapter 8, and a conclusion. The content of these chapters is reproduced, in fact with added detail, in 10 appendices.

Appendix B presents a formal definition of the compression problem, information content measures and some classical solutions for the compression problem.

Appendix C contains a study of the matching properties of Gaussian vectors that justify the construction of the UMMP and MMP algorithms, as well as form a basis for the performance analysis.

In appendices D and E we present detailed descriptions of the UMMP and MMP, and their variations.

In appendix F we introduce a rate-distortion optimized version of the MMP.

Appendix G discusses techniques to control the blocking effect when compressing image data using MMP.

In appendix H we show results of the MMP applied to images in the DWT (Discrete Wavelet transform) domain.

We conclude in appendix I where we also present some open problems and suggestions for further improvements.

Apêndice B

The compression problem

When we face the problem of digital signal compression, we are lead to question how much one can compress a particular signal. In other words, we want to know how many bits are necessary to describe a source of data within a given reproduction quality. This problem was first studied by Shannon, who created a new field of study called information theory [1, 2]. In the next section we show some basic results of rate-distortion theory [3], a branch of information theory that studies the compression of a source subject to a fidelity criterion.

B.1 Rate-distortion theory

The rate-distortion theory [3] studies the compression of a source subject to a fidelity criterion. In this theory a source is characterized by its statistical properties. Over the last 50 years, rate-distortion theory studies have focused to a large extent on the derivation of performance bounds for the trade-off between coding rate and achievable distortion for a given source. This trade-off is called $\mathsf{R}(\mathsf{D})$ function of the source, where R is the *average* rate required to describe any output produced by the source with *average* distortion at most D. One can show that the $\mathsf{R}(\mathsf{D})$ function of any source is a decreasing and convex function defined over the interval $[0, \mathsf{D}_{max}]$ [2]. D_{max} is the smallest average distortion that can be obtained using zero bits to describe the source (at zero rate we use always the same pattern to approximate the actual output produced by the source). In other words, if we want more quality we have to pay the cost of using more bits. The $\mathsf{R}(\mathsf{D})$ function defines the achievable

performance of any compression code, that is, the rate of any compression code must satisfy $R \geq R(D)$.

For example, a Gaussian source is a random variable (rv) [4] x with a Gaussian function as its probability density function. The $R(D)$ function for a Gaussian source, when the mean squared error is the fidelity criterion, is:

$$R(D) = \frac{1}{2} \log \left(\frac{\sigma^2}{D} \right) \tag{B.1}$$

If the logarithm in equation B.1 is base 2, then the rate $R(D)$ is given in bits.

A sequence of N independent Gaussian rvs form a *memoryless* vector Gaussian source $\mathbf{x} = (x_1, x_2, \ldots, x_N)$. Its $R(D)$ function is given in parametric form by:

$$\begin{aligned} R(\theta) &= \sum_{i=1}^{N} \frac{1}{2} \log \left(\frac{\sigma_i^2}{D_i(\theta)} \right) \\ D_i(\theta) &= \min[\theta, \sigma_i^2] \\ D(\theta) &= \sum_{i=1}^{N} D_i(\theta) \end{aligned} \tag{B.2}$$

where σ_i^2 are the variances of each rv.

An infinitely long sequence of independent Gaussian rvs form a *discrete-time* memoryless Gaussian source $\mathbf{x} = \{\ldots, x(-1), x(0), x(1), \ldots\}$. An infinite number of bits is required to specify any output of this source at any distortion level. Therefore we define a *per-sample* $R(D)$ as:

$$\begin{aligned} R(\theta) &= \lim_{N \to \infty} \frac{1}{N} \sum_{i=1}^{N} \frac{1}{2} \log \left(\frac{\sigma_i^2}{D_i(\theta)} \right) \\ D_i(\theta) &= \min[\theta, \sigma_i^2] \\ D(\theta) &= \lim_{N \to \infty} \frac{1}{N} \sum_{i=1}^{N} D_i(\theta) \end{aligned} \tag{B.3}$$

If all the rvs in a discrete-time memoryless Gaussian source have the same variance σ^2 then equation B.3 reduces to:

$$R(D) = \frac{1}{2} \log \left(\frac{\sigma^2}{D} \right) \tag{B.4}$$

We can conclude, comparing equations B.1 and B.4, that a discrete-time memoryless Gaussian source with iid (independent and identically distributed) rvs has a per-sample $R(D)$ numerically equal to The $R(D)$ function of any of its components.

N correlated Gaussian variables form a vector Gaussian source *with memory*. The $N \times N$ covariance matrix $\mathbf{K}_x$ is symmetric, so it can be diagonalized by a unitary transformation $\mathbf{A}$. This unitary transformation preserves the quadratic distortion measure and it is invertible so the $R(D)$ of a transformed vector is the same as the original one [2]. The transformed vector has uncorrelated Gaussian rvs and therefore independent components. So in the transformed domain, equation B.2 can be applied, we can set $\sigma_i^2 = \lambda_i$, where λ_i are the $\mathbf{K}_x$ eigenvalues.

$$\begin{aligned} R(\theta) &= \sum_{i=1}^{N} \frac{1}{2} \log\left(\frac{\lambda_i}{D_i(\theta)}\right) \\ \{\lambda_i\}_{i=1}^{N} &= \text{eigen}(\mathbf{K}_x) \\ D_i(\theta) &= \min[\theta, \lambda_i] \\ D(\theta) &= \sum_{i=1}^{N} D_i(\theta) \end{aligned} \tag{B.5}$$

An infinitely long sequence of correlated Gaussian rvs form a discrete-time Gaussian source *with memory* $\mathbf{x} = \{\ldots, x(-1), x(0), x(1), \ldots\}$. If the sequence has power spectral density $S(f)$, its per-sample $R(D)$ is given by:

$$\begin{aligned} R(\theta) &= \frac{1}{2} \int_{-\frac{1}{2}}^{\frac{1}{2}} \max\left[0, \log\left(\frac{S(f)}{\theta}\right)\right] df \\ D(\theta) &= \int_{-\frac{1}{2}}^{\frac{1}{2}} \min[\theta, S(f)] \end{aligned} \tag{B.6}$$

An important example of a discrete-time Gaussian source with memory is the Gauss-Markov source. This source has autocorrelation function ϕ_k and power spectral density $S(f)$ given by:

$$\begin{aligned} \phi_k &= \sigma^2 \rho^{|k|} \\ S(f) &= \frac{\sigma^2(1-\rho^2)}{\rho^2 - 2\rho\cos(2\pi f) + 1} \end{aligned} \tag{B.7}$$

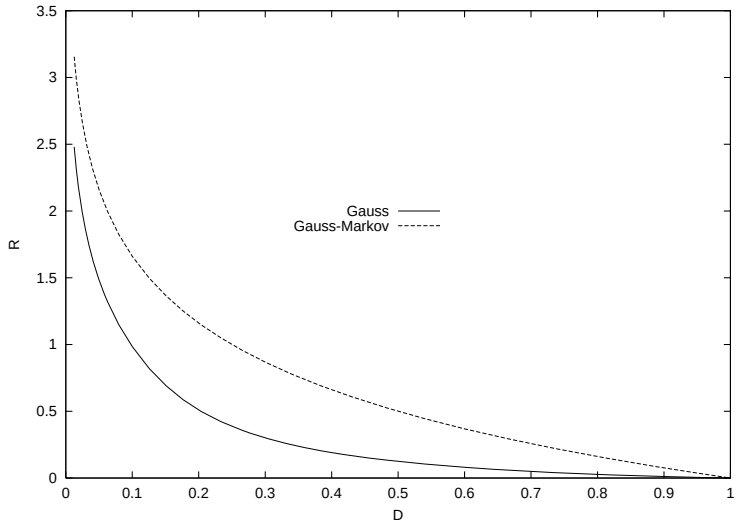


Figura B.1: $R(D)$ function for the Gaussian source ($\sigma^2 = 1$) and Gauss-Markov source ($\sigma^2 = 1, \rho = 0.78$).

Figure B.1 shows the $R(D)$ function for a memoryless Gaussian source with variance $\sigma^2 = 1$ and for a Gauss-Markov source with same variance and correlation coefficient $\rho = 0.78$. It is clear that $D_{max} = \sigma^2$ in these cases. It's also important to note that $R(D) \to \infty$ as $D \to 0$. This happens because every possible output of both sources is a real number requiring infinite precision to be described with zero distortion. We can also see in the figure that the source with memory needs a smaller rate R at the same distortion D (unless when $R = 0$) and is therefore easier to compress.

If the source is not Gaussian we can't usually find its $R(D)$ function in closed form. For the memoryless case, the $R(D)$ function can be numerically evaluated by the Blahut-Arimoto algorithm [2]. For non-Gaussian sources with memory, the method used in the Gaussian case is useless, because after the covariance matrix diagonalization the transformed source won't be memoryless (decorrelation implies independence only in the Gaussian case). As an approximate solution we can group the source in blocks of N symbols and apply the Blahut-Arimoto algorithm to the blocks considered as single entities. The block source will be approximately memoryless if the block length N is large enough.

Even when we can find the $R(D)$ function for a given source, we may be unable to find a coding method to compress that source using exactly $R(D)$ bits at distortion D. In fact, the proofs of the coding theorems involving the $R(D)$ function

are not constructive. They don't give us any hint of how to compress the source at the rates predicted by its $R(D)$ function. However, for $D = 0$ and *discrete sources*, there are compression methods that achieve $R(0)$, at least asymptotically when the sequence of source symbols to compress becomes infinitely long.

A discrete source x produces output symbols that belong to a finite alphabet $\mathcal{A} = \{a_i\}_{i=0}^{M-1}$. It can be modeled as a discrete random variable. The minimum average rate required to represent with zero distortion any source symbol X is called the *entropy* of the source $H(x) = R(0)$. The entropy of a discrete source x, with symbols in an alphabet of size $|\mathcal{A}| = M$ and probability density function $P_r\{x = a_i\} = p_i$ is given by [2]:

$$H(x) = -\sum_{i=0}^{M-1} p_i \log(p_i) \tag{B.8}$$

A discrete memoryless vector source $\mathbf{x} = (x_0 \dots x_{N-1})$ can be modeled as a vector of independent discrete random variables. Its entropy is the sum of each component's entropy, $H(\mathbf{x}) = \sum_{i=0}^{N-1} H(x_i)$.

A discrete memoryless source can be modeled as an infinitely long sequence of discrete random variables $\mathbf{x} = \{\dots, x(-1), x(0), x(1), \dots\}$. The minimum average per-symbol rate R required to represent this source is called the *entropy rate* and is given by:

$$R = \lim_{N\to\infty} \frac{1}{N} \sum_{m=-N/2}^{N/2-1} H(x(m)) \tag{B.9}$$

A discrete vector source with memory is a random vector $\mathbf{x}$ with correlated discrete components. Its entropy is given by:

$$H(\mathbf{x}) = -\sum_{i_1=0}^{M-1} \sum_{i_2=o}^{M-1} \dots \sum_{i_M=0}^{M-1} p_{i_1,i_2,\dots,i_M} \log(p_{i_1,i_2,\dots,i_M}) \tag{B.10}$$

where $p_{i_1,i_2,\dots,i_M}$ is the joint probability density function of $\mathbf{x}$. A discrete source with memory is an infinitely long sequence of correlated discrete random variables. Its entropy rate is given by the limit:

$$\begin{aligned} R &= \lim_{N\to\infty} \frac{1}{N} H(\mathbf{x}^{(N)}) \\ \mathbf{x}^{(N)} &= (x(-N/2)x(-N/2+1)\dots x(N/2-1)) \end{aligned} \tag{B.11}$$

B.2 Classical solutions to the compression problem

There are several methods to compress a discrete source with zero distortion at rates arbitrarily close to the source's entropy. A compression with zero distortion is called *lossless*. Huffman coding, arithmetic coding and Lempel-Ziv [2, 5] coding are examples of such lossless methods. The rate of these codes approaches the source entropy, at least when the number of source symbols processed is large. Huffman and arithmetic coders require the knowledge of a statistic model for the source in order to be implemented. This means that they are matched to a particular source and only achieve optimum performance for that source. Adaptive versions of them start with an initial model that is updated as the encoding proceeds. The Lempel-Ziv algorithm on the other hand is *universal* in the sense that no source model is required and the code rate equals the source's entropy rate when the number of source symbols tends to infinity.

B.2.1 Lossy compression: The two-step approach

A compression scheme with distortion different from zero is called *lossy*. Unlike the lossless compression case, a universal method for lossy compression whose performance achieves $\mathsf{R}(\mathsf{D})$ for any source is unknown. A widely used solution to the problem of lossy compression is a *two-step* approach: In the first step, we create a version $\hat{\mathsf{x}}$ of the source x with smaller entropy, $\mathsf{H}(\hat{\mathsf{x}}) < \mathsf{H}(\mathsf{x})$. This step is called quantization and is done by a non-invertible mapping $\hat{\mathsf{x}} = \mathsf{Q}(\mathsf{x})$ (If the mapping was invertible, then the entropy $\mathsf{H}(\hat{\mathsf{x}})$ would necessarily equal $\mathsf{H}(\mathsf{x})$). In the second step, the entropy coding step, we apply any lossless compression method to $\hat{\mathsf{x}}$.

A simple and widely studied kind of quantization method is the *scalar quantizer*. A scalar quantizer encodes the source in a sample-by-sample basis. It divides the source output space of each component in a finite number of intervals and associates to each interval a reproduction value. Whenever a sample is output by the source and falls in one of the intervals, the quantizer uses the associated reproduction value as the sample representation. It encodes one symbol at a time by assigning one reproduction value to each input sample. A *vector quantizer* on the

other hand encodes blocks of N samples, or vectors, at once. It divides the N dimensional space into regions and associates one reproduction vector to each region. The theory of scalar and vector quantization is well developed an tells us how to chose the intervals/regions and reproduction values/vectors to maximize different performance criteria. For example there are the Loyd-Max quantizers [6] optimized for minimum distortion when used with a given source. If we are willing to skip the entropy coding step in order to reduce the implementation complexity, this can be the best approach. However, if we wish a performance closer to $R(D)$, we can use the entropy-constrained quantizers [7], optimized to minimize distortion while keeping the output entropy fixed. Vector quantizers get performance closer (although not equal) to $R(D)$ than scalar ones, and performance usually increases with the number of dimensions.

The design of the quantizers rely on optimization techniques and are strongly dependent on the source model's choice. If the source statistics varies widely or is unknown, some kind of adaptive quantization must be used. An interesting approach to adaptive quantization is the Lossy-Lempel-Ziv (LLZ) algorithm [8]. It merges the two steps, quantization and entropy coding in a single one, and it's fully adaptive since it builds iteratively its own encoding rules while processing the data from the source. The LLZ algorithm compresses data form a source $x(n)$ with distortion at most d^*, a target distortion parameter. It is equivalent to the lossless Lempel-Ziv algorithm when the target distortion is zero, so it is universal and asymptotically optimal for lossless compression. However, for target distortions greater than zero, the LLZ algorithm's performance is bounded away from $R(D)$ [9]. In fact, LLZ performance gets worse as the target distortion increases.

B.2.2 Lossy Compression: The three-step approach

Sometimes, a simpler statistical source model can be found if we perform a transformation on the data before we apply a compression scheme. This usually leads to better performance even if the compression scheme is adaptive since simpler statistics tend to be learned faster. For example, in image compression applications, almost all high performance coders employ such transformation step prior to compression. This is called the *three-step approach*: The first step is a *transformation*

step where data from the source is transformed to another domain; for example the DCT (discrete cosine transform) in the JPEG coder [10] and the DWT (discrete wavelet transform) in EZW and SPIHT coders [12, 13]. This transformations allow the use of simple although useful assumptions on the source model in the transformed domain. The second step is the *quantization step* where the entropy of the transformed source data is reduced by quantizers optimized for the transformed source. The third step is the *entropy coding* step where redundancy on the quantized data is reduced by lossless compression algorithms. In the three step framework, rate-distortion theory concepts are applied to optimize performance in a $\mathsf{R}(\mathsf{D})$ sense. For instance, The JPEG coder uses a two-dimensional 8×8 DCT transform followed by scalar quantization of the transform coefficients. The DCT approximately diagonalizes first order Markov types of memory, which turn out to be a fairly good model to images. This property improves the efficiency of the simple scalar quantization used in the second step. Each coefficient is quantized by a different scalar quantizer, to explore their different statistical behavior and run-length encoding (RLE) is used to explore the high probability of zero runs after the quantization step. In addition to RLE, Huffman encoding is used in the entropy coding step. Although not specified in JPEG standard, further $\mathsf{R}(\mathsf{D})$ optimizations are possible within JPEG syntax. For example, sometimes a DCT coefficient will not be quantized as zero, breaking what would be a long sequence of zeros and so increasing the coding rate. However, the increase in quality due to that non-zero coefficient could be sometimes very small. So, a $\mathsf{R}(\mathsf{D})$ enhanced encoder could force this coefficient to be zero improving performance, while still standing fully compliant to JPEG standard [16]. Other optimizations are possible as the JPEG standard allows the specification of custom Huffman and quantization tables. In other words, JPEG's second and third step can be adapted to source statistics to a certain degree. Although not possible with JPEG, perhaps an adaptive transformation at the first step would be worth it. In fact, coders based on DWT usually outperform those based on DCT. So we are lead to search for adaptive transformations which can be effectively implemented in the first step.

In this work, we develop a compression algorithm that merges the transformation, quantization and entropy coding steps in one. It also requires no prior

knowledge of the source statistics as it is fully adaptive, that is, it adapts the transformation, the quantization and the entropy coding to the statistics of the source while compressing it. In the next section, we study some transformations used in popular image coding methods.

B.3 Image transforms and Matching Pursuits

The term transform is usually associated to the expansion of a signal in a *basis*. A basis for an N−dimensional Hilbert space $\mathcal{H}$ is a set $\mathcal{B} = \{\mathbf{b}_k\}_{k=0}^{N-1}$ of N *linearly independent* vectors whose *closed linear span* $\mathcal{V}$ equals the space $\mathcal{H}$ [17]. In other words, any $N \times 1$ vector $\mathbf{x} = (x_0 x_1 \dots x_{N-1})^T \in \mathcal{H}$ can be represented as a linear combination of the N basis vectors $\mathbf{b}_k$ as:

$$\mathbf{x} = \sum_{k=0}^{N-1} X_k \mathbf{b}_k \tag{B.12}$$

The coefficients X_k are the components of the $N \times 1$ $\mathbf{X}$ vector, a transform domain version of $\mathbf{x}$. In matrix form, equation B.12 is:

$$\begin{aligned} \mathbf{x} &= \mathbf{B}\mathbf{X} \\ \mathbf{B} &= \begin{pmatrix} \mathbf{b}_0 & \mathbf{b}_1 & \dots & \mathbf{b}_{N-1} \end{pmatrix} \end{aligned} \tag{B.13}$$

The transformed vector $\mathbf{X}$ can be expressed as:

$$\begin{aligned} \mathbf{X} &= \mathbf{A}\mathbf{x}, \\ \mathbf{I} &= \mathbf{B}\mathbf{A} \end{aligned} \tag{B.14}$$

Just as a vector can be expanded in a series of basis vectors, an $N \times N$ array $\mathbf{x}$ like an image can be represented by a linear combination of N^2 *basis images*.

$$\mathbf{x} = \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} X_{k,l} \mathbf{b}_{k,l} \tag{B.15}$$

The basis images $\mathbf{b}_{k,l}$ in equation B.15 are $N \times N$ arrays, and the coefficients $X_{k,l}$ form a transform domain representation of the $\mathbf{x}$ image. To reduce the complexity, we can use a restricted class of two-dimensional transforms, the *separable transforms*. To be separable, a transform must have basis images like:

$$\mathbf{b}_{k,l} = \mathbf{b}'_k \mathbf{b}''^{T}_l \tag{B.16}$$

where $\mathbf{b}'_k$ and $\mathbf{b}''_l$ are $N \times 1$ vectors.

In matrix form, a separable transform becomes:

$$\begin{aligned} \mathbf{x} &= \mathbf{B}'\mathbf{X}\mathbf{B}''^{T} \\ \mathbf{B}' &= \begin{pmatrix} \mathbf{b}'_0 & \mathbf{b}'_1 & \dots & \mathbf{b}'_{N-1} \end{pmatrix} \\ \mathbf{B}'' &= \begin{pmatrix} \mathbf{b}''_0 & \mathbf{b}''_1 & \dots & \mathbf{b}''_{N-1} \end{pmatrix} \end{aligned} \tag{B.17}$$

The transformed domain image $\mathbf{X}$ can be expressed as:

$$\begin{aligned} \mathbf{X} &= \mathbf{A}'\mathbf{x}\mathbf{A}''^{T} \\ \mathbf{I} &= \mathbf{B}'\mathbf{A}' = \mathbf{B}''\mathbf{A}'' \end{aligned} \tag{B.18}$$

where 'T' denotes transposition.

B.3.1 The DCT

One popular transform for image compression is the DCT, used in JPEG. The DCT ("discrete cosine transform") of a $N \times 1$ vector $\mathbf{x} = (x_0 x_1 \dots x_{N-1})^T$ is defined as:

$$\begin{aligned} \mathbf{X} &= \mathbf{C}_N \mathbf{x}, \\ \mathbf{C}_N &= \{c(k,n)\}_{k,n=0}^{k,n=N-1} \\ c(k,n) &= \begin{cases} \dfrac{1}{\sqrt{N}}, & k = 0, 0 \le n < N \\ \sqrt{\dfrac{2}{N}} \cos\left(\dfrac{\pi(2n+1)k}{2N}\right), & 1 \le k < N, 0 \le n < N \end{cases} \end{aligned} \tag{B.19}$$

The $N \times N$ matrix $\mathbf{C}_N$ is the cosine transform matrix. It is unitary and real, that is, $\mathbf{C}_N^{-1} = \mathbf{C}_N^T$. Therefore, a vector $\mathbf{x}$ can be recovered from its transformed version $\mathbf{X}$ as:

$$\mathbf{x} = \mathbf{C}_N^T \mathbf{X} \tag{B.20}$$

The two-dimensional DCT of a $N \times N$ array $\mathbf{x}$ is a separable transform given by:

$$\mathbf{X} = \mathbf{C}_N \mathbf{x} \mathbf{C}_N^T \tag{B.21}$$

If $\mathbf{x}$ is a first order Markov vector, the coefficients of the transformed vector $\mathbf{X}$ are almost uncorrelated, allowing the use of simple scalar quantization of the coefficients. The first order Markov model is usually good for small blocks of a typical image, but not for the whole image. So in an image compression application, the image is first divided in blocks (8×8 for the JPEG) and the two-dimensional DCT is applied independently to each block. For example, for an $MN \times MN$ image, a block-DCT applied to $N \times N$ blocks would be:

$$\begin{aligned} \mathbf{X} &= \mathbf{C}^b_{MN} \mathbf{x} (\mathbf{C}^b_{MN})^T \\ \mathbf{C}^b_{MN} &= \begin{pmatrix} \mathbf{C}_N & 0 & \dots & 0 \\ 0 & \mathbf{C}_N & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \mathbf{C}_N \end{pmatrix} = \mathbf{I}_M \otimes \mathbf{C}_N \end{aligned} \tag{B.22}$$

When a block-DCT-transformed image is compressed by quantization of its coefficients, the continuity at the boundaries of the blocks is broken. This creates visible artifacts known as the *blocking effect*. At high compression rates this artifacts become visible and highly annoying as they are visually enhanced by the Mach band effect [18]. One way to preserve the continuity of the boundaries is by use of overlapping blocks. The LOT (Lapped Orthogonal Transform) [19] uses $N \times 2N$ matrixes instead of the $N \times N$ $\mathbf{C}_N$ matrixes in equation B.22 to dramatically reduce the blocking effect. In other words, it uses N^2 basis images of size $2N \times 2N$ to represent a $N \times N$ block. Therefore the LOT is likely to reduce the blocking effect without an increase in bit-rate.

The DCT basis vectors are sampled cosine functions. Therefore their Fourier spectrum is highly concentrated. Also they are uniformly distributed in space, as their energy is not concentrated at any particular point. If we try to expand a vector well localized in space, like an impulse, we have to use many coefficients in the expansion, because the information we need is spread across the whole basis.

B.3.2 The DWT

Another popular transform for image compression is the *Discrete Wavelet Transform* (DWT), used in the EZW and SPHIT algorithms.

The continuous time DWT, uses expansions and contractions of a mother function $\psi(t)$ as basis vectors, that is,

$$\begin{aligned} x(t) &= \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} c_{m,n} \psi_{m,n}(t) \\ \psi_{m.n}(t) &= 2^{-m/2} \psi(2^{-m} t - n) \end{aligned} \tag{B.23}$$

The Fourier spectrum of the wavelet decomposition is:

$$\begin{aligned} X(\omega) &= \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} c_{m,n} \Psi_{m,n}(\omega) \\ \Psi_{m.n}(\omega) &= 2^{m/2} e^{-jn\omega 2^m} \Psi(2^m \omega) \\ \Psi(\omega) &= \int_{-\infty}^{\infty} \psi(t) e^{-j\omega t} dt \end{aligned} \tag{B.24}$$

Equations B.23 and B.24 show that as m increases the basis vectors become less localized in time and more localized in frequency. It can be shown that $\Psi(\omega)$ must be band-pass or equation B.23 will not hold. In this case we get an *octave-band* decomposition, where the central frequency of the m^{th} basis vector spectrum is twice that of the $(m+1)^{th}$ basis vector spectrum. Also the bandwidth of the m^{th} basis vector is half that of the $(m+1)^{th}$ basis vector so we have a *constant Q* decomposition.

In the discrete-time case, if the signal has finite duration we don't need an infinite size basis as in equation B.23. There is a minimum value for m that corresponds to the maximum resolution of the discrete-time signal, and therefore we don't need to go below that. However there is not such a maximum value for m. Thus we choose a value of m where we stop the decomposition, and add an extra vector $\phi(t)$ to the base to make it complete. The DWT is usually implemented by perfect reconstruction filter banks [20]. In these implementations, the dimension of the basis vectors are related to each filter's length and the number of basis vectors to the number of channels in the filter bank. In a typical image coding application

there is no blocking effect because the number of basis vectors is much smaller than the dimension of each vector and so overlapping comes in a natural way.

Many DWT based coders outperform DCT based compression methods for typical images. In fact, the better spatially localized basis vectors allow the use of fewer basis vectors, and therefore fewer bits, to represent some sources up to a given level of accuracy. However, for signals well localized in frequency, DCT can perform better.

B.3.3 The matching pursuits

It would be interesting if we could have vectors well localized in time and in frequency in the same basis. One way to achieve that is to drop the linear independence constraint an move to *overcomplete* expansions using frames instead of bases.

A set of vectors $\mathcal{D} = \{\phi_k\}_{k=0}^{K-1}$ in a Hilbert space $\mathcal{H}$ is called a *frame* [17] if there are two constants $A > 0$ and $B < \infty$, such that for all $\mathbf{x} \in \mathcal{H}$:

$$A||\mathbf{x}||^2 \leq \sum_k | < \phi_k, \mathbf{x} > |^2 \leq B||\mathbf{x}||^2 \tag{B.25}$$

where $< ., . >$ denotes inner product.

The constants A and B are the *frame bounds* and when $A = B$ the frame is *tight*. For a tight frame we have:

$$\begin{aligned} ||\mathbf{x}||^2 &= A^{-1} \sum_{k=0}^{K-1} | < \phi_k, \mathbf{x} > |^2 \text{and} \\ \mathbf{x} &= A^{-1} \sum_{k=0}^{K-1} < \phi_k, \mathbf{x} > \phi_k \end{aligned} \tag{B.26}$$

If $||\phi_k|| = 1$ for all k, then the constant A gives the redundancy ratio of the frame. For example if $A = 2$ there are twice as many vectors as needed to span the space $\mathcal{H}$. The expansion in equation B.26 is not unique because the vectors in the frame are linearly dependent. In fact, the linear dependence means that $\sum_{k=0}^{K-1} \alpha_k \phi_k = 0$ has a nontrivial solution (some $\alpha_k \neq 0$) so $\mathbf{x} = \sum_{k=0}^{K-1} (A^{-1} < \phi_k, \mathbf{x} > + \alpha_k)\phi_k$ is also a valid expansion.

We can use a subset of $N < K$ vectors from $\mathcal{D}$ to approximate $\mathbf{x}$ as:

$$\hat{\mathbf{x}} = \sum_{i=0}^{N-1} \alpha_i \phi_{k_i} \tag{B.27}$$

We would like to minimize the error norm $||\mathbf{x} - \hat{\mathbf{x}}||^2$ in the approximation of equation B.27 by choosing the N best vectors of $\mathcal{D}$ to represent $\mathbf{x}$. Instead of trying to find at once those N best vectors (it is an NP hard problem [21]) we can find a sub-optimal-N solution by a greedy algorithm called *matching pursuits* [22]. The matching pursuits algorithm attempts to optimize the choice of vectors from $\mathcal{D}$ one at a time.

The decomposition begins by choosing the k_0 value that maximizes the inner product
$< \phi_{k_0}, \mathbf{x} >$. Then a *residue* $R^1\mathbf{x}$ is computed as:

$$\begin{aligned} R^0\mathbf{x} &= \mathbf{x} \\ R^1\mathbf{x} &= R^0\mathbf{x} - < \phi_{k_0}, \mathbf{x} > \phi_{k_0} \end{aligned} \tag{B.28}$$

This residue $R^1\mathbf{x}$ is then expanded in the same way as $\mathbf{x}$. At step n we have:

$$R^{n+1}\mathbf{x} = R^n\mathbf{x} - < \phi_{k_n}, \mathbf{x} > \phi_{k_n} \tag{B.29}$$

After N steps, the vector $\mathbf{x}$ can be approximated like in equation B.27 as:

$$\hat{\mathbf{x}} = \sum_{i=0}^{N-1} < \phi_{k_i}, \mathbf{x} > \phi_{k_i} \tag{B.30}$$

This procedure is quite general and convergence is guaranteed for any arbitrary frame as $N \to \infty$. However, if the frame is an orthonormal basis ($A = 1$ and $||\phi_k|| = 1$), then the matching pursuits algorithm finds the optimal solution in N steps. MP has some useful properties. For example it has the *energy compaction property* since the frame elements which are closer to the vector $\mathbf{x}$ are chosen first. Therefore the vector and the coefficients used in the expansion are naturally ordered by importance. In [22] a frame of *gabor functions* (time-scaled, shifted and modulated Gaussian functions) was used with matching pursuits defining an adaptive time-frequency transform with good resolution both in time and frequency.

In compression applications the coefficients given by the inner products in equation B.30 are quantized and the matching pursuits algorithm performs like a *multistage gain-shape vector quantizer* [23].

In appendix D we describe The UMMP algorithm. It is based on matching pursuits, and creates its own frame while expanding the data. It also uses string matching to achieve entropy coding, merging transformation, quantization and entropy coding in a single step.

Apêndice C

Matching probability of Gaussian vectors subject to a fidelity criterion

In this appendix we investigate the matching properties of Gaussian vectors. Our goal is to get some insight on the workings of approximate pattern matching with scales, justifying its use in a low-rate lossy compression scheme.

C.1 Matching with a dictionary of Gaussian vectors

Let $\mathbf{x} = (x_1 x_2 \dots x_N)^T$ be a zero mean Gaussian vector with covariance matrix $\mathbf{K}_x$ and let $\mathcal{D} = \{\mathbf{y}^1, \mathbf{y}^2, \dots, \mathbf{y}^M\}$ be a dictionary where $\mathbf{y}^i = (y_1^i y_2^i \dots y_N^i)^T$ are zero mean Gaussian vectors with covariance matrix $\mathbf{K}_{y^i} = \mathbf{K}_x$. Let $\mathbf{x}$ and $\mathbf{y}^i, i \in 1, M$ be mutually independent [4].

We say a match occurs when at least one of the dictionary vectors can be used to replace $\mathbf{x}$ with distortion at most Nd^*. If the distortion measure is the mean squared error, then we have a match if:

$$\|\mathbf{x} - \mathbf{y}^i\|^2 \leq Nd^* \text{ for some } i \in [1, M] \tag{C.1}$$

The probability of match P_m is then:

$$P_m = Pr\{||\mathbf{x} - \mathbf{y}^i||^2 \leq Nd^* \text{ for some } i \in [1, M]\} \tag{C.2}$$

or

$$P_m = 1 - Pr\{||\mathbf{x} - \mathbf{y}^i||^2 > Nd^* \text{ for all } i \in [1, M]\} \tag{C.3}$$

Using all the M vectors of the dictionary we form a vector $\mathbf{u}$ defined by:

$$\mathbf{u} = ((\mathbf{y}^1)^T(\mathbf{y}^2)^T \dots (\mathbf{y}^M)^T)^T \tag{C.4}$$

The dictionary vectors are mutually independent zero mean Gaussian vectors with covariance matrices $\mathbf{K}_x$, so the vector $\mathbf{u}$ is zero mean Gaussian with covariance matrix

$$\mathbf{K}_u = \mathbf{I}_M \otimes \mathbf{K}_x \tag{C.5}$$

where $\mathbf{I}_M$ is a $M \times M$ identity matrix and $\otimes$ denotes a Kronecker product.

For a particular realization $\mathbf{X}$ of the random vector $\mathbf{x}$ we can define the difference vector $\mathbf{r}^i$ as:

$$\mathbf{r}^i = \mathbf{y}^i - \mathbf{X}, i \in [1, M] \tag{C.6}$$

And the difference vector vector $\mathbf{v}$ as:

$$\mathbf{v} = ((\mathbf{r}^1)^T(\mathbf{r}^2)^T \dots (\mathbf{r}^M)^T)^T \tag{C.7}$$

or using equations C.4 and C.6:

$$\mathbf{v} = \mathbf{u} - \mathbf{1}_M \otimes \mathbf{X} \tag{C.8}$$

where $\mathbf{1}_M$ is the $M \times 1$ all-ones vector $(11 \dots 1)^T$

The $\mathbf{v}$ vector's mean μ_v is:

$$\mu_v = E\{\mathbf{v}\} = E\{\mathbf{u} - \mathbf{1}_M \otimes \mathbf{X}\} = -\mathbf{1}_M \otimes \mathbf{X} \tag{C.9}$$

and it's covariance is:

$$\begin{aligned}\mathbf{K}_\nu &= E\{(\mathbf{v}-E\{\mathbf{v}\})(\mathbf{v}-E\{\mathbf{v}\})^T\}\\ &= E\{(\mathbf{u}-\mathbf{1}_M\otimes\mathbf{X}+\mathbf{1}_M\otimes\mathbf{X})(\mathbf{u}-\mathbf{1}_M\otimes\mathbf{X}+\mathbf{1}_M\otimes\mathbf{X})^T\}\\ &= E\{\mathbf{u}\mathbf{u}^T\}=\mathbf{K}_u=\mathbf{I}_M\otimes\mathbf{K}_x \qquad (C.10)\end{aligned}$$

Therefore, $\mathbf{v}$ has a conditional p.d.f (probability density function) [4] of the form:

$$p_{\nu|X}(\mathbf{V})=\frac{1}{(2\pi)^{\frac{MN}{2}}|\mathbf{K}_\nu|^{\frac{1}{2}}}e^{-\frac{1}{2}(\mathbf{V}-\mu_\nu)^T\mathbf{K}_\nu^{-1}(\mathbf{V}-\mu_\nu)} \qquad (C.11)$$

where μ_ν is a function of $\mathbf{X}$ as given by equation C.9.

The difference vectors $\mathbf{r}^i$ defined in equation C.6 have norm given by:

$$||\mathbf{r}^i||^2=(\mathbf{r}^i)^T\mathbf{r}^i=\mathbf{v}^T(\phi_M^i\otimes\mathbf{I}_N)\mathbf{v} \qquad (C.12)$$

where ϕ_M^i is the $M\times M$ *indicator matrix* where all elements are zero except the i-row i-column element which is one.

The difference norm vector $\mathbf{z}$ is defined as:

$$\mathbf{z}=(||\mathbf{r}^1||^2||\mathbf{r}^2||^2\ldots||\mathbf{r}^M||^2)^T \qquad (C.13)$$

and it's characteristic function [4] is:

$$\begin{aligned}M_z(\mathbf{s})=E\{e^{-\mathbf{s}^T\mathbf{z}}\}&=E\{e^{-\sum_{i=1}^{M}s_i\mathbf{v}^T(\phi_M^i\otimes\mathbf{I}_N)\mathbf{v}}\}\\ &=\int_{-\infty}^{\infty}\ldots\int_{-\infty}^{\infty}e^{-\sum_{i=1}^{M}s_i\mathbf{V}^T(\phi_M^i\otimes\mathbf{I}_N)\mathbf{V}}p_{\nu|X}(\mathbf{V})d\mathbf{V}\\ &=\int_{-\infty}^{\infty}\ldots\int_{-\infty}^{\infty}e^{-\sum_{i=1}^{M}s_i\mathbf{V}^T(\phi_M^i\otimes\mathbf{I}_N)\mathbf{V}}\frac{1}{(2\pi)^{\frac{MN}{2}}|\mathbf{K}_\nu|^{\frac{1}{2}}}e^{-\frac{1}{2}(\mathbf{V}-\mu_\nu)^T\mathbf{K}_\nu^{-1}(\mathbf{V}-\mu_\nu)}d\mathbf{V}\\ &=\left(\frac{|\mathbf{K}'_\nu|}{|\mathbf{K}_\nu|}\right)^{\frac{1}{2}}e^{-\frac{1}{2}\mu_\nu^T\mathbf{D}\mu_\nu}\int_{-\infty}^{\infty}\ldots\int_{-\infty}^{\infty}\frac{1}{(2\pi)^{\frac{MN}{2}}|\mathbf{K}'_\nu|^{\frac{1}{2}}}e^{-\frac{1}{2}(\mathbf{V}-\mu'_\nu)^T\mathbf{K}'^{-1}_\nu(\mathbf{V}-\mu'_\nu)}d\mathbf{V}\\ &=\left(\frac{|\mathbf{K}'_\nu|}{|\mathbf{K}_\nu|}\right)^{\frac{1}{2}}e^{-\frac{1}{2}\mu_\nu^T\mathbf{D}\mu_\nu}\end{aligned}$$

where

$$\mathbf{K}'_\nu = \left(\mathbf{K}_\nu^{-1} + 2\sum_{i=1}^{M} s_i(\phi_M^i \otimes \mathbf{I}_N)\right)^{-1}$$

$$\mu'_\nu = \left(\mathbf{I}_{MN} + 2\sum_{i=1}^{M} s_i(\phi_M^i \otimes \mathbf{I}_N)\mathbf{K}_\nu\right)^{-1} \mu_\nu$$

$$\mathbf{D} = \left(\mathbf{K}_\nu + \sum_{i=1}^{M} \frac{1}{2s_i}(\phi_M^i \otimes \mathbf{I}_N)\right)^{-1}$$

and finally:

$$M_z(\mathbf{s}) = \left(\left|\mathbf{I}_{MN} + 2\sum_{i=1}^{M} s_i\mathbf{K}_\nu(\phi_M^i \otimes \mathbf{I}_N)\right|\right)^{-\frac{1}{2}} e^{-\frac{1}{2}\mu_\nu^T\left(\mathbf{K}_\nu + \sum_{i=1}^{M}\frac{1}{2s_i}(\phi_M^i \otimes \mathbf{I}_N)\right)^{-1}\mu_\nu} \tag{C.14}$$

Using equations C.10 and C.9 and $\mathbf{I}_M = \sum_{i=1}^{M} \phi_M^i$ in C.14 we get:

$$\begin{aligned}
M_z(\mathbf{s}) &= \left|\sum_{i=1}^{M} \phi_M^i \otimes \mathbf{I}_N + 2\sum_{i=1}^{M} s_i(\mathbf{I}_M \otimes \mathbf{K}_x)(\phi_M^i \otimes \mathbf{I}_N)\right|^{-\frac{1}{2}} \\
&\quad \cdot\ e^{-\frac{1}{2}(\mathbf{1}_M^T \otimes \mathbf{X}^T)\left(\mathbf{I}_M \otimes \mathbf{K}_x + \sum_{i=1}^{M}\frac{1}{2s_i}(\phi_M^i \otimes \mathbf{I}_N)\right)^{-1}(\mathbf{1}_M \otimes \mathbf{X})} \\
&= \left|\sum_{i=1}^{M} (\phi_M^i \otimes \mathbf{I}_N + 2s_i\phi_M^i \otimes \mathbf{K}_x)\right|^{-\frac{1}{2}} \\
&\quad \cdot\ e^{-\frac{1}{2}(\mathbf{1}_M^T \otimes \mathbf{X}^T)\left(\sum_{i=1}^{M}(\frac{1}{2s_i}\phi_M^i \otimes \mathbf{I}_N + \phi_M^i \otimes \mathbf{K}_x)\right)^{-1}(\mathbf{1}_M \otimes \mathbf{X})} \\
&= \left|\sum_{i=1}^{M} \phi_M^i \otimes (\mathbf{I}_N + 2s_i\mathbf{K}_x)\right|^{-\frac{1}{2}} \\
&\quad \cdot\ e^{-\frac{1}{2}(\mathbf{1}_M^T \otimes \mathbf{X}^T)\left(\sum_{i=1}^{M}\phi_M^i \otimes (\frac{1}{2s_i}\mathbf{I}_N + \mathbf{K}_x)\right)^{-1}(\mathbf{1}_M \otimes \mathbf{X})} \\
&= \prod_{i=1}^{M} |\mathbf{I}_N + 2s_i\mathbf{K}_x|^{-\frac{1}{2}}\, e^{-s_i\mathbf{X}^T(\mathbf{I}_N + 2s_i\mathbf{K}_x)^{-1}\mathbf{X}} = \prod_{i=1}^{M} M_{z_i}(s_i)
\end{aligned} \tag{C.15}$$

Equation C.15 shows that, for fixed $\mathbf{X}$, the components $z_i = \|\mathbf{r}^i\|^2$ of the vector $\mathbf{z}$ are i.i.d. (independent and identically distributed) random variables. The cumulative distribution function of z_i is given by the inverse Laplace transform as:

$$F_{z_i|\mathbf{X}}(Z_i, \mathbf{X}) = \int_0^{Z_i} p_{z_i|\mathbf{X}}(\lambda, \mathbf{X})\, d\lambda = \mathcal{L}^{-1}\left\{\frac{1}{s} M_{z_i}(s)\right\} \tag{C.16}$$

So we can write:

$$Pr\{||\mathbf{x} - \mathbf{y}^i||^2 > N d^*|\mathbf{X}\} = Pr\{z_i > N d^*|\mathbf{X}\} = 1 - F_{z_i|\mathbf{X}}(N d^*, \mathbf{X}) \tag{C.17}$$

and the conditional matching probability is:

$$P_{m|\mathbf{X}}(N, M, d^*) = 1 - \left(1 - F_{z_i|\mathbf{X}}(N d^*, \mathbf{X})\right)^M \tag{C.18}$$

The matching probability will be:

$$P_m(N, M, d^*) = \int_{-\infty}^{\infty} \dots \int_{-\infty}^{\infty} P_{m|\mathbf{X}}(N, M, d^*, \mathbf{X}) p_x(\mathbf{X})\, d\mathbf{X} \tag{C.19}$$

If the components of the Gaussian vector $\mathbf{x}$ are iid, its covariance matrix is $\mathbf{K}_x = \sigma_x^2 \mathbf{I}_N$. Equation C.15 is then:

$$M_{z_i}(s) = (1 + 2s\sigma_x^2)^{-\frac{N}{2}} e^{-\frac{s\mathbf{X}^T\mathbf{X}}{1+2s\sigma_x^2}} \tag{C.20}$$

We then find the conditional matching probability using C.16, C.18 and C.20 as:

$$P_{m|\mathbf{X}}(N, M, d^*, \mathbf{X}) = 1 - e^{-\frac{M(N d^* + \mathbf{X}^T\mathbf{X})}{2\sigma_x^2}} \left(\sum_{k=0}^{\infty} \frac{1}{k!} \left(\frac{\mathbf{X}^T\mathbf{X}}{2\sigma_x^2}\right)^k \sum_{l=0}^{k+\frac{N}{2}-1} \frac{1}{l!} \left(\frac{N d^*}{2\sigma_x^2}\right)^l\right)^M \tag{C.21}$$

Equation C.21 shows that in the memoryless case $P_{m|\mathbf{X}}(N, M, d^*)$ is a function of the norm of $\mathbf{X}$ and is independent of the shape. So we can define a matching probability conditioned to the norm as:

$$P_{m|||\mathbf{X}||^2}(N, M, d^*, ||\mathbf{X}||^2) = 1 - e^{-\frac{M(N d^* + ||\mathbf{X}||^2)}{2\sigma_x^2}} \left(\sum_{k=0}^{\infty} \frac{1}{k!} \left(\frac{||\mathbf{X}||^2}{2\sigma_x^2}\right)^k \sum_{l=0}^{k+\frac{N}{2}-1} \frac{1}{l!} \left(\frac{N d^*}{2\sigma_x^2}\right)^l\right)^M \tag{C.22}$$

Since the conditional matching probability is independent of the shape, we can take the expected value using the norm's p.d.f instead of the p.d.f of $\mathbf{x}$ when evaluating $P_m(N, M, d^*)$ in equation C.19.

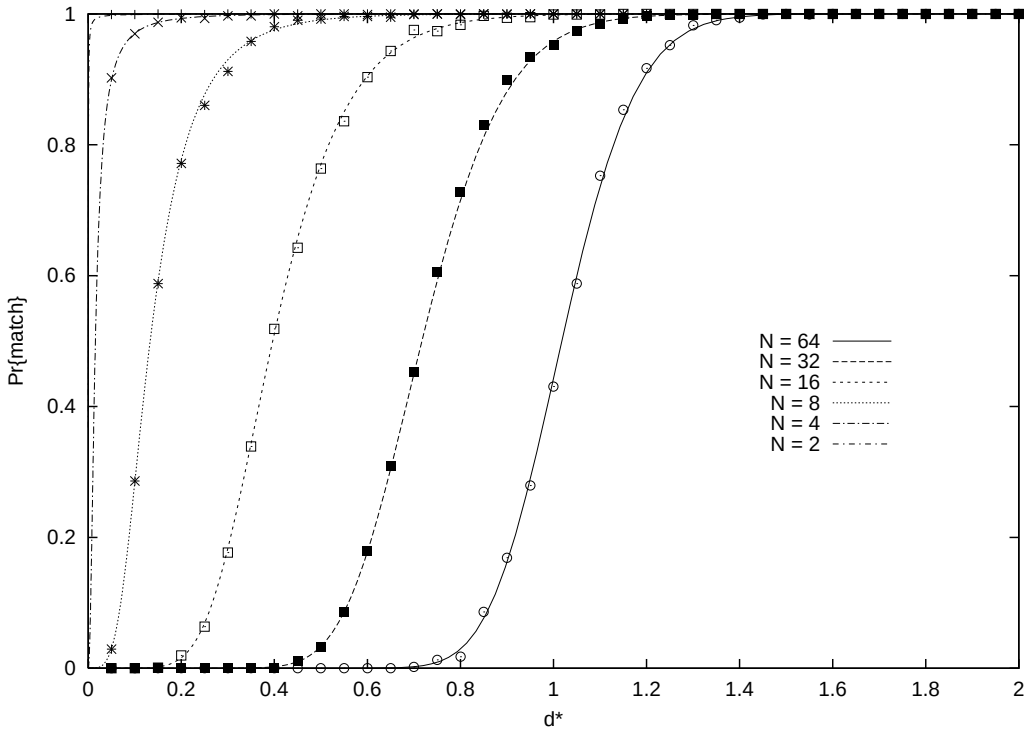


Figura C.1: Matching probability P_m × target distortion d^*, random dictionary.

The componentes of the $\mathbf{x}$ vector are iid zero mean Gaussian variables with variance σ_x^2, so the sum of its squared components is *chi-square* distributed [4].

The matching probability is then:

$$P_m(N, M, d^*) = \int_0^\infty P_{m|\lambda}(N, M, d^*, \lambda) \frac{\lambda^{\frac{N}{2}-1} e^{-\frac{\lambda}{2\sigma_x^2}}}{(2\sigma_x^2)^{\frac{N}{2}} \Gamma(\frac{N}{2})} d\lambda \qquad \text{(C.23)}$$

where $P_{m|\lambda}(N, M, d^*)$ is given by equation C.22 with λ replacing $\|\mathbf{X}\|^2$.

Figure C.1 shows the matching probability given by equation C.23 with $N = 64, 32, \ldots, 2$ and $M = 8192$ compared to experimental results. The experiment was carried on MatLab with 1024 matching trials and averaging the results.

The Probability distribution function $F_d(D) = Pr\{d \leq D\}$ of the per-sample distortion d is given by equation C.23 with $d^* = D$. So the conditional probability density function $f_{d|d\leq d^*}(D)$ of $d|d \leq d^*$, and the conditional expected value $E\{d|d \leq d^*\}$ are:

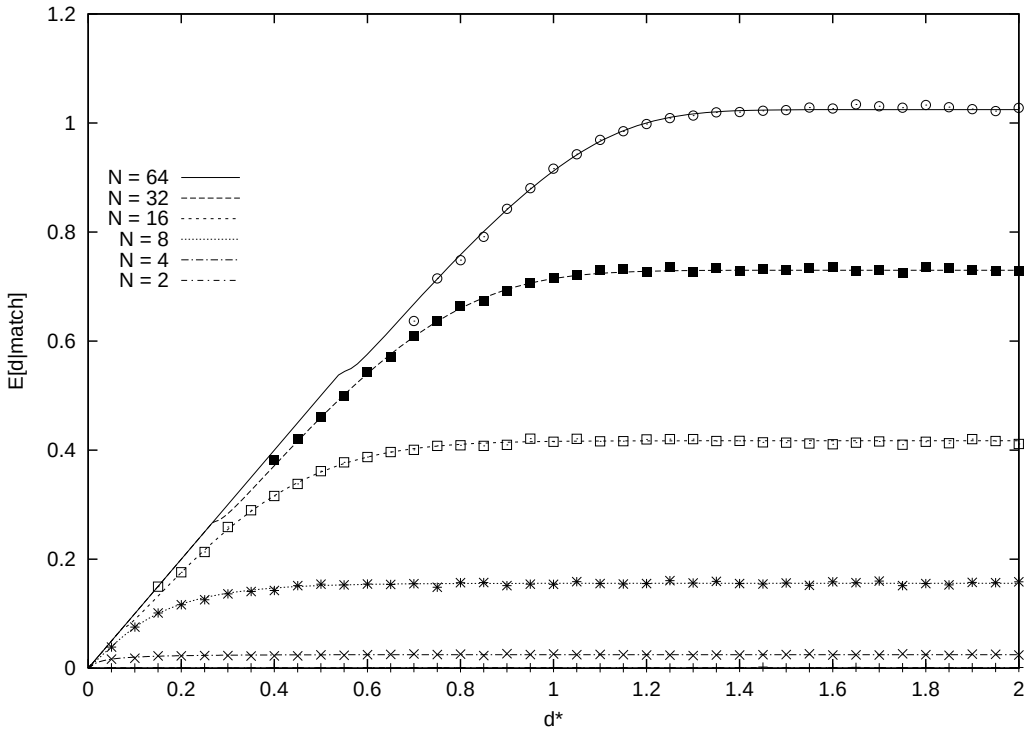


Figura C.2: Mean distortion $Dm \times$ target distortion d^*, random dictionary.

$$\begin{aligned} f_{d|d\leq d^*}(D) &= \frac{1}{F_d(d^*)}\frac{d}{dD}F_d(D) \\ E\{d|d \leq d^*\} &= \frac{1}{F_d(d^*)}\int_0^{d^*} \xi\frac{d}{d\xi}F_d(\xi)d\xi \\ &= d^* - \frac{1}{F_d(d^*)}\int_0^{d^*} F_d(\xi)d\xi \end{aligned} \tag{C.24}$$

If there is a match, the distortion sum conditional expected value $E\{d|match\}$ for a vector of length N is given by equation C.24:

$$Dm(N, M, d^*) = E\{d|match\} = d^* - \frac{1}{F_d(d^*)}\int_0^{d^*} Pm(N, M, \xi)d\xi \tag{C.25}$$

Figure C.2 shows the distortion expected value in equation C.25 with $N = 64, 32, \ldots, 2$ and $M = 8192$ compared to experimental results.

We can see in figure C.2 that, for $N = 64$, the mean distortion for large d^* is greater than the theoretical maximum given by $D_{max} = D(0) = \sigma_x^2 = 1$. This is expected since, for finite M, there is a non-zero probability that the dictionary does not include the optimum solution to the compression problem at rate zero, in

this case the all-zeros vector. Therefore, it is interesting to evaluate the matching probability when we replace one of the dictionary vectors by the all-zeros vector $\mathbf{0}_N$.

With vector $\mathbf{0}_N$ in the dictionary, equation C.18 changes to:

$$\begin{aligned} P_{m|\mathbf{X}}(N, M, d^*, \mathbf{X}) &= \begin{cases} 1 - \left(1 - F_{z_1|\mathbf{X}}(Nd^*, \mathbf{X})\right)^{M-1} & , \quad \|X\|^2 > Nd^* \\ 1 & , \quad \|X\|^2 \leq Nd^* \end{cases} \\ &= 1 - \left(1 - F_{z_1|X}(Nd^*, \mathbf{X})\right)^{M-1} S(\mathbf{X}^T\mathbf{X} - Nd^*) \end{aligned} \tag{C.26}$$

where $S(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$ is the *unitary step function.*

Then, in the memoryless case, equation C.22 becomes:

$$\begin{aligned} P_{m|\|\mathbf{X}\|^2}(N, M, d^*, \|\mathbf{X}\|^2) = 1 - e^{-\frac{(M-1)(Nd^* + \|\mathbf{X}\|^2)}{2\sigma_x^2}} \cdot \\ \cdot \left(\sum_{k=0}^{\infty} \frac{1}{k!} \left(\frac{\|\mathbf{X}\|^2}{2\sigma_x^2} \right)^k \sum_{l=0}^{k+\frac{N}{2}-1} \frac{1}{l!} \left(\frac{Nd^*}{2\sigma_x^2} \right)^l \right)^{M-1} S(\|\mathbf{X}\|^2 - Nd^*) \end{aligned} \tag{C.27}$$

The matching probability is then:

$$P_m(N, M, d^*) = \int_0^{\infty} P_{m|\lambda}(N, M, d^*, \lambda) \frac{\lambda^{\frac{N}{2}-1} e^{-\frac{\lambda}{2\sigma_x^2}}}{(2\sigma_x^2)^{\frac{N}{2}} \Gamma(\frac{N}{2})} d\lambda \tag{C.28}$$

where $P_{m|\lambda}(N, M, d^*, \lambda)$ is given by equation C.27 with λ replacing $\mathbf{X}^T\mathbf{X}$.

Figure C.3 shows the matching probability given by equation C.28 with $N = 64, 32, \ldots, 2$ and $M = 8192$ compared to experimental results. The experiment was carried on MatLab with 1024 matching trials and averaging the results.

The Probability distribution function $F_d(D) = Pr\{d \leq D\}$ of the distortion d is given by equation C.28 with $d^* = D$. If there is a match, the distortion sum conditional expected value $E\{d|match\}$ for a vector of length N is given by:

$$D_m(N, M, d^*) = E\{d|match\} = d^* - \frac{1}{F_d(d^*)} \int_0^{d^*} P_m(N, M, \xi) d\xi \tag{C.29}$$

where $P_m(N, M, \xi)$ is given by equation C.28.

Figure C.4 shows the expected value for the distortion $E\{d\}\times$ the target distortion d^* for several block-lengths N and $M = 8192$. As expected, the dictionary

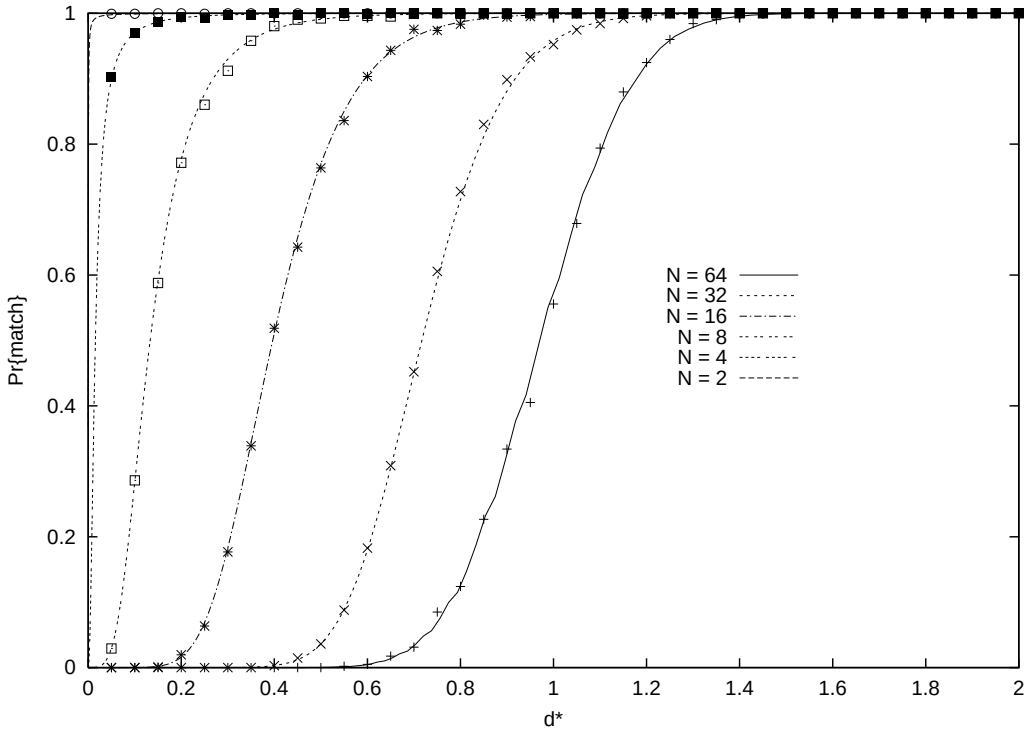


Figura C.3: Matching probability Pm $\times$ target distortion d^*, random dictionary with zero included.

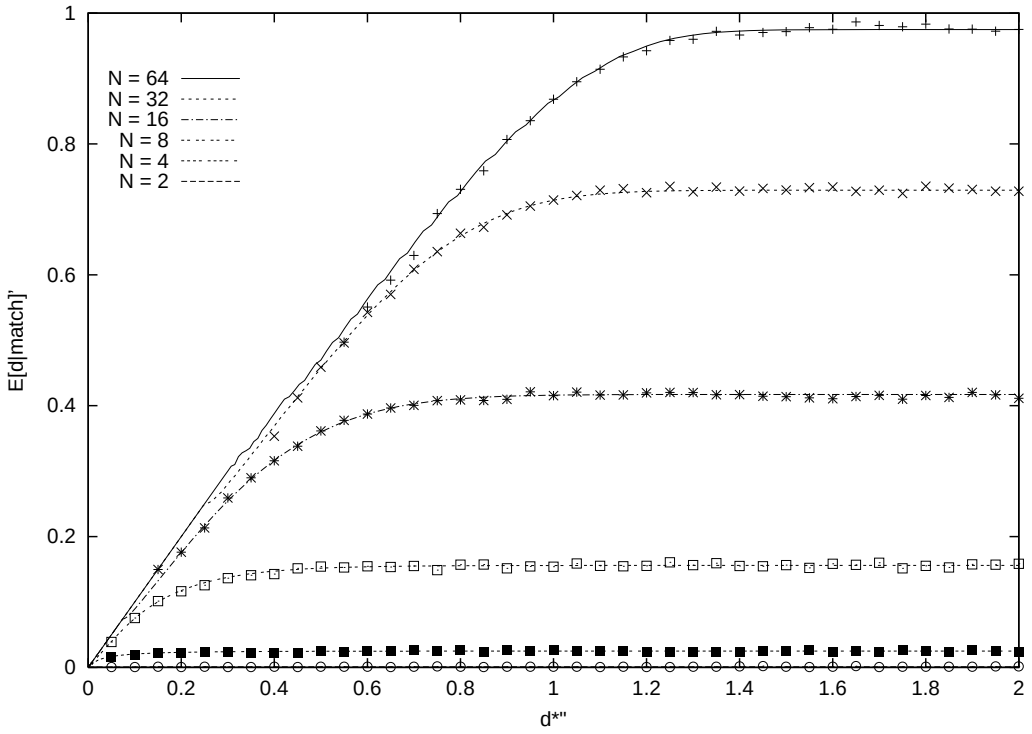


Figura C.4: Dm $\times$ target distortion d^*, random dictionary with zero included.

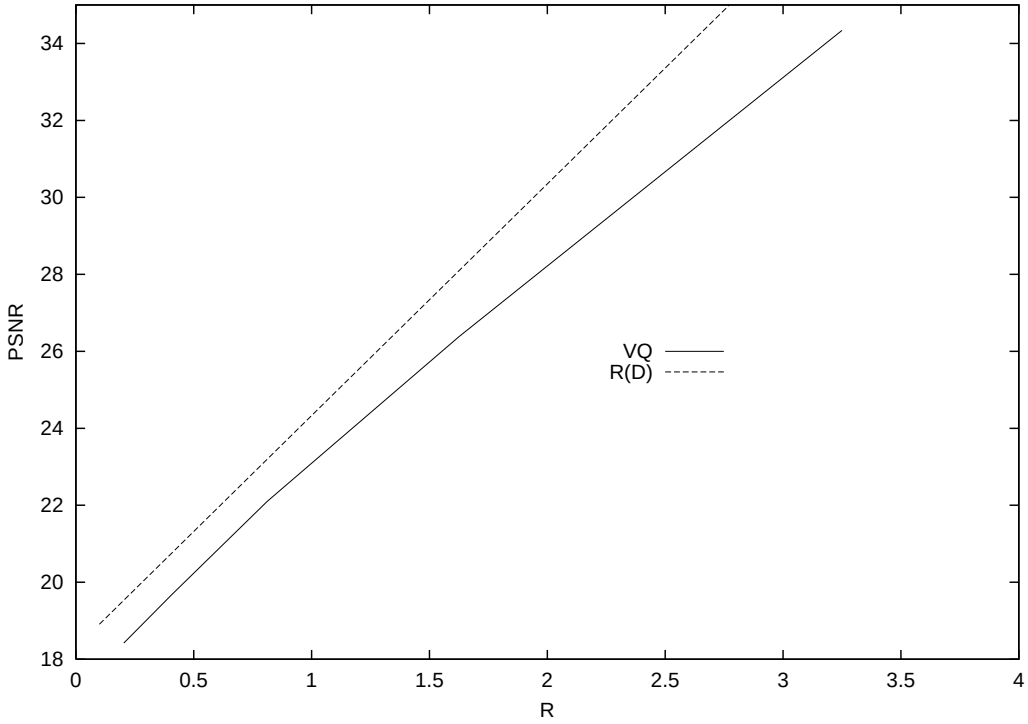


Figura C.5: Rate-distortion performance of a 8192-level random vector-quantizer.

with $\mathbf{0}_N$ included performs better, with mean distortion below or equal to σ_x^2 for all values of N.

In figures C.2 and C.4 we can see that for high target distortion, the distortion reaches a maximum for every N. If we use the random dictionary as a vector quantizer (setting $d^* \to \infty$), we get the rate-distortion performance shown in figure C.5 . In this figure, we show the peak-signal-to-noise-ratio $PSNR = 10\log_{10}(N255^2/Dm(N, M, \infty))$, versus the rate given by $R = \log_2(M)/N$, for the dictionary with the zero included. The performance difference for the dictionary without the zero is only $0.22dB$ at $R = 0.2031$ for this value of $M = 8192$ but it increases dramatically for smaller values of M.

C.2 Matching Gaussian vectors at different scales

A distinctive feature of this work is the approximate matching of patterns using scales. Therefore it is interesting to analyze the matching of a Gaussian vector of size N to vectors in a dictionary of Gaussian vectors of size $N/2, N/4, \ldots, 2$ interpolated to size N.

We will use the notation $\mathbf{y}^s = T^N_{N'}[\mathbf{y}]$ to denote the scaling operation that changes from a vector of size $N' \times 1$ to another vector of size $N \times 1$. One way to implement the interpolation operation is by use of a DCT [29]. If we want to change from a vector $\mathbf{y}$ of size $N' \times 1$ to a vector $\mathbf{y}^s$ of size $N \times 1$ ($N' \leq N$), we first compute the N' point DCT of $\mathbf{y}$, given by $\mathcal{Y} = \mathbf{C}_{N'}\mathbf{y}$. Then we append $N - N'$ zeros to $\mathcal{Y}$ obtaining $\mathcal{Y}^s = \begin{pmatrix} \mathcal{Y} \\ \mathbf{0}_{N-N'} \end{pmatrix}$. Finally we get the upsampled version by the N point inverse DCT of $\mathcal{Y}^s$, $\mathbf{y}^s = \mathbf{C}_N^T\mathcal{Y}^s$. Since the DCT is a unitary transformation, the matching probability can be evaluated in the transformed domain. In fact:

$$\begin{aligned} Pr\{\|\mathcal{X} - \mathcal{Y}^s\|^2 \leq N d^*\} &= Pr\{\|\mathbf{C}_N\mathbf{x} - \mathbf{C}_N\mathbf{y}^s\|^2 \leq N d^*\} \\ &= Pr\{(\mathbf{C}_N\mathbf{x} - \mathbf{C}_N\mathbf{y}^s)^T(\mathbf{C}_N\mathbf{x} - \mathbf{C}_N\mathbf{y}^s) \leq N d^*\} \\ &= Pr\{(\mathbf{x} - \mathbf{y}^s)^T\mathbf{C}_N^T\mathbf{C}_N(\mathbf{x} - \mathbf{y}^s) \leq N d^*\} \\ &= Pr\{\|\mathbf{x} - \mathbf{y}^s\|^2 \leq N d^*\} \end{aligned} \quad (C.30)$$

Since the DCT is a linear transformation, the transformed version of the Gaussian vector $\mathbf{x}$ is also Gaussian [4] with covariance matrix given by:

$$\mathbf{K}_{\mathcal{X}} = E\{\mathcal{X}\mathcal{X}^T\} = E\{\mathbf{C}_N^T\mathbf{x}\mathbf{x}^T\mathbf{C}_N\} = \mathbf{C}_N^T\mathbf{K}_{\mathbf{x}}\mathbf{C}_N \quad (C.31)$$

If $\mathbf{x}$ is memoryless then $\mathcal{X}$ will be also memoryless. In fact, $\mathbf{K}_{\mathbf{x}} = \sigma^2\mathbf{I}_N$ and $\mathbf{K}_{\mathcal{X}}$ is given by:

$$\mathbf{K}_{\mathcal{X}} = \mathbf{C}_N^T\sigma^2\mathbf{I}_N\mathbf{C}_N = \sigma^2\mathbf{C}_N^T\mathbf{C}_N = \sigma^2\mathbf{I}_N = \mathbf{K}_{\mathbf{x}} \quad (C.32)$$

Therefore the matching probability of a $N \times 1$ memoryless Gaussian vector $\mathbf{x}$ considering a dictionary of scaled Gaussian vectors of original size $N' \times 1$ is the matching probability of an $N \times 1$ memoryless Gaussian vector with a dictionary of memoryless vectors of size $N' \times 1$ with $N - N'$ zeros appended to each of the dictionary vectors.

$$\begin{aligned} Pr\{\|\mathbf{x} - T_{N'}^{N}[\mathbf{y}^i]\|^2 \leq N d^*\} &= Pr\{\|\mathbf{x} - \begin{pmatrix} \mathbf{y}^i \\ \mathbf{0}_{N-N'} \end{pmatrix}\|^2 \leq N d^*\} \\ &= Pr\{\|\mathbf{x}' - \mathbf{y}^i\|^2 + \|\mathbf{x}''\|^2 \leq N d^*\} \\ &= E_{\mathbf{x}''}\{Pr\{\|\mathbf{x}' - \mathbf{y}^i\|^2 \leq N d^* - \|\mathbf{X}''\|^2\}\} \quad \text{(C.33)} \end{aligned}$$

where $\mathbf{x}'$ is a memoryless Gaussian vector of size $N' \times 1$ and $\mathbf{X}''$ is a realization of the memoryless Gaussian vector $\mathbf{x}''$ of size $N - N' \times 1$. The matching probability with scales $Pm^s(N, N', M, d^*)$ is then:

$$Pm^s(N, N', M, d^*) = \int_0^{N d^*} Pm(N', M, \frac{N d^* - \lambda}{N'}) \frac{\lambda^{\frac{N-N'}{2}-1} e^{-\frac{\lambda}{2\sigma_x^2}}}{(2\sigma_x^2)^{\frac{N-N'}{2}} \Gamma(\frac{N-N'}{2})} d\lambda \quad \text{(C.34)}$$

Figure C.6 shows the matching probability with scales for a memoryless Gaussian vector of size 64×1 using a dictionary of 8192 vectors of lengths $64, 32, 16, \ldots, 2$. It is interesting that, in this example, the probability of match for a dictionary of vectors using scales is greater than that of a dictionary of original size $N' = 64$. This can be understood considering:

- With $N = 64$ and $M = 8192$ the approximation to $\mathbf{X}$ using the dictionary without scales will have mean rate $R = 0.2031$ and mean distortion $D = 0.9747$. At this distortion level, a low-pass version of $\mathbf{X}$ will usually be sufficient to approximate the source. Therefore, there is little to gain with a dictionary of vectors of size $N' = 64$ over another dictionary of vectors of size $N' = 32$, for example.

- The matching with scales can be viewed as a match of vectors of size N' and a match of a vector of size $N - N'$ with the all-zeros vector. As the size N' decreases, it becomes easier to match the first part of size N' because we keep the dictionary size M fixed (this means we have more bits available to each component of the vector to match) but on the other hand it is harder to match the size-$(N - N')$ vector to the all-zeros vector. In the example illustrated in figure C.6, the matching probability is maximum when $N' = 32$ or $N' = 16$.

Figure C.7 shows the mean distortion given a match with scales for a memoryless Gaussian vector of size 64×1 using a dictionary of 8192 vectors of lengths

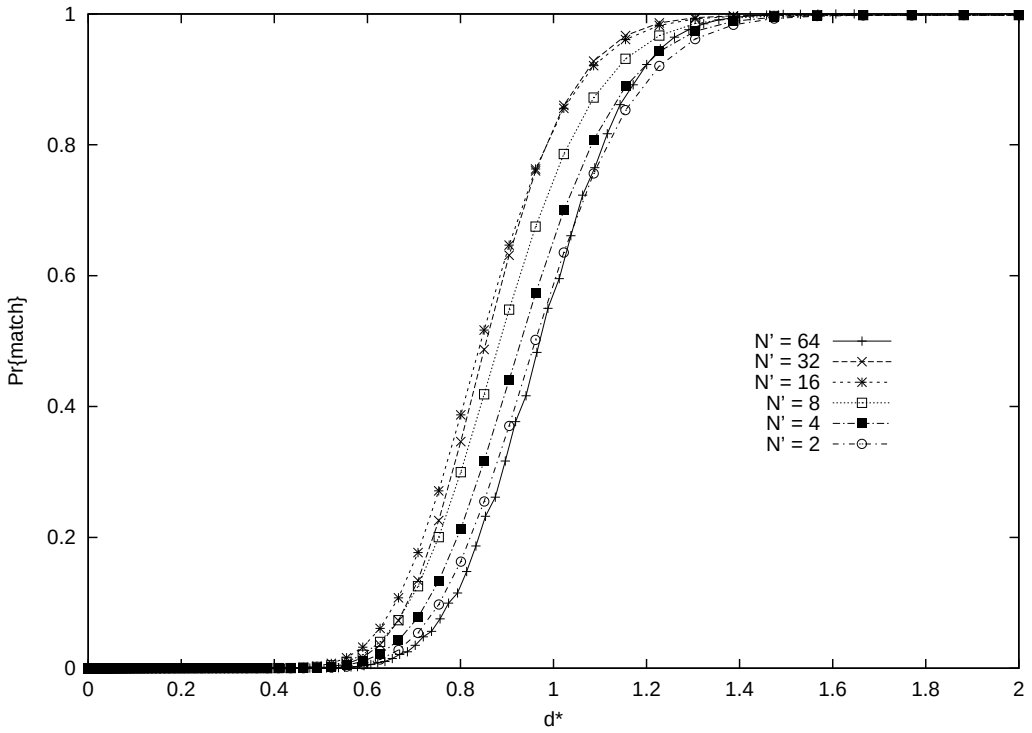


Figura C.6: Matching with scales for a dictionary with 8192 vectors.

$64, 32, 16, \ldots, 2$. Since the matching probability with scales is greater in this case, the mean distortion is lower for the matching with scales.

We can conclude that *the matching with scales can improve the performance of approximate pattern matching at low-rates.*

C.3 Asymptotic behavior of the matching probability

In this section we will investigate what happens to the matching probability when the blocklength N is large. For a dictionary of M vectors, we use (without entropy coding) $\log_2(M)$ bits to represent an index. If the input vector is of size N, the rate will be $R = \log_2(M)/N$ bits per sample. If we make $M = 2^{NR}$ we can vary the blocklength N while keeping the rate fixed. Figure C.8 shows the matching probability variation for several values of N and rate $R = 0.25$. The curves suggest that, for N large enough, the matching probability variation with d^* will be similar to a unitary step function.

To evaluate the behavior of the matching probability for very large N we can

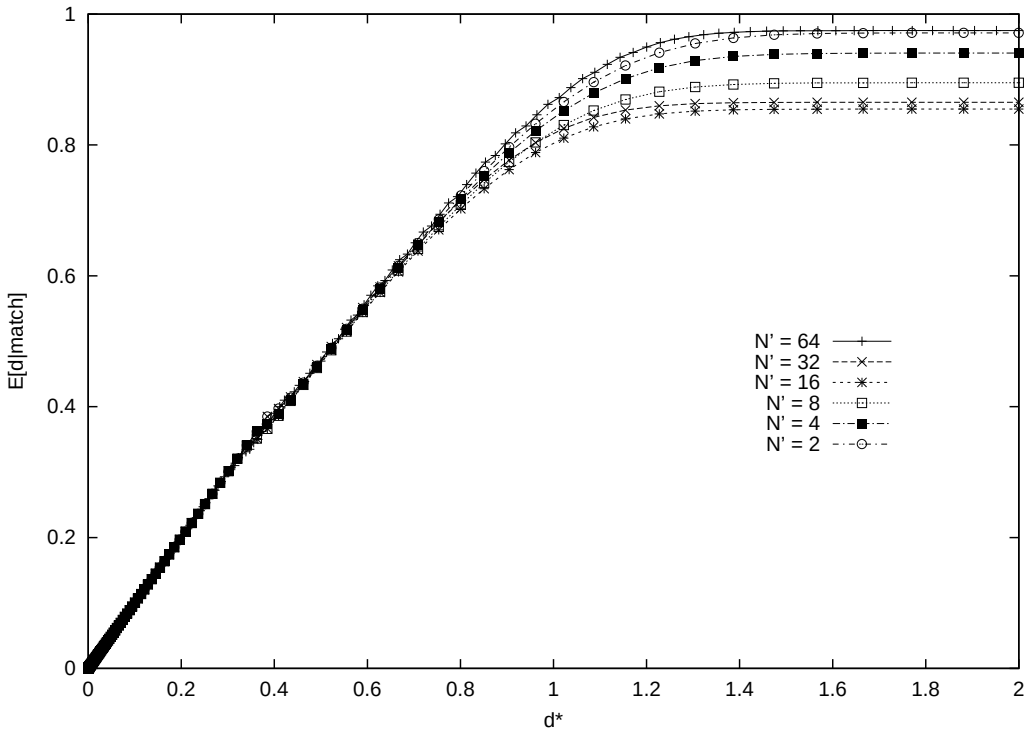


Figura C.7: Distortion in a match with scales for a dictionary with 8192 vectors.

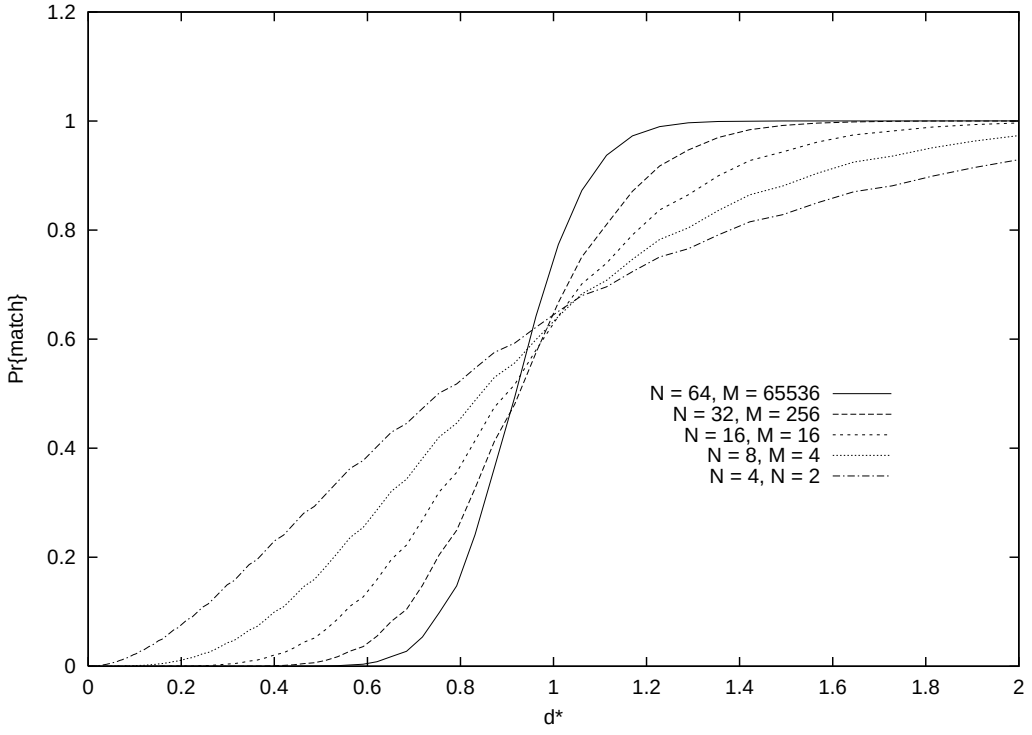


Figura C.8: Matching probability variation for increasing blocklength values N.

calculate the limit:

$$\begin{aligned} Pm(R, d^*) &= \lim_{N\to\infty} Pm(N, 2^{NR}, d^*) \\ &= \lim_{N\to\infty} \int_0^\infty P_{m|\lambda}(N, 2^{NR}, d^*, N\lambda) \frac{N^{\frac{N}{2}} \lambda^{\frac{N}{2}-1} e^{-\frac{N\lambda}{2\sigma_x^2}}}{(2\sigma_x^2)^{\frac{N}{2}} \Gamma(\frac{N}{2})} d\lambda \end{aligned} \tag{C.35}$$

We can recognize as part of the integral the *delta sequence* or *Dirac* [24] as:

$$\lim_{N\to\infty} \frac{N^{\frac{N}{2}} \lambda^{\frac{N}{2}-1} e^{-\frac{N\lambda}{2\sigma_x^2}}}{(2\sigma_x^2)^{\frac{N}{2}} \Gamma(\frac{N}{2})} = \delta(\lambda - \sigma_x^2) \tag{C.36}$$

$P_{m|\lambda}(N, 2^{NR}, d^*, N\lambda)$ always exist and belongs to the interval $[0, 1]$. If it is also continuous in the variable λ at the point $\lambda = \sigma_x^2$, then the sampling property of the Dirac holds and we expect that:

$$Pm(R, d^*) = \lim_{N\to\infty} P_{m|\|\mathbf{x}\|^2}(N, 2^{NR}, d^*, N\sigma_x^2) \tag{C.37}$$

Since the matching probability seems to approach a step function, we expect from equation C.37 the same behavior for the conditional matching probability for large N. Moreover, if the dictionary is optimal in an R(D) sense, then this step function should be:

$$\lim_{N\to\infty} P_{m|\|\mathbf{x}\|^2}(N, 2^{NR}, d^*, N\sigma_x^2) = S(d^* - \sigma_x^2 2^{-2R}) \tag{C.38}$$

In the case of a match with scales, we can evaluate the limit:

$$\begin{aligned} & Pm^s(K, R, d^*) = \lim_{N\to\infty} Pm^s(\frac{N}{K}, N, 2^{NR}, d^*) \\ = & \lim_{N\to\infty} \int_0^{Nd^*} Pm(\frac{N}{K}, 2^{NR}, K(d^* - \frac{\lambda}{N})) \frac{\lambda^{\frac{N}{2}(1-\frac{1}{K})-1} e^{-\frac{\lambda}{2\sigma_x^2}}}{(2\sigma_x^2)^{\frac{N}{2}(1-\frac{1}{K})} \Gamma(\frac{N}{2}(1-\frac{1}{K}))} d\lambda \\ = & \begin{cases} 0, & d^* \le \left(1 - \frac{1}{k}\right) \sigma_x^2 \\ \lim_{N\to\infty} Pm\left(\frac{N}{K}, 2^{-NR}, Kd^* - (K-1)\sigma_x^2\right), & d^* > \left(1 - \frac{1}{k}\right) \sigma_x^2 \end{cases} \end{aligned} \tag{C.39}$$

where $K = \frac{N}{N'}$ is the upsampling factor.

If the random dictionary with zero is optimum in an R(D) sense, that is equation C.38 holds, then the matching with scales for large N becomes:

$$Pm^s(K, R, d^*) = S\left(d^* - \sigma_x^2\left(1 + \frac{2^{-2KR} - 1}{K}\right)\right) \tag{C.40}$$

Equation C.40 shows that, if the ordinary matching is optimum for large N, then the matching with scales is not unless $R = 0$. For rates greater than zero the penalty in performance increases as the rate increases. For example, for $K = 2, R = 0.5$ the performance of the matching with scales will be 0.9 dB worse than the optimum $D(R)$.

It is interesting that, for low-rates and small blocklengths N, the matching with scales can perform better than the ordinary matching, as pointed out in section C.2; while the opposite happens when N becomes indefinitely large. This suggests the existence of a threshold value $T(R, K)$ for the blocklength N, that indicates when the matching with scales becomes worse than the ordinary match. The knowledge of the exact value of the threshold could be used to improve the performance of a system based on approximate pattern matching with scales, for example, by dynamically switching the scale calculation on and off. However, practical implementations of compression algorithms based on approximate pattern matching with scales will always have finite blocklengths and, since larger blocklengths imply greater complexity, it might be that many cases of interest are below the threshold value.

Apêndice D

UMMP

In this appendix we propose a compression algorithm that we call UMMP (Universal Multiscale Matching Pursuits). It is based on *approximate matching with scales of recurrent patterns*. The UMMP algorithm adaptively builds a dictionary $\mathcal{D}$ that is updated with concatenations of *dilated/contracted* versions of previously occurred patterns while encoding the data. UMMP uses the dictionary as a *basis* to decompose the input data, That is, UMMP tries to reconstruct an input block of data using a *linear combination* of the elements in the dictionary.

D.1 UMMP description

In appendix B we discussed generic techniques to solve the data compression problem. A popular approach is to perform the compression in three steps: transformation, quantization and entropy encoding. The Matching Pursuits algorithm (MP), described in section B.3, is one of the options that can be used in the transformation step. It operates with an overcomplete fixed dictionary $\mathcal{D}$ that is known before the expansion of the data begins. In a sense the expansion is adaptive since MP explicitly chooses the subset $\mathcal{D}_{\text{opt}}$ of the dictionary $\mathcal{D}$ that better fits the input data. However, the level of adaptation available depends on the amount of redundancy of the dictionary vectors. If the dictionary is highly redundant, the expansion is highly adaptable but the cost to encode the dictionary indexes is also high. It would be interesting to investigate a method that, instead of just using a subset $\mathcal{D}_{\text{opt}}$ of the dictionary, attempts to build $\mathcal{D}_{\text{opt}}$ directly while encoding the data. In

order to do this, we can borrow some ideas from the Lempel-Ziv algorithm.

The Lempel-Ziv algorithm (LZ) [5] is a lossless compression algorithm based on matching variable-sized vectors extracted from the input signal with variable-sized vectors in a dictionary. The LZ algorithm builds its dictionary by including in it concatenations of previously encoded vectors. Since it is lossless, the components of the vectors constructed by concatenation are likely to form *typical sequences* [2]. Therefore the LZ algorithm performs entropy coding because the dictionary is composed by typical sequences that become almost equally probable as the size of the dictionary vectors increases. We can use the rule of concatenation of previously encoded vectors to build a lossy compression algorithm based on Matching Pursuits that attempts to build a dictionary $\mathcal{D}_{\text{opt}}$. If we use the concatenation rule, we can also expect that some entropy encoding will be performed by the dictionary and its updating procedure, as happens with the LZ.

The LZ algorithm is based on matching of recurrent patterns. That is, it uses previously encoded vectors, belonging to a dictionary known to both encoder and decoder, to represent an input vector. We can build a lossy algorithm if we allow the matching to be approximate, that is a vector in the dictionary can be used to represent an input vector if the distortion incurred is considered acceptable. The LLZ algorithm, presented in [8], describes one of these extensions to the LZ algorithm that is controlled by a target distortion parameter d^*. In this work we use a new concept that we call *approximate matching with scales*. This means that a vector $\mathbf{S}$ can be used to approximate another vector $\mathbf{X}$ *regardless of their lengths*. That is, we say that the vector $\mathbf{S}$ can be used to represent the vector $\mathbf{X}$ if $\mathbf{S}^s$ is close enough to $\mathbf{X}$, where $\mathbf{S}^s$ is a version of $\mathbf{S}$ with the same length as $\mathbf{X}$. To change the length, or scale, we use a scale transformation $\mathbf{S}^s = \mathsf{T}^{\ell(\mathbf{S})}_{\ell(\mathbf{X})}[\mathbf{S}]$ that maps a vector of length $\ell(\mathbf{S})$ (the length of $\mathbf{S}$) to a vector of length $\ell(\mathbf{X})$ (the length of $\mathbf{X}$). This operation is ilustrated in figure D.1(a), for $\ell(\mathbf{X}) > \ell(\mathbf{S})$, and in figure D.1(b), for $\ell(\mathbf{X}) < \ell(\mathbf{S})$. From the theoretical analysis we did in appendix C we know that the use of scales can improve the performance of a compression scheme based on approximate pattern matching, especially at low rates. Figure D.2 illustrates the concept of approximate matching with scales. In this example the vector $\mathbf{S}^s$ is within distance $\mathsf{d} = 1$ from the vector $\mathbf{X}$. If this distortion is acceptable, we can use $\mathbf{S}^s$

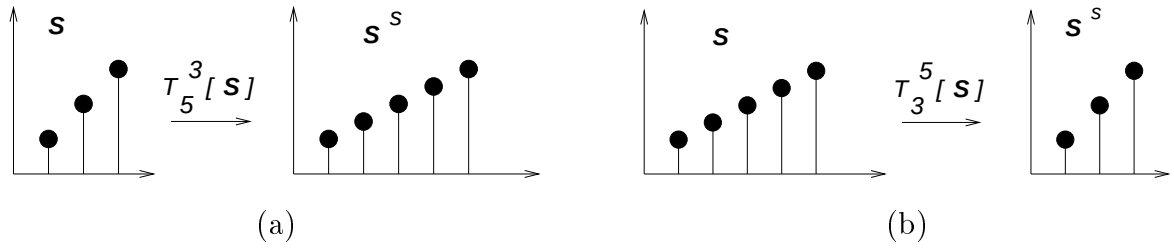


Figura D.1: Scale transformation $\mathbf{S}^s = \mathsf{T}_N^M[\mathbf{S}]$: (a) N > M; (b) N < M

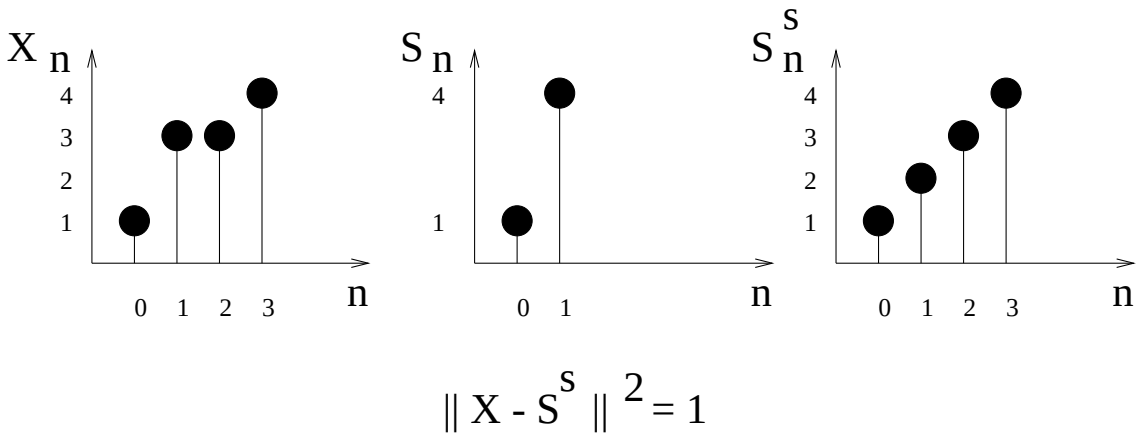


Figura D.2: Approximate matching with scales.

to represent the input vector $\mathbf{X}$. In this case, we can say that we can make an approximate match with scales of $\mathbf{X}$ and $\mathbf{S}$.

In the MP algorithm described in section B.3.3, an input vector is initially represented by a vector in the dictionary multiplied by a scalar value. If this first representation is not accurate enough, a residue vector is evaluated and the procedure is repeated for this residue. We can interpret the MP algorithm as a successive approximation scheme based on approximate pattern matching. This algorithm can be naturally extended to use approximate matching with scales if we introduce the scale transformation and allow the dictionary to contain vectors of different lengths. Also, the algorithm can be modified in order to update the dictionary by concatenation while encoding the data. Our idea is to split every residue vector in two and repeat the procedure for each segment. This way, the successive approximation procedure will start with one vector, then proceed with two smaller residue vectors. That is, in the first iteration we have two residue vectors, in the second iteration we can have up to four residue vectors, in the third iteration we can have up to eight vectors, and so on. The number of residue vectors at iteration n is not strictly

equal to 2^n because each residue vector is independently matched (with scales) to the vectors in the dictionary. This means that at iteration n some of the residues vectors will be split but others won't. Only those residues that cannot be approximated with sufficient accuracy need to be split. Therefore, while in the original MP there is only one residue vector at each stage, in our algorithm the number of residue vectors at stage n can be anywhere in the interval $[2, 2^n]$. The different sizes are easily accommodated by the approximate pattern matching with scales. Since the expansion now contains vectors of different lengths, at any iteration we can use concatenations of vectors already encoded to update the dictionary.

For example, let $\mathbf{R}^0 = \mathbf{X}$ be a vector of length N to be encoded using a dictionary $\mathcal{D}$. As in MP, we chose the best vector $\mathbf{S}_{k_0}$ in the dictionary and evaluate the residue $\mathbf{R}^1 = \mathbf{R}^0 - < \mathbf{S}_{k_0}, \mathbf{R}^0 > \mathbf{S}_{k_0}$. But instead of evaluating $\mathbf{R}^2$ using the same dictionary, we parse the residue $\mathbf{R}^{1,0} = \mathbf{R}^1$ in two segments, $\mathbf{R}^{1,1}$ of length p and $\mathbf{R}^{1,2}$ of length $N - p$. We use the second superscript to index the segments. Then we change the length of all the elements in the dictionary to length p, by the use of a scale transformation $T_p^N[\cdot]$ (see figure D.1), and use them to evaluate the new residue $\mathbf{R}^{2,1} = \mathbf{R}^{1,1} - < T_p^N[\mathbf{S}_{k_{1,1}}], \mathbf{R}^{1,1} > T_p^N[\mathbf{S}_{k_{1,1}}]$. The same procedure is applied to the second segment of the first residue $\mathbf{R}^{1,2}$. As in MP, we can stop the process when a given distortion level is achieved. Therefore, for a given target distortion d^*, there is a segmentation tree associated to the recursive parsing of the residues, as illustrated in figure D.3.

In this tree we use the first superscript i in $\mathbf{R}^{i,j}$ to indicate the number of projections in the dictionary already made to generate $\mathbf{R}^{i,j}$, and the second superscript j is used to address the multiple segments. We number the nodes of the segmentation tree such that a parent node j has two children with indexes $2j + 1$ and $2j + 2$.

In figure D.3, the squared norm $\|\mathbf{R}^{3,3}\|^2$, for example, is not greater than the target distortion d^* times the length of the residue, otherwise the segmentation would continue. This is true for all leaf nodes of the segmentation tree. Therefore all leaf nodes of the segmentation tree can be approximated by the zero vector of the corresponding length and the overall distortion of the approximation for the original vector $\mathbf{X}$ is guaranteed to be within Nd^*.

This segmentation procedure allows the use of our rule of concatenation to

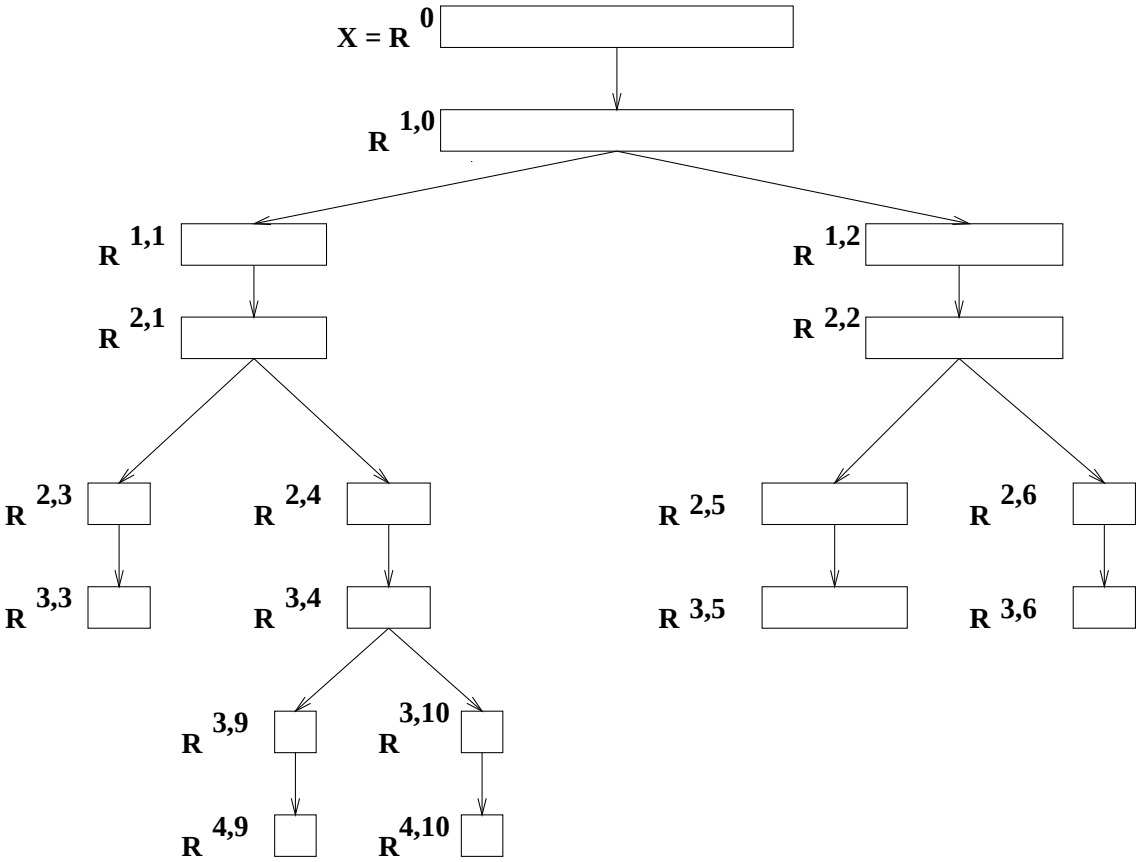


Figura D.3: Segmentation tree in UMMP.

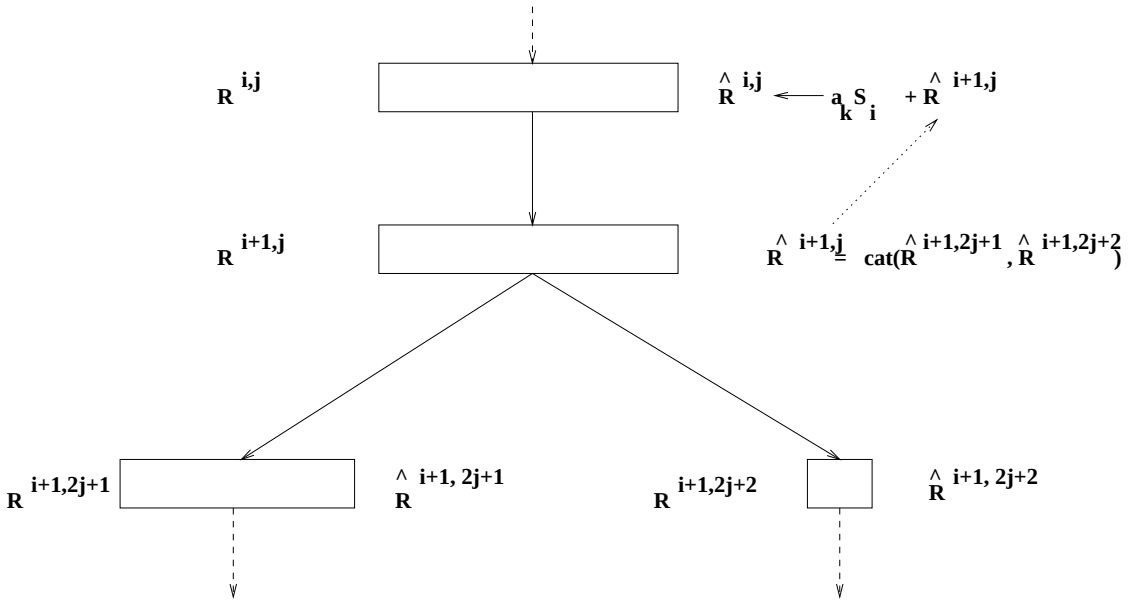


Figura D.4: Update of the dictionary in UMMP.

update the dictionary $\mathcal{D}$ as follows: If we have the approximations to the two segments $\mathbf{R}^{i+1,2j+1}$ and $\mathbf{R}^{i+1,2j+2}$, denoted respectively by $\hat{\mathbf{R}}^{i+1,2j+1}$ and $\hat{\mathbf{R}}^{i+1,2j+2}$, we can update the dictionary by including the concatenation of the two approximations in $\mathcal{D}$. We can also add the result of the concatenation to the approximation $< \mathbf{S}_{k_{i,j}}, \mathbf{R}^{i,j} > \mathbf{S}_{k_{i,j}}$ of the parent node $\mathbf{R}^{i,j}$ that we have previously made using the dictionary, and include the sum in $\mathcal{D}$. This process is illustrated in figure D.4.

We now describe UMMP (Universal multiscale Matching Pursuits), an algorithm to lossy compress data from any source without previous knowledge of it's statistics. UMMP operates on a vector $\mathbf{X} = (X_0 \dots X_{N-1})^T$ output by a vector source $\mathbf{x} = (x_0 \dots x_{N-1})^T$. It uses:

1) A dictionary $\mathcal{D}$ of vectors, initially set to $\mathcal{D}_0 = \{\mathbf{S}_0, \dots, \mathbf{S}_{M-1}\}$, as the frame to expand the data,

2) A scalar quantizer defined by the set of reproducing values $\mathcal{Q} = \{\alpha_0, \dots, \alpha_{L-1}\}$ and

3) A target distortion d^*.

The dictionary vectors are not normalized. To begin the expansion, all the vectors in $\mathcal{D}_0$ are initially scaled to size N using a transformation $\mathbf{S}_k^s = T_N^{\ell(\mathbf{S}_k)}[\mathbf{S}_k]$, where $\ell(\mathbf{S}_k)$ is the length of $\mathbf{S}_k$. The scale transformation is a function $T_N^M : \mathbb{R}^M \rightarrow$

$\mathbb{R}^N$ that maps a vector of size M into a vector of size N, as illustrated in figure D.1. For the sake of simplicity, we will drop the superscript $\ell(\mathbf{S}_k)$ and write just $T_N[\mathbf{S}_k]$ for the scale transformation. In the spirit of matching pursuits, UMMP searches in the dictionary for the vector $\mathbf{S}^s_{k_{0,0}}$ which maximizes the inner product $< \mathbf{S}^s_{k_{0,0}}/\|\mathbf{S}^s_{k_{0,0}}\|, \mathbf{X} >= {\mathbf{S}^s_{k_{0,0}}}^T\mathbf{X}/\|\mathbf{S}^s_{k_{0,0}}\|$. It then chooses $\alpha_{l_{0,0}} \in \mathcal{Q}$ to minimize the residue $\mathbf{R}^{1,0} = \mathbf{X} - \alpha_{l_{0,0}}\mathbf{S}_{k_{0,0}}$. If the residue's squared norm $\|\mathbf{R}^{1,0}\|^2$ is not greater than the target distortion d^* times the vector size N, then the expansion is finished. Otherwise, the residue $\mathbf{R}^{1,0} = (R^{1,0}_0 \dots R^{1,0}_{N-1})^T$ is parsed in two segments $\mathbf{R}^{1,1} = (R^{1,0}_0 \dots R^{1,0}_{p-1})^T$ and $\mathbf{R}^{1,2} = (R^{1,0}_p \dots R^{1,0}_{N-1})^T$, $p \in [0, N-2]$.

For example, let $\mathbf{X} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 6 \end{pmatrix}^T$, $\mathcal{D}_0 = \{1\}$, $d^* = 0.5$ and $\mathcal{Q} = \{-2, -1, -0.5, 0, 0.5, 1, 2\}$.

- We first scale all vectors in the dictionary to length 7, that is, $\mathbf{S}^s_0 = T_7[\mathbf{S}_0] = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}^T$.
- Then we evaluate the inner product $< \mathbf{X}, \mathbf{S}^s_0/\|\mathbf{S}_0\| >= 10.2050$.
- To minimize the residue's norm we choose $\hat{\mathbf{X}} = \alpha_6\mathbf{S}^s_0 = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 \end{pmatrix}^T$. Therefore $\mathbf{R}^{1,0} = \mathbf{X} - \hat{\mathbf{X}} = \begin{pmatrix} -1 & 0 & 1 & 2 & 3 & 4 & 4 \end{pmatrix}^T$ and $\|\mathbf{R}^{1,0}\|^2 = 47$.
- Since $\|\mathbf{R}^{1,0}\|^2 > N d^* = 3.5$ the residue is split in two segments $\mathbf{R}^{1,1}$ and $\mathbf{R}^{1,2}$.

The segmentation point p can be chosen to minimize some performance criterion. For example, we could choose p such that $\|\mathbf{R}^{1,1} - \hat{\mathbf{R}}^{1,1}\|^2 + \|\mathbf{R}^{1,2} - \hat{\mathbf{R}}^{1,2}\|^2$ is minimum, where $\hat{\mathbf{R}}^{1,1}$ and $\hat{\mathbf{R}}^{1,2}$ are the best approximations to $\mathbf{R}^{1,1}$ and $\mathbf{R}^{1,2}$ that we can make using the current dictionary $\mathcal{D}$. In the above example, the best p equals 2 so $\mathbf{R}^{1,1} = \begin{pmatrix} -1 & 0 \end{pmatrix}^T$ and $\mathbf{R}^{1,2} = \begin{pmatrix} 1 & 2 & 3 & 4 & 4 \end{pmatrix}^T$. At this moment we have an integer sequence, composed by a scalar quantizer index $l = 6$, a dictionary index $k = 0$, a one bit flag 0 to indicate splitting and a segmentation point $p = 2$, representing the encoded data. This process is illustrated in figure D.5.

Next, all the vectors in $\mathcal{D}$ are scaled to size p using a transformation $\mathbf{S}^s_k = T_p[\mathbf{S}_k]$ and the same procedure is recursively applied to $\mathbf{R}^{1,1}$. Then the vectors in $\mathcal{D}$ are scaled to size $N - p$ using a transformation $\mathbf{S}^s_k = T_{N-p}[\mathbf{S}_k]$ and the process is repeated for $\mathbf{R}^{1,2}$. After that, we will have two approximations $\hat{\mathbf{R}}^{1,1}$ and $\hat{\mathbf{R}}^{1,2}$, and we can then form an estimate of the original residue $\mathbf{R}^{1,0}$ as $\hat{\mathbf{R}}^{1,0} = (\hat{\mathbf{R}}^{1,1^T}\hat{\mathbf{R}}^{1,2^T})^T$.

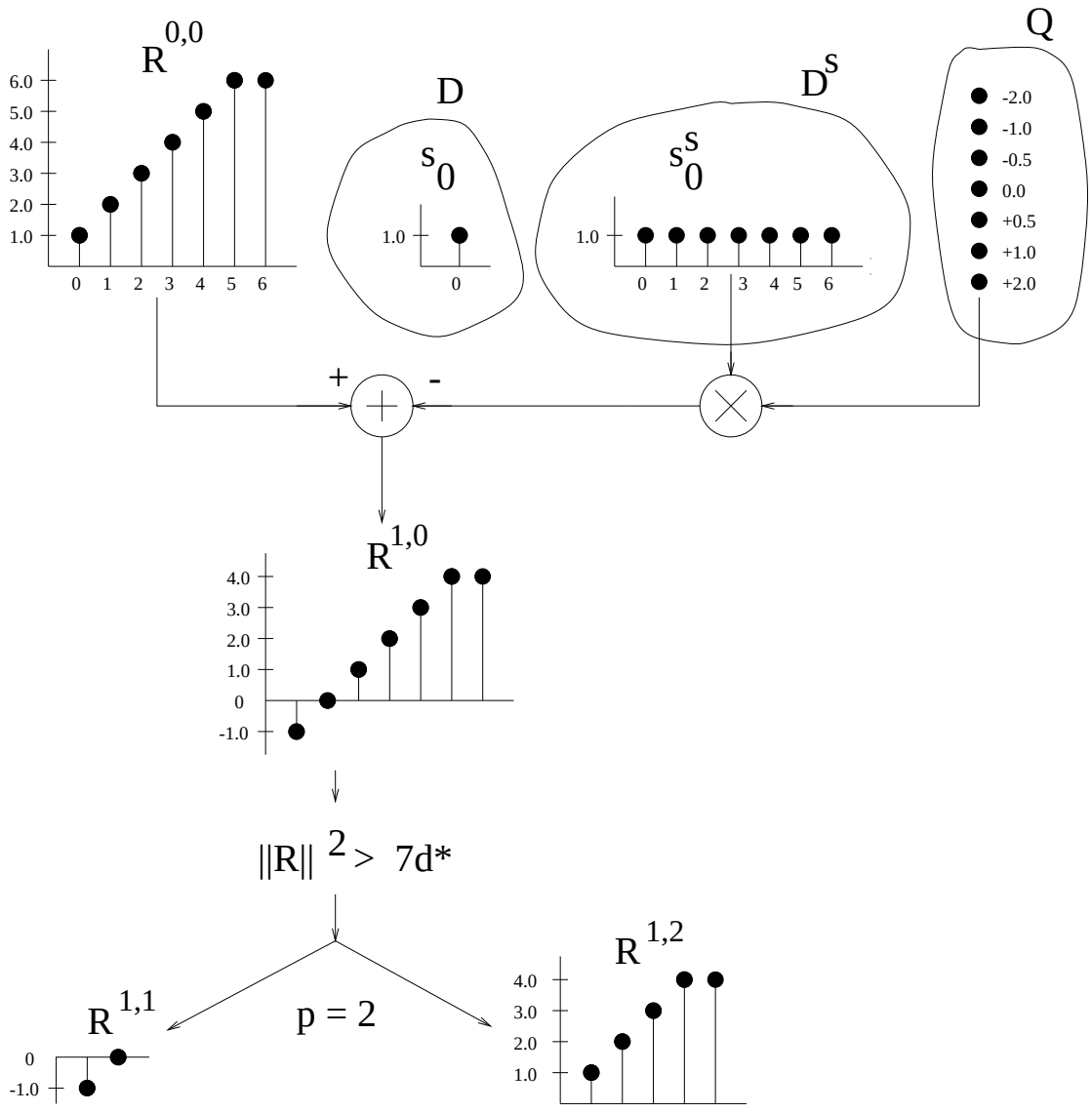


Figura D.5: Segmentation of the first residue.

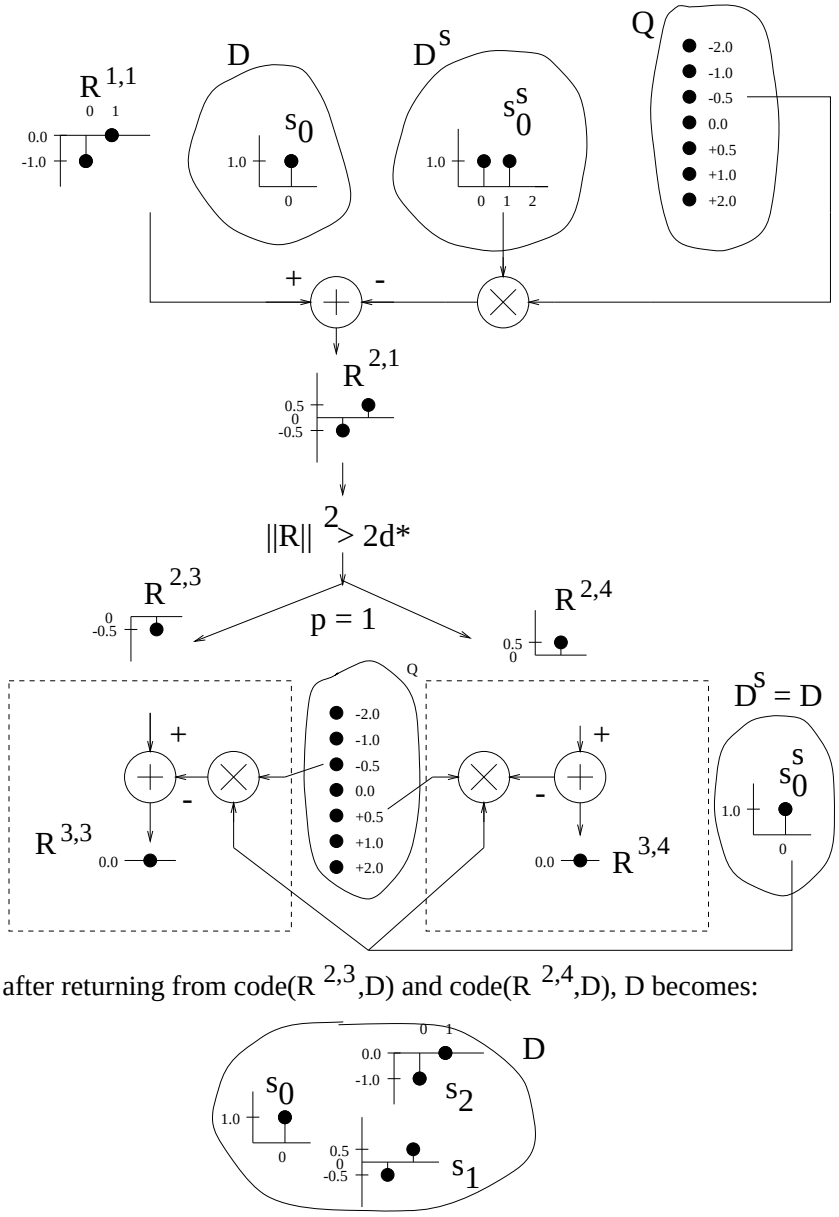


Figura D.6: Coding of the first segment of the first residue.

Then we refine our estimate of $\mathbf{R}^{0,0}$ given by $\hat{\mathbf{R}}^{0,0} = \alpha_l \mathbf{S}_k^s$ to $\hat{\mathbf{R}}^{0,0} \leftarrow \hat{\mathbf{R}}^{0,0} + \hat{\mathbf{R}}^{1,0}$. Finally, the dictionary $\mathcal{D}$ is updated by including $\hat{\mathbf{R}}^{1,0}$ and $\hat{\mathbf{R}}^{0,0}$ in it. Figure D.6 illustrates the recursive application of the procedure to $\hat{\mathbf{R}}^{1,1}$ and figure D.7 illustrates the procedure applied to $\hat{\mathbf{R}}^{1,2}$.

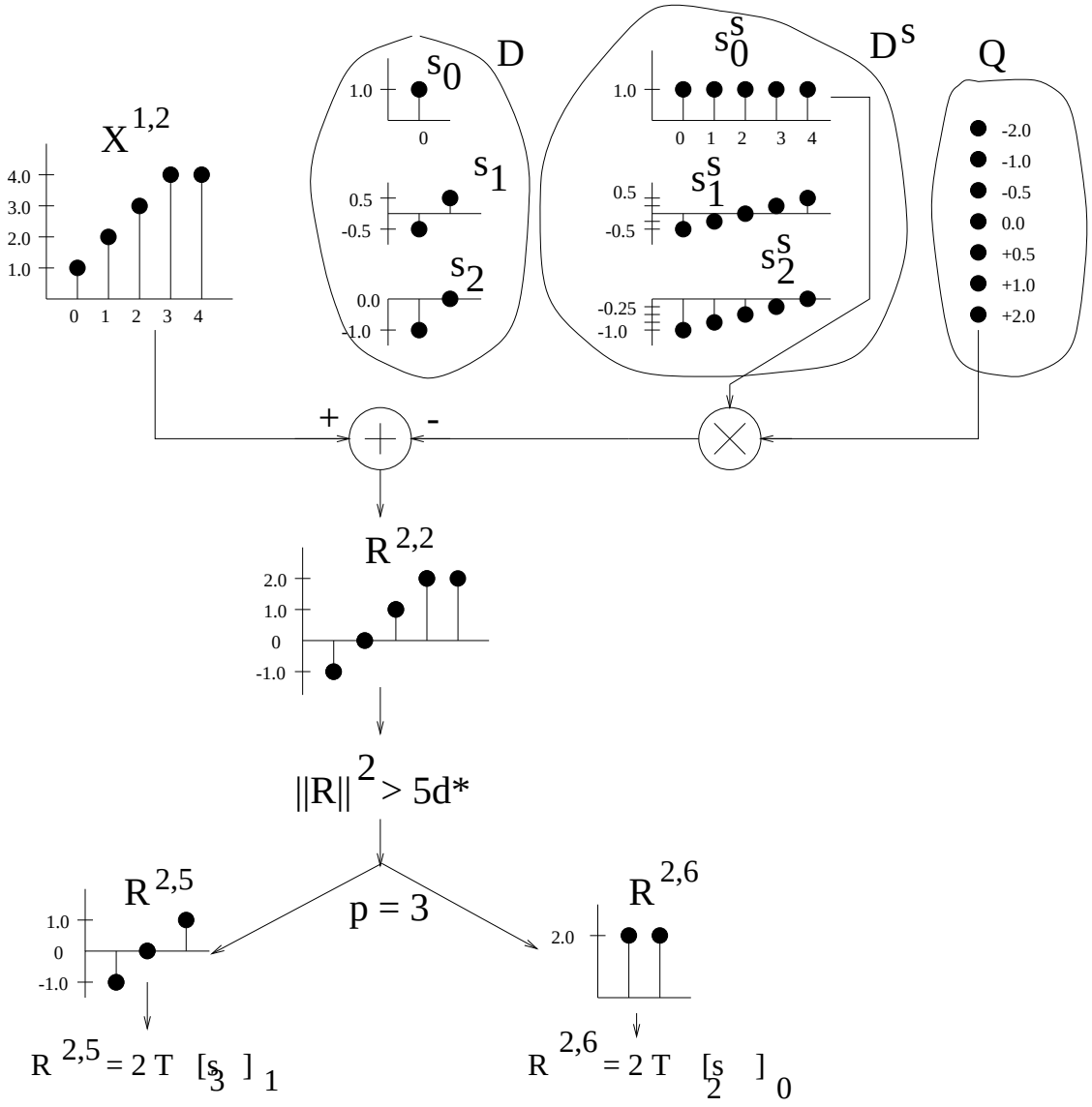


Figura D.7: Coding of the second segment of the first residue.

The complete algorithm is described as follows:

Procedure $\hat{\mathbf{R}}^{i,j} = \text{encode}(\mathbf{R}^{i,j}, \mathcal{D}, d^*)$:

step 1 scale all dictionary vectors to the size n of $\mathbf{R}^{i,j}$,
that is: $\mathbf{S}^s_k = T_n[\mathbf{S}_k]$ for all k.

step 2 find $k_{i,j}$ such that $\frac{\mathbf{S}^T_{k_{i,j}} \mathbf{R}^{i,j}}{\|\mathbf{S}^s_{k_{i,j}}\|}$ is maximum

step 3 find $l_{i,j}$ such that $\|\mathbf{R}^{i,j} - \alpha_{l_{i,j}} \mathbf{S}^s_{k_{i,j}}\|$ is minimum and make $\hat{\mathbf{R}}^{i,j} = \alpha_{l_{i,j}} \mathbf{S}^s_{k_{i,j}}$.

step 4 output $k_{i,j}, l_{i,j}$

step 5 `if` $\|\mathbf{R}^{i,j} - \hat{\mathbf{R}}^{i,j}\|^2 \leq n d^*$ `then` output flag '1' and return $\hat{\mathbf{R}}^{i,j}$.
`else` go to step 6.

step 6 compute the residue $\mathbf{R}^{i+1,j} = \mathbf{R}^{i,j} - \hat{\mathbf{R}}^{i,j}$

step 7 choose $p \in [0, n-2]$ to minimize
$\|\mathbf{R}^{i+1,2j+1} - \hat{\mathbf{R}}^{i+1,2j+1}\|^2 + \|\mathbf{R}^{i+1,2j+2} - \hat{\mathbf{R}}^{i+1,2j+2}\|^2$ and then
split $\mathbf{R}^{i+1,j} = \begin{pmatrix} \mathbf{R}^{i+1,2j+1} & \mathbf{R}^{i+1,2j+2} \end{pmatrix}$

step 8 output flag '0' and p.

step 9 compute $\hat{\mathbf{R}}^{i+1,2j+1} = \text{encode}(\mathbf{R}^{i+1,2j+1}, \mathcal{D}, d^*)$ and
$\hat{\mathbf{R}}^{i+1,2j+2} = \text{encode}(\mathbf{R}^{i+1,2j+2}, \mathcal{D}, d^*)$

step 10 Make $\hat{\mathbf{R}}^{i+1,j} = \begin{pmatrix} \hat{\mathbf{R}}^{i+1,2j+1} & \hat{\mathbf{R}}^{i+1,2j+2} \end{pmatrix}$

step 11 update $\hat{\mathbf{R}}^{i,j} = \alpha_{l_{i,j}} \mathbf{S}^s_{k_{i,j}} + \hat{\mathbf{R}}^{i+1,j}$

step 12 update $\mathcal{D} = \mathcal{D} \cup (\hat{\mathbf{R}}^{i,j}) \cup (\hat{\mathbf{R}}^{i+1,j})$

step 13 return $\hat{\mathbf{R}}^{i,j}$

Decoding is easily accomplished by the procedure below:

Procedure $\hat{\mathbf{R}}^{i,j} = \text{decode}(n, \mathcal{D}, d^*)$:

step 1 scale all dictionary vectors to the size n,
that is: $\mathbf{S}_i^s = T_n[\mathbf{S}_i]$ for all i.

step 2 input $k_{i,j}, l_{i,j}$ and make $\hat{\mathbf{R}}^{i,j} = \alpha_{l_{i,j}} \mathbf{S}^s_{k_{i,j}}$.

step 3 `if` input flag '1' `then` return $\hat{\mathbf{R}}^{i,j}$.
`else` go to step 4.

step 4 input p.

step 5 compute $\hat{\mathbf{R}}^{i+1,2j+1} = \text{decode}(p, \mathcal{D}, d^*)$ and $\hat{\mathbf{R}}^{i+1,2j+2} = \text{decode}(n-p, \mathcal{D}, d^*)$

step 6 make $\hat{\mathbf{R}}^{i+1,j} = \begin{pmatrix} \hat{\mathbf{R}}^{i+1,2j+1} & \hat{\mathbf{R}}^{i+1,2j+2} \end{pmatrix}$

step 7 update $\hat{\mathbf{R}}^{i,j} = \alpha_{l_{i,j}} \mathbf{S}^s_{k_{i,j}} + \hat{\mathbf{R}}^{i+1,j}$

step 8 update $\mathcal{D} = \mathcal{D} \cup (\hat{\mathbf{R}}) \cup (\hat{\mathbf{X}})$

step 9 return $\hat{\mathbf{R}}^{i,j}$

The UMMP algorithm has some interesting features:

i) Due to the use of the multiscale approach, UMMP can perform arbitrarily close to the source's $R(D)$ when $d^* \to \infty$.

ii) Fidelity criteria other than the mean squared error can be easily accommodated in the algorithm. For example, some researchers have pointed out that $\sum_i |X_i - S_i|$ is a better fidelity measure for image processing applications [25].

iii) It is easily extendable to higher dimensions. That is, instead of operating on a vector it can operate on a matrix or multi-dimensional array. For example, if the input is a matrix $\mathbf{R}^{i,j,k}$ of size $N \times M$, UMMP can split the residue in four sub-matrices. The segmentation point in this case is a point $\begin{pmatrix} p_l & p_c \end{pmatrix}$ in the two-dimensional space. The four sub matrices would be: $\mathbf{R}^{i,2j+1,2k+1}$ of size $p_l \times p_c$, $\mathbf{R}^{i,2j+2,2k+1}$ of size $(N - p_l) \times p_c$, $\mathbf{R}^{i,2j+1,2k+2}$ of size $p_l \times (M - p_c)$ and $\mathbf{R}^{i,2j+2,2k+2}$ of size $(N - p_l) \times (M - p_c)$

iv) It is fully adaptive and therefore universal at least in the sense that no prior knowledge of a source model is needed for it to perform best.

v) It merges the transformation, the quantization and the entropy coding steps of the three-step approach in one. The transformation is performed because the input data is represented as a weighted sum of vectors in the dictionary. The vectors in the dictionary play the role of basis vectors of the transformation. The quantization comes naturally since the matching operations are only approximate and controlled by a fidelity measure. The entropy coding is performed as the dictionary is built to fit the input data statistics, like in the Lempel-Ziv algorithm. All three steps are merged by the dictionary operation and cannot be separated.

We also observed , after some computer simulations, the following properties:

1. If we are using $\log_2(N)$ bits (using for example an arithmetic coder with uniform N-level distribution) to encode p, we found out that it is better, in a rate-distortion performance sense, to use a fixed segmentation point $p = N/2$, and encode it with 0 bits. Unless a more complex integer code is used for p, the gain we get with the better partition is not worth the increase in bit rate.

2. Better performance is achieved if the quantizer has one reconstruction level, $\mathcal{Q} = \{1\}$. We observed that when the number of quantization levels increases, the number of segmentations (and therefore the number of different vectors learned by the algorithm) decreases and vice-versa. In fact, the algorithm learns the same basis vectors with several different amplitudes when the number of quantization levels is small. Therefore, unless a complex joint entropy coding method is used to encode both the dictionary indexes and the quantizer indexes, it is better to use a one-level quantizer and allow the dictionary to explore the joint statistics.

D.2 UMMP performance for $d^* = 0$ and $d^* \to \infty$.

When the vector X is a sample of a discrete memoryless vector in an alphabet $\mathcal{A} = \{a_u\}_{u=0}^{U-1}$, the algorithm should be able to compress it with zero distortion.

Without loss of generality, we will consider the alphabet $\mathcal{A}$ a subset of the integers $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$. If the initial dictionary is not empty, UMMP expands X in a frame that contains at least impulses centered at every component of X. This can be easily seen as, in the worst case, the vector can be repeatedly parsed in segments of length one. Therefore, for lossless capability, we must have at least a combination of $\alpha_l T_l[\mathbf{S}_k]$ equal to each alphabet element a_u. For example, $\mathcal{D}_0 = \{1\}$, $\mathcal{Q} = \mathcal{A}$. However, this is not a sufficient condition for lossless UMMP because the components of the residue vector can have values outside the set $\mathcal{A}$. We will call $\mathcal{V} = \{v_q\}$ the set of values that a component of the residue vector can take. Since we considered $\mathcal{A}$ a subset of the integers, $\mathcal{V}$ will also be a subset of the integers if we restrict the scale transformation to be a map of integers, that is $T_N^M : \mathbb{Z}^M \to \mathbb{Z}^N$. UMMP can have lossless capability if $\mathcal{V}$ has finite size. This will always be the case for finite blocklength N, but $|\mathcal{V}|$ can become too large if the norm of the residue grows unbounded. Therefore, it is important that the residue's norm decays at each iteration, specially for large N. We can guarantee that it is at least non increasing by including the zero vector in the dictionary. So either $0 \in \mathcal{D}_0$ or $0 \in \mathcal{Q}$ are necessary conditions. Considering the zero in the dictionary, the norm of the residue is bounded by $\|\mathbf{R}\| \leq \|\mathbf{X}\|$. In the worst case, $\mathbf{R}$ has just one non-zero component which concentrates all the error energy, that is $R_n = \|\mathbf{X}\|$ for some $n \in [0, N-1]$. Therefore, for lossless capability, UMMP must have a combination of $\alpha_l T_l[\mathbf{S}_k]$ equal to any value in the range $[-\|\mathbf{X}\|, \|\mathbf{X}\|]$ for all $\mathbf{X}$. We then have the following conditions for good convergence properties:

Theorem D.2.1 *UMMP can lossless compress data for a vector source* $\mathbf{x}$ *if:*

i) for all $v_q \in \mathcal{V} = \{-\sqrt{N}a_{max}, -\sqrt{N}a_{max} + 1, \ldots, \sqrt{N}a_{max}\}$ *there are* i, k *such that* $v_q = \alpha_l T_l[\mathbf{S}_k], \mathbf{S}_k \in \mathcal{D}_0$, *where* $a_{max} = \max(|\sup\{\mathcal{A}\}|, |\inf\{\mathcal{A}\}|)$.

ii) $\mathbf{0} \in \mathcal{D}_0$ *or* $0 \in \mathcal{Q}$

The performance of the UMMP at the low rate range is bounded in the following theorem:

Theorem D.2.2 *UMMP can lossy compress data from a discrete stationary source with performance within any* $\Delta > 0$ *to it's* $D(R)$ *function as* $R \to 0$ *and* $N \to \infty$.

At zero rate, any vector $\mathbf{X}$ output from a source $\mathbf{x}$ can be compressed with average per-symbol distortion $d_{max} = \|E\{\mathbf{x} - E\{\mathbf{x}\}\}\|^2/N$, where $E\{\cdot\}$ denotes expectation. If the source is stationary, $E\{\mathbf{x}\}$ is a constant vector where all components are the same. We can always choose i, j such that $\|E\{\mathbf{x}\} - \alpha_i T_N[\mathbf{S}_j]\|^2 < N\delta^2$ provided that the UMMP algorithm starts with a suitable combination of scalar quantizer and initial dictionary. For example, we can use a one-level quantizer $\mathcal{Q} = \{1\}$, and an initial dictionary of scalars uniformly distributed $\mathcal{D}_0 = \{\ldots, -4\delta, -2\delta, 0, 2\delta, 4\delta, \ldots\}$ with step size 2δ (we are considering a scale transformation $T_N[a]$ that maps a scalar a in a vector $a\mathbf{1}_N$ of size N where each component equals a). If the source is also discrete with finite alphabet $\mathcal{A}$, the number of levels of this quantizer is $K = (\sup\{\mathcal{A}\} - \inf\{\mathcal{A}\})/2\delta$. Therefore, when the target distortion $d^* \geq d_{max}$, UMMP can represent any possible output $\mathbf{X}$ of a discrete vector source $\mathbf{x}$ in a finite alphabet with average per-symbol distortion less than $d_{max} + \Delta$, $\Delta = 2\delta\sqrt{d_{max}} + \delta^2$, using $\lceil \log_2(K) \rceil$ bits, where $\lceil y \rceil$ denotes the smaller integer greater or equal to y. As the vector's size $N \to \infty$, the per-symbol rate goes to zero. We conclude that for large blocklengths, we can encode with distortion $d_{max} + \Delta$ for any $\Delta > 0$ and rate arbitrarily close to zero when $d^* \geq d_{max}$.

D.3 Experimental results

We have made computer simulations of UMMP. We wanted to evaluate the rate-distortion performance of UMMP when applied to two-dimensional data, specifically gray-scale digital images. So we implemented the two-dimensional version of UMMP (2D-UMMP). The 2D-UMMP operates on two-dimensional arrays, or matrices, instead of vectors. The basic difference to the one-dimensional algorithm is that the segmentation point $\mathbf{p} = \begin{pmatrix} p_l & p_c \end{pmatrix}^T$ is a point in a two-dimensional space. We use as the squared norm $\|\mathbf{X}\|^2$ of a matrix $\mathbf{X}$ the sum of its squared elements. If we allow generic segmentation points, a residue matrix $\mathbf{R}^{r,j,q}$ is segmented in four sub-matrixes as:

$$\mathbf{R}^{r,j,q} = \begin{pmatrix} \mathbf{R}^{r,2j+1,2q+1} & \mathbf{R}^{r,2j+1,2q+2} \\ \mathbf{R}^{r,2j+2,2q+1} & \mathbf{R}^{r,2j+2,2q+2} \end{pmatrix} \tag{D.1}$$

The sub-matrices in equation D.1 are of sizes $p_l \times p_c$, $p_l \times (M - p_c)$,

$(N - p_l) \times p_c$ and $(N - p_l) \times (M - p_c)$. As in the one-dimensional case however, we get better performance if we use a fixed partition rule that splits the residue matrix in two. The rule we adopted is:

- `if` $\mathbf{R}$ is $N \times M$ where $N > M$, `then` $p_l = N/2$ and $p_c = 0$.
- `else` $p_l = 0$ and $p_c = M/2$.

We have implemented 2D-UMMP with this fixed segmentation rule. This implementation also used a one level scalar quantizer $\mathcal{Q} = \{1\}$ because, as in the one-dimensional case, this leads to better performance. We have limited the matrix size to powers of two, that is, $\mathbf{X}$ must be of size $(N \times M) = (2^K \times 2^L)$ with K, L integers. To use the segmentation rule described above, we need $K+L+1$ dictionaries at the scales $(1 \times 1), (2 \times 1), (2 \times 2), \ldots, (N \times M)$. We denote each dictionary $\mathcal{D}^{(k,l)}$, meaning that it contains matrices of size (scale) $2^k \times 2^l$. The 2D-UMMP is described next:

Let:

- $\mathbf{X}$ be an $(N \times M) = (2^K \times 2^L)$ matrix.
- $\mathcal{D} = \{\mathbf{S}_0, \ldots, \mathbf{S}_{I-1}\}$ a dictionary with I matrices of arbitrary sizes. This dictionary must be initialized to $\mathcal{D} = \mathcal{D}_0$.
- d^* be a target distortion.
- $T_{N,M}[\mathbf{X}]$ be a scale transformation that maps the matrix $\mathbf{X}$ in a matrix of size $N \times M$.
- $\mathbf{R}^{r,j}$ be a residue matrix, where the superscript r corresponds to the number of projections in the dictionary already made and the superscript j indexes the nodes of the associated segmentation tree. $\mathbf{R}^{0,0}$ is initially set to $\mathbf{X}$.

Procedure $\hat{\mathbf{R}}^{r,j} = \text{encode}(\mathbf{R}^{r,j}, d^*)$:

step 1 scale all matrices in the dictionary to the size $2^k \times 2^l$ of $\mathbf{R}^{r,j}$, that is make $\mathbf{S}_i^s = T_{2^k,2^l}[\mathbf{S}_i]$ for all i.

step 2 find index $i_{r,j}$ in the dictionary such that $\|\mathbf{R}^{r,j} - \mathbf{S}^s_{i_{r,j}}\|$ is minimum and make $\hat{\mathbf{R}}^{r,j} = \mathbf{S}^s_{i_{r,j}}$.

step 3 output index $i_{r,j}$

step 4 `if` $k = l = 0$ return $\hat{\mathbf{R}}^{r,j}$.
`else` go to step 4.

step 5 `if` $\|\mathbf{R}^{r,j} - \hat{\mathbf{R}}^{r,j}\|^2 \le 2^{k+l} d^*$ `then` output flag '1' and return $\hat{\mathbf{R}}^{r,j}$.
`else` go to step 5.

step 6 make $\mathbf{R}^{r+1,j} = \mathbf{R}^{r,j} - \hat{\mathbf{R}}^{r,j}$ and output flag '0'.

step 7 `if` $k > l$ split $\mathbf{R}^{r+1,j} = \begin{pmatrix} \mathbf{R}^{r+1,2j+1} \\ \mathbf{R}^{r+1,2j+2} \end{pmatrix}$,
where $\mathbf{R}^{r+1,2j+1}$ and $\mathbf{R}^{r+1,2j+2}$ are $2^{k-1} \times 2^l$
`else` split $\mathbf{R}^{r+1,j} = \begin{pmatrix} \mathbf{R}^{r+1,2j+1} & \mathbf{R}^{r+1,2j+2} \end{pmatrix}$
where $\mathbf{R}^{r+1,2j+1}$ and $\mathbf{R}^{r+1,2j+2}$ are $2^k \times 2^{l-1}$

step 8 compute $\hat{\mathbf{R}}^{r+1,2j+1} = \text{encode}(\mathbf{R}^{r+1,2j+1}, d^*)$

step 9 compute $\hat{\mathbf{R}}^{r+1,2j+2} = \text{encode}(\mathbf{R}^{r+1,2j+2}, d^*)$

step 10 `if` $k > l$ make $\hat{\mathbf{R}}^{r+1,j} = \begin{pmatrix} \hat{\mathbf{R}}^{r+1,2j+1} \\ \hat{\mathbf{R}}^{r+1,2j+2} \end{pmatrix}$,
`else` make $\hat{\mathbf{R}}^{r+1,j} = \begin{pmatrix} \hat{\mathbf{R}}^{r+1,2j+1} & \hat{\mathbf{R}}^{r+1,2j+2} \end{pmatrix}$

step 11 make $\hat{\mathbf{R}}^{r,j} = \hat{\mathbf{R}}^{r,j} + \hat{\mathbf{R}}^{r+1,j}$.

step 12 $\mathcal{D} = \mathcal{D} \cup \left\{\hat{\mathbf{R}}^{r,j}\right\} \cup \left\{\hat{\mathbf{R}}^{r+1,j}\right\}$.

step 13 return $\hat{\mathbf{R}}^{r,j}$

We have also implemented a 2D-UMMP with arbitrary $\mathbf{p}$ and a multi-level scalar quantizer (2D-UMMPG). The 2D-UMMPG is described next:

Let:

- $\mathbf{X}$ be an $(N \times M) = (2^K \times 2^L)$ matrix.
- $\mathcal{D} = \{\mathbf{S}_0, \ldots, \mathbf{S}_{I-1}\}$ a dictionary with I matrices of arbitrary sizes. This dictionary must be initialized to $\mathcal{D} = \mathcal{D}_0$.
- d^* be a target distortion.
- $d(\mathbf{X}, \mathbf{Y})$ be the distortion when $\mathbf{Y}$ replaces $\mathbf{X}$.
- $T_{N,M}[\mathbf{X}]$ be a scale transformation that maps the matrix $\mathbf{X}$ to a matrix of size $N \times M$.
- $\mathbf{R}^{r,j,q}$ be a residue matrix, where the superscript r corresponds to the number of projections in the dictionary already made and the superscripts j, q index the nodes of the associated segmentation trees. $\mathbf{R}^{0,0,0}$ is initially set to $\mathbf{X}$.
- $J(\mathbf{R}^{r,j,q}, p_l, p_c, \mathcal{D}), p_l \in [0, N), p_c \in [0, M)$ be an objective function.
- $\mathcal{Q}(a), a \in \mathbb{R}$ be a scalar quantizer.

The procedure $\hat{\mathbf{R}}^{r,j,q} = \text{code}(\mathbf{X}^{r,j,q}, \mathcal{D}, d^*)$ encodes $\mathbf{R}^{r,j,q}$ such that $d(\mathbf{R}^{r,j,q}, \hat{\mathbf{R}}^{r,j,q}) \leq d^*$ and is defined as

Procedure $\hat{\mathbf{R}}^{r,j,q} = \text{code}(\mathbf{R}^{r,j,q}, \mathcal{D}, d^*)$:

step 1 find $i_{r,j,q}$ such that $d(\mathbf{R}^{r,j,q}, \hat{\mathbf{R}}^{r,j,q})$ is minimized, where

`if` $||T_{n,m}(\mathbf{S}_{i_{r,j,q}})||^2 > 0$

`then` $\hat{\mathbf{R}}^{r,j,q} = \mathcal{Q}\left(< \mathbf{R}^{r,j,q}, \frac{T_{n,m}(\mathbf{S}_{i_{r,j,q}})}{||T_{n,m}(\mathbf{S}_{i_{r,j,q}})||^2} > \right) T_{n,m}(\mathbf{S}_{i_{r,j,q}})$

`else` $\hat{\mathbf{R}}^{r,j,q} = \mathbf{0}$

step 2 output $i_{r,j,q}$ and $\mathcal{Q}\left(< \mathbf{R}^{r,j,q}, \frac{T_{n,m}(\mathbf{S}_{i_{r,j,q}})}{||T_{n,m}(\mathbf{S}_{i_{r,j,q}})||^2} > \right)$

step 3 `if` $d(\mathbf{R}^{r,j,q}, \hat{\mathbf{R}}^{r,j,q}) \leq d^*$ `then` output flag '1' and return $\hat{\mathbf{R}}^{r,j,q}$.

`else` go to step 4.

step 4 compute the residue $\mathbf{R}^{r+1,j,q} = \mathbf{R}^{r,j,q} - \hat{\mathbf{R}}^{r,j,q}$

step 5 choose $p_l \in [0, M-1], p_c \in [0, N-1]$ to minimize $J(\mathbf{R}^{r+1,j,q}, p_l, p_c, \mathcal{D})$ and then

split $\mathbf{R}^{r+1,j,q} = \begin{pmatrix} \mathbf{R}^{r+1,2j+1,2q+1} & \mathbf{R}^{2j+1,2q+2} \\ \mathbf{R}^{r+1,2j+2,2q+1} & \mathbf{R}^{r+1,2j+2,2q+2} \end{pmatrix}$

where $\mathbf{R}^{r+1,2j+1,2q+1}$ is $p_l \times p_c$

step 6 output flag '0', p_l, p_c

step 7 compute $\hat{\mathbf{R}}^{r+1,2j+p,2q+t}$ =code($\mathbf{R}^{r+1,2j+p,2q+t}, \mathcal{D}, d^*$), $l, t \in \{0, 1\}$

step 8 make $\hat{\mathbf{R}}^{r+1,j,q} = \begin{pmatrix} \hat{\mathbf{R}}^{r+1,2j+1,2q+1} & \hat{\mathbf{R}}^{2j+1,2q+2} \\ \hat{\mathbf{R}}^{r+1,2j+2,2q+1} & \hat{\mathbf{R}}^{r+1,2j+2,2q+2} \end{pmatrix}$, $\hat{\mathbf{R}}^{r,j,q} = \hat{\mathbf{R}}^{r,j,q} + \hat{\mathbf{R}}^{r+1,j,q}$

step 9 $\mathcal{D} = \mathcal{D} \cup (\hat{\mathbf{R}}^{r,j,q}) \cup (\hat{\mathbf{R}}^{r+1,j,q})$

step 10 return $\hat{\mathbf{R}}^{r,j,q}$

We used the computer programs to lossy compress the image LENA 256×256. The image was initially divided in 8×8 blocks that were processed in sequence by the algorithm. Figure D.8 shows the results for 2D-UMMP and 2D-UMMPG. The size of the dictionaries were limited by the use of a pruning strategy. Whenever a new element is included in the dictionary, the least used element of that dictionary is discarded. In 2D-MMP and 2D-UMMP the maximum size of the dictionaries was 8192. The initial dictionary for 2D-UMMP was $\mathcal{D}_0 = \{-128, -124, \ldots, 0, 4, \ldots, 128\}$. The two-dimensional scale transformation $T_{N.M}[\mathbf{X}]$ was implemented as follows: Firstly, a one-dimensional scale transformation $T_M[\cdot]$ is applied to all the rows of $\mathbf{X}$. Then we apply another one-dimensional transformation, $T_N[\cdot]$ to the M columns of the intermediate result.

The one-dimensional scale transformation was implemented using classical sampling-rate change operations. When we want to change from length N_0 to length N, with $N > N_0$, we first change to length $N_0 N$ using a linear interpolator as the filter. Then we make a downsampling by N_0 operation. The whole procedure is defined in equation D.2.

$$
\begin{aligned}
m_n^0 &= \left\lfloor \frac{n\,(N_0-1)}{N} \right\rfloor \\
m_n^1 &= \begin{cases} m_n^0+1, & m_n^0 < N_0-1 \\ m_n^0, & m_n^0 = N_0-1 \end{cases} \\
\alpha_n &= n\,(N_0-1) - N m_n^0 \\
S_n^s &= \left\lfloor \frac{\alpha_n \left(S_{m_n^1} - S_{m_n^0}\right)}{N} \right\rfloor + S_{m_n^0}, n = 0,1,\ldots,N-1
\end{aligned} \tag{D.2}
$$

When $N_0 > N$, we first change to length $N_0 N$ using a linear interpolator as the filter an then we apply a mean filter of length $N_0 + 1$. Then we make a downsampling by N_0 operation. The procedure is defined in equation D.3.

$$
\begin{aligned}
m_{n,k}^0 &= \left\lfloor \frac{n\,(N_0-1)+k}{N} \right\rfloor \\
m_{n,k}^1 &= \begin{cases} m_{n,k}^0+1, & m_{n,k}^0 < N_0-1 \\ m_{n,k}^0, & m_{n,k}^0 = N_0-1 \end{cases} \\
\alpha_n &= n\,(N_0-1)+k - N m_n^0 \\
S_n^s &= S_{m_{n,k}^0} + \frac{1}{N_0+1} \sum_{k=0}^{N_0} \left\lfloor \frac{\alpha_{n,k} \left(S_{m_{n,k}^1} - S_{m_{n,k}^0}\right)}{N} \right\rfloor, n = 0,\ldots,N-1
\end{aligned} \tag{D.3}
$$

The two-dimensional scale transformation was implemented as in equation D.4:

$$
\begin{aligned}
\mathbf{Y}_i &= T_M[\mathbf{S}_i], i = 0,1,\ldots,\ell\,(\mathbf{S}_i)-1 \\
\mathbf{S}_j^s &= \left(T_N\left[\left(\mathbf{Y}^T\right)_j\right]\right)^T, j = 0,1,\ldots,M-1
\end{aligned} \tag{D.4}
$$

where we use $\mathbf{X}_i$ to denote the rows of $\mathbf{X}$.

For the 2D-UMMPG, we used an initial dictionary $\mathcal{D}_0 = \{1\}$, the scalar quantizer $\mathcal{Q} = \{-128,-120,\ldots,0,4,\ldots,+128\}$, and maximum dictionary size of 256. It should be noted that the 2D-UMMP and 2D-UMMPG have dictionaries of comparable size, since for 2D-UMMPG there are $32 \times 256 = 8192$ combinations of amplitudes and shapes. The distortion measure used with 2D-UMMPG was the mean squared error, given by:

$$d(\mathbf{X},\mathbf{Y}) = \frac{1}{NM}\sum_{n=0}^{N-1}\sum_{m=0}^{M-1}(X_{n,m}-Y_{n,m})^2 \tag{D.5}$$

The objective function was defined as:

$$J(\mathbf{R}^{r,j,q}, p_l, p_c, \mathcal{D}) = \min_{p_l}\min_{p_c}\sum_{u=0}^{1}\sum_{t=0}^{1}\min_{i_{u,t}} d\left(\mathbf{R}^{r,2j+u,2q+t}, \alpha_{i_{u,t}} T_{n,m}(\mathbf{S}_{i_{u,t}})\right),$$

$$\alpha_{i_{u,t}} = \begin{cases} \mathcal{Q}\left(< \mathbf{R}^{r,2j+u,2q+t}, \frac{T_{n,m}(\mathbf{S}_{i_{u,t}})}{\|T_{n,m}(\mathbf{S}_{i_{u,t}})\|^2} >\right) & \|T_{n,m}(\mathbf{S}_{i_{u,t}})\| > 0 \\ 0 & \|T_{n,m}(\mathbf{S}_{i_{u,t}})\| = 0 \end{cases} \tag{D.6}$$

With this objective function, 2D-UMMPG attempts to find the segmentation point that minimizes the sum of the distortions on the reproduction of each submatrix using the current dictionary.

The integers output by the algorithms were encoded using an adaptive arithmetic coder with different models for the dictionary indexes at each scale. The flags were also encoded using the adaptive arithmetic coder with independent models for each flag.

Figure D.9 shows the performance of 2D-UMMP and 2D-UMMPG with a Gaussian memoryless source. The parameters used in the algorithms were the same as in the case of the image LENA. The source was a 256×256 matrix of samples of a memoryless Gaussian process with mean 128.0 and variance 960.0. The optimum theoretical $R(D)$ is also shown.

We can see from figures D.8 and D.9 that the performance of UMMP with fixed segmentation and one-level scalar quantizer is far better than that of 2D-UMMPG. The advantage over 2D-UMMPG can be understood if we consider that a full-search VQ will generally perform better than a gain-shape vector quantizer of comparable size, unless a very complex joint entropy coding method is used to encode the indexes and the gain. Also, for segments not greater than 8×8 blocks as used in the experiment, it may be that the most probable segmentation point is $p = \ell(\mathbf{R}^{i,j})/2$, so that is little to gain if we use a generic p. The computational effort required to evaluate the best p in the generic algorithm is also very high. Therefore we will focus on the version with fixed segmentation point and one-level scalar quantizer in the remaining of this work.

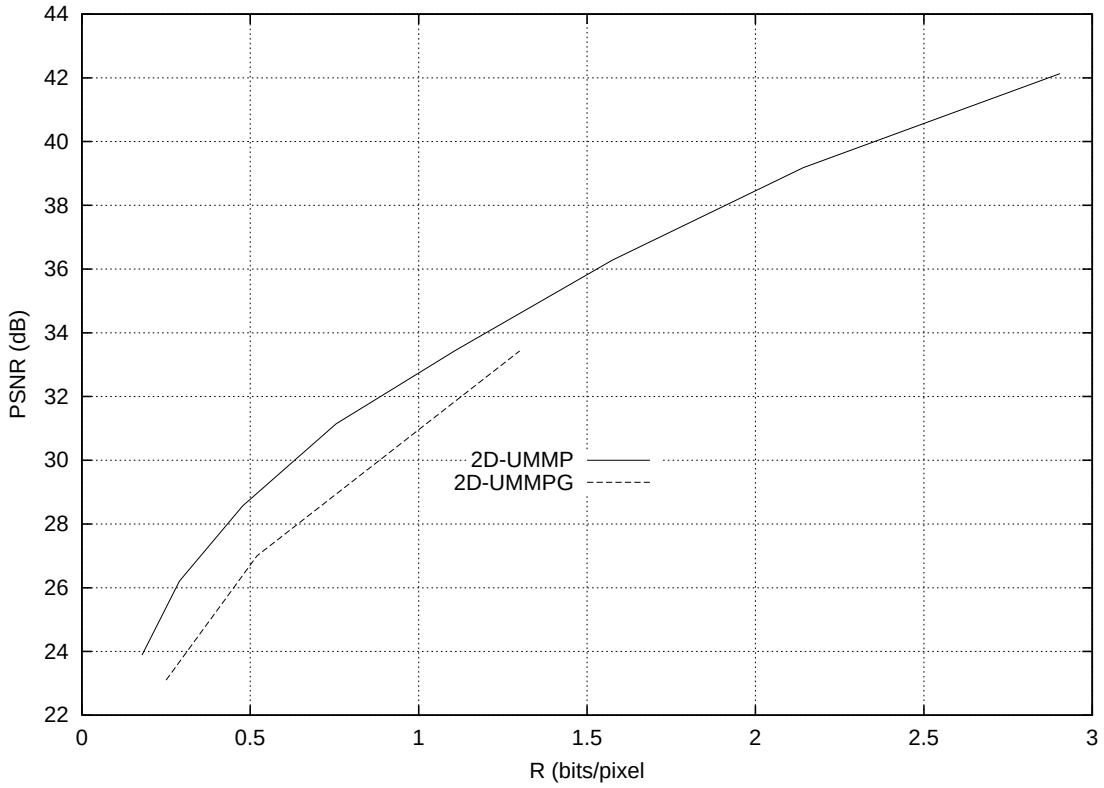


Figura D.8: Rate × distortion for 2D-UMMP and 2D-UMMPG with LENA 256 × 256.

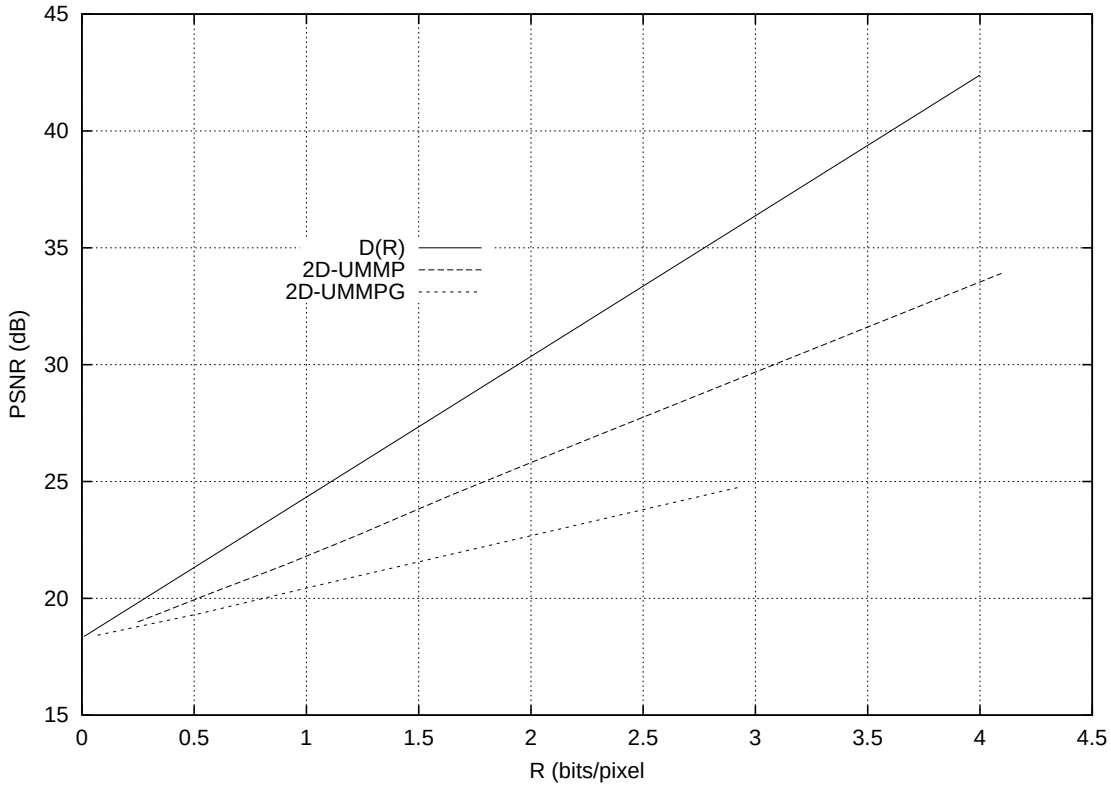


Figura D.9: R(D) curve for 2D-UMMP and 2D-UMMPG with Gaussian memoryless source.

In the next appendix we introduce MMP, a variation of UMMP that is more efficient in a rate-distortion sense. There we also present results for UMMP with a variety of other sources and the performances of both UMMP and MMP are compared to those of other algorithms.

Apêndice E

MMP

In this appendix we propose a compression algorithm that we call MMP. Similarly to the UMMP algorithm described on section D, it is based on *approximate matching with scales of recurrent patterns*. It also adaptively builds a dictionary $\mathcal{D}$ while encoding the data. As in UMMP, the dictionary is regularly updated with concatenations of *dilated/contracted* versions of previously occurred patterns. The basic difference between UMMP and MMP is that UMMP tries to reconstruct a segment of the input vector using a *linear combination* of the elements in the dictionary while MMP uses a *single element* of the dictionary to represent that segment.

E.1 MMP description

The second type of algorithm we study in this work is the MMP (Multi-dimensional Multiscale Parser). It is also based on approximate matching with scales of recurrent patterns but differs from UMMP because it doesn't evaluates residues vectors. If we describe MMP using the UMMP framework, we can say that in MMP the residue is always set to $\mathbf{R}^{i+1,j} = \mathbf{R}^{i,j}$. In other words, when we find a match with distortion above the target d^*, we simply discard that match and split the input vector. Therefore, MMP doesn't encode the dictionary index until a match with distortion below the target is found. Setting the residue equal to the input vector is equivalent to choose, in UMMP, the all-zeroes vector to represent the input block whenever we don't have a match. Therefore, the MMP algorithm does

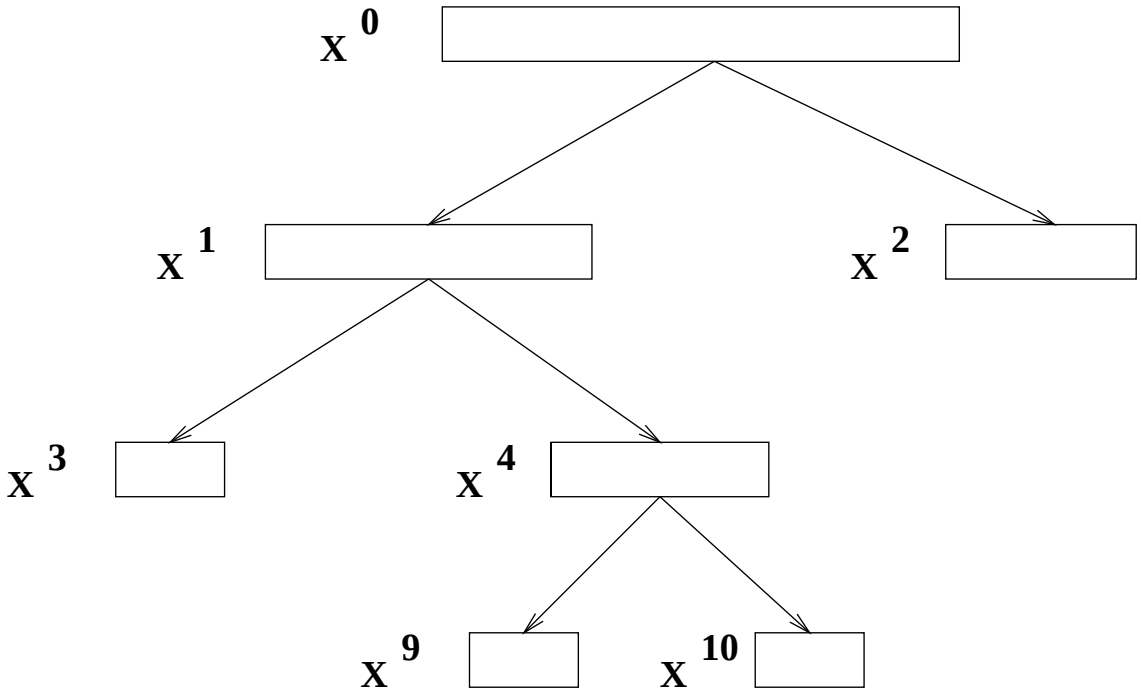


Figura E.1: Segmentation tree on MMP.

not use the dictionary as a basis to expand the input data, since only one element of the dictionary will be used to represent an input block.

In MMP, we attempt to approximate the input vector $\mathbf{X}$ using the best dictionary element as $\hat{\mathbf{X}} =< T_N[\mathbf{S}_{i_0}], \mathbf{X} > T_N[\mathbf{S}_{i_0}]$. If the approximation fails, that is, the distortion is above the target, we split $\mathbf{X}^0 = \mathbf{X}$ in two segments, $\mathbf{X}^1$ and $\mathbf{X}^2$, an then we try to approximate each one using scaled versions of the vectors in the dictionary. If the attempt to represent any of the segments using the current dictionary fails, that segment will be split in two and the procedure repeated. Therefore, similarly to the UMMP, in MMP there is a segmentation tree associated to a given input vector $\mathbf{X}$ and a given target distortion d^*. But since MMP does not evaluate residue vectors, the segmentation tree is a little different, as shown in figure E.1.

The segmentation tree in figure E.1 corresponds to the segmentation of the input vector $\mathbf{X}$ in four segments, $\left(\mathbf{X}^3 \quad \mathbf{X}^9 \quad \mathbf{X}^{10} \quad \mathbf{X}^2 \right)$. In this example, each segment will be approximated by scaled versions of vectors in the dictionary, leading to the representation $\hat{\mathbf{X}} = \left(\hat{\mathbf{X}}^3 \quad \hat{\mathbf{X}}^9 \quad \hat{\mathbf{X}}^{10} \quad \hat{\mathbf{X}}^2 \right)$. A node j of the segmentation tree has either two children, numbered $2j+1$ and $2j+2$, or no child at all. A node that has no child is a leaf node. In MMP only the leaf nodes of the segmentation tree are associated to vectors in the dictionary and are used to approximate the input

vector. MMP attempts to find a representation of the input vector with distortion at most Nd^*. In order to do that, MMP repeatedly parses the input until it can successfully approximate a segment (that is, the distortion is at most d^* times the length of the segment). Therefore, the distortion on the approximation at a leaf node j, given by $\|\mathbf{X}^j - < \mathbf{S}^s_{i_j}, \mathbf{X}^j > \mathbf{S}^s_{i_j}\|$, can't be greater than the target distortion d^* times the length of the segment, otherwise the segmentation would continue and the node wouldn't be a leaf.

The dictionary in MMP can be updated in the same fashion as in the UMMP: Whenever the approximations $\hat{\mathbf{X}}^{2j+1}$ and $\hat{\mathbf{X}}^{2j+2}$, associated to the children nodes $2j+1$ and $2j+2$, are available, MMP forms an estimate to segment $\hat{\mathbf{X}}^j$, associated to the parent node j, as the concatenation of the approximations to the segments associated to the two children nodes. In the example of figure E.1, when $\hat{\mathbf{X}}^9$ and $\hat{\mathbf{X}}^{10}$ are available we can concatenate them to get a new approximation $\hat{\mathbf{X}}^4$. This new approximation can then be included in the dictionary.

In both MMP and UMMP we need to output to the decoder the information regarding the segmentation tree and the segmentation points. But in MMP we only need to output the dictionary indexes and the inner products corresponding to the approximations *at the leaf nodes*. This is different from UMMP, where we need information of the best approximations made *at all nodes*. A complete algorithm to implement MMP is described next:

MMP operates on a vector $\mathbf{X} = (X_0 \dots X_{N-1})^T$ output by a vector source $\mathbf{x} = (x_0 \dots x_{N-1})^T$. It uses:

1) A dictionary $\mathcal{D}$ of vectors, initially set to $\mathcal{D}_0 = \{\mathbf{S}_0, \dots, \mathbf{S}_{M-1}\}$,

2) A scalar quantizer defined by the set of reproducing values $\mathcal{Q} = \{\alpha_0, \dots, \alpha_{K-1}\}$ and

3) A target distortion d^*.

The MMP is described as follows:

Procedure $\hat{\mathbf{X}}^j = \text{encode}(\mathbf{X}^j, \mathcal{D}, d^*)$:

step 1 scale all dictionary vectors to the size N of $\mathbf{X}^j$,
that is: $\mathbf{S}_i^s = T_n[\mathbf{S}_i]$ for all i.

step 2 find i_j, k_j such that $\|\mathbf{X}^j - \alpha_{k_j}\mathbf{S}_{i_j}^s\|$ is minimum and make $\hat{\mathbf{X}}^j = \alpha_{k_j}\mathbf{S}_{i_j}^s$.

step 3 `if` $\|\mathbf{X}^j - \hat{\mathbf{X}}^j\|^2 \leq n d^*$ `then` output flag '1', output indexes i_j, k_j and return $\hat{\mathbf{X}}^j$.
`else` go to step 4.

step 4 choose $p \in [0, N-2]$ to minimize $\|\mathbf{X}^{2j+1} - \hat{\mathbf{X}}^{2j+1}\|^2 + \|\mathbf{X}^{2j+2} - \hat{\mathbf{X}}^{2j+2}\|^2$ and then
split $\mathbf{X}^j = \begin{pmatrix} \mathbf{X}^{2j+1} & \mathbf{X}^{2j+2} \end{pmatrix}$

step 5 output flag '0' and p.

step 6 Compute $\hat{\mathbf{X}}^{2j+1} = \text{encode}(\mathbf{X}^{2j+1}, \mathcal{D}, d^*)$ and $\hat{\mathbf{X}}^{2j+2} = \text{encode}(\mathbf{X}^{2j+2}, \mathcal{D}, d^*)$

step 7 make $\hat{\mathbf{X}}^j = \begin{pmatrix} \hat{\mathbf{X}}^{2j+1} & \hat{\mathbf{X}}^{2j+2} \end{pmatrix}$

step 8 $\mathcal{D} = \mathcal{D} \cup (\hat{\mathbf{X}}^j)$

step 9 return $\hat{\mathbf{X}}^j$

The MMP algorithm described above has the same properties given in appendix D for the UMMP algorithm, that are:

i) Due to the use of the multiscale approach, MMP can perform arbitrarily close to the source $R(D)$ when $d^* \to \infty$.

ii) Fidelity criteria other than the mean squared error can be easily accommodated in the algorithm.

iii) It is easily extendable to higher dimensions. That is, instead of operating on a vector it can operate on a matrix or multi-dimensional array.

iv) It is fully adaptive and therefore universal at least in the sense that no prior knowledge of a source model is needed for it to perform best.

v) It merges the transformation, the quantization and the entropy coding steps of the three-step approach in one.

We also observed, as we did with the UMMP, the following properties:

1. If we are using $\log_2(N)$ bits (using for example an arithmetic coder with uniform N-level distribution) to encode p, we found out that it is better, in a rate-distortion performance sense, to use a fixed segmentation point $p = N/2$, and encode it with 0 bits. Unless a more complex integer code is used for p, the gain we get with the better partition is not worth the increase in bit rate.

2. Better performance is achieved if the quantizer has one reconstruction level, $\mathcal{Q} = \{1\}$. We observed that when the number of quantization level increases, the number of segmentations (and therefore the number of different vectors learned by the algorithm) decreases and vice-versa. In fact, the algorithm learns the same basis vectors with several different amplitudes when the number of quantization levels is small. Therefore, unless a complex joint entropy coding method is used to encode both the dictionary indexes and the quantizer indexes, it is better to use a one-level quantizer and allow the dictionary to learn and explore the joint statistics.

E.2 MMP interpretation as an adaptive vector quantization scheme

.

Another way to look at MMP is as an adaptive vector quantization scheme for Multi-dimensional sources. Differently from classical vector quantizers like LBG and ECVQ, in MMP there is no need to previously optimize the codebook because it is built while MMP encodes the data. The one-dimensional MMP operates on a vector $\mathbf{X} = (X_0 \dots X_{N-1})^T$ output by a vector source $\mathbf{x} = (x_0 \dots x_{N-1})^T$. MMP uses N codebooks $\{\mathcal{C}^{(1)}, \dots, \mathcal{C}^{(N)}\}$ of vectors, initially set to $\mathcal{C}_0^{(n)} = \{\mathbf{S}_0^{(n)}, \dots, \mathbf{S}_{I_n-1}^{(n)}\}, n =$

$1, 2, \ldots, N$ and a scalar quantizer $\mathcal{Q} = \{\alpha_0, \ldots, \alpha_{K-1}\}$. The vectors in $\mathcal{C}^{(n)}$ are of size n, that is, they are at *scale* n. MMP starts searching in the codebook of larger scale, $\mathcal{C}^{(N)}$, for the vector $\alpha_{k_0}\mathbf{S}_{i_0}^{(N)}$ which minimizes the squared error $\xi = \|\mathbf{X}^0 - \alpha_{k_0}\mathbf{S}_{i_0}^{(N)}\|^2$. If the squared error ξ is not greater than a given target distortion d^* times the vector size N, then the search is finished. Otherwise, $\mathbf{X}^0$ is parsed in two segments $\mathbf{X}^1 = (X_0 \ldots X_{p-1})^T$ and $\mathbf{X}^2 = (X_p \ldots X_{N-1})^T$. Then MMP searches the codebooks $\mathcal{C}^{(p)}$ and $\mathcal{C}^{(N-p)}$ for representations of the two segments, that is, the same procedure is recursively applied to $\mathbf{X}^1$ and $\mathbf{X}^2$. After returning from the recursive calls, we have two approximations $\hat{\mathbf{X}}^1$ and $\hat{\mathbf{X}}^2$. With these, we form an estimate of the original vector $\mathbf{X}$ as $\hat{\mathbf{X}}^0 = (\hat{\mathbf{X}}^{1^T}\hat{\mathbf{X}}^{2^T})^T$. The codebook $\mathcal{C}^{(N)}$ is updated by including $\hat{\mathbf{X}}^0$ in it. Since the performance of a codebook, in a rate distortion sense, usually improves as its cardinality $|\mathcal{C}^{(k)}|$ increases, MMP updates the codebooks in all other scales. In order to do that, MMP changes the scale of the vector $\hat{\mathbf{X}}^0$ of size N to scale n using the scale transformation $T_n^N[\hat{\mathbf{X}}^0]$. We know, from appendix C, that good codebooks can result from this updating rule. In fact, in the memoryless Gaussian case, a codebook updated by interpolated vectors can perform even better than ordinarily updated ones.

This description of MMP using multiple codebooks leads to a faster implementation of the MMP algorithm, where the scale transformation is used only when the dictionary is updated (this approach of multiple codebooks can be also used with UMMP). This implementation uses N *gain-shape* codebooks, however, better rate-distortion performance was obtained in our simulations if we adopted a fixed parsing rule $p = N/2$ and a one-level scalar quantizer $\mathcal{Q} = \{1\}$, as in the UMMP. In this case, the MMP implementation can use $\log_2(N) + 1$ *full-search* codebooks.

E.3 MMP performance for $d^* = 0$ and $d^* \to \infty$.

When the vector X is a sample of a discrete memoryless vector in an alphabet $\mathcal{A} = \{a_j\}_{j=0}^{J-1}$, the algorithm should be able to compress it with zero distortion. If necessary, the input vector X can be repeatedly parsed in segments of length one by MMP. Therefore, for lossless capability, we must have at least a combination of $\alpha_k T_1[s_i]$ equal to each alphabet element a_j. For example, $\mathcal{D}_0 = \mathcal{A}, 1 \in \mathcal{Q}$. This is a

sufficient condition for lossless MMP.

Theorem E.3.1 *MMP can lossless compress data for a vector source* $\mathbf{x}$ *if* $\mathcal{D}_0 = \mathcal{A}, 1 \in \mathcal{Q}$.

The performance of MMP at the low rate range is bounded in the following theorem:

Theorem E.3.2 *MMP can lossy compress data from a discrete stationary source with performance within any* $\Delta > 0$ *to it's* $\mathsf{D}(\mathsf{R})$ *function as* $\mathsf{R} \to 0$ *and* $\mathsf{N} \to \infty$.

The argument used to justify theorem D.2.2 applies here.

Theorem E.3.2 lead us to the conjecture:

Conjecture E.3.1 *The performance of MMP is close to the* $\mathsf{R}(\mathsf{D})$ *at low rates.*

The MMP rate-distortion behavior is likely to be continuous. Therefore, we expect the algorithm to perform well in a vicinity of $\mathsf{d}_{\max}$.

In the next section we will use a Gaussian model for the source to carry on a theoretical analysis of some of the aspects of MMP.

E.4 MMP with Gaussian sources

In this section, to investigate the performance of MMP, we will first study the performance of a *fixed-database* version of MMP, the FD-MMP. In FD-MMP, the dictionary is not updated and the vectors in $\mathcal{D}_0$ are used to compress the input $\mathbf{X}$. We expect the performance of MMP to be close to FD-MMP for long stationary sequences, considering that:

1. Practical implementations of MMP must deal with finite memory requirements. Therefore the actual size of the dictionary must be limited to a maximum size M, for example by the use of some pruning strategy. If MMP is applied to a long sequence of samples, the input string will be initially broken in vectors of size N that will be sequentially processed by the algorithm. Eventually the dictionary will reach the maximum size M and MMP will then operate with a fixed size dictionary.

2. The dictionary in MMP is updated using concatenations of segments that resemble blocks of the input sequence. Therefore, if the input sequence is memoryless Gaussian, we expect the vectors in the dictionary to be approximately memoryless Gaussian. In fact, if we set the target distortion to zero, they will be strictly memoryless Gaussian. If the target distortion is $d^* > 0$, then the vectors learned by the algorithm are not identical to the vectors produced by the source and we cannot guarantee that they have the same distribution of the input. However, one can see the process of approximate matching as an exact matching with added noise. Therefore the vectors learned have two components: One that is equal to the input vector $\mathbf{X}$, and another one, η, that is noise. The distribution of the noise vector is unknown but its energy is bounded by $\|\eta\|^2 \leq N d^*$. Therefore, as d^* gets smaller the distribution of the learned vectors gets closer to the distribution of the input vectors. On the other hand, when $d^* \to \infty$, we can compress any $\mathbf{X}$, a sample of a zero-mean memoryless Gaussian process, with mean distortion $d_{max} = \sigma_x^2$ using near zero rate, by replacing the input vector with the all-zeros vector $\mathbf{0}_N$. Also, we know from the $R(D)$ function of the Gaussian process that this performance cannot be improved, as the length of $\mathbf{X}$ increases, by using any other vectors in the dictionary. If both dictionaries of MMP and FD-MMP contain $\mathbf{0}_N$, then MMP and FD-MMP are almost identical since the only vector needed in the dictionary is $\mathbf{0}_N$. In this case the difference in performance is only due to the smaller dictionary of MMP, which will not grow to the maximum size, allowing a smaller rate than the larger (and in this case unnecessary) dictionary of FD-MMP.

3. MMP uses scaled versions of dictionary vectors. If a dictionary vector is memoryless Gaussian, its downsampled versions will also be memoryless Gaussian but its interpolated versions will be Gaussian with memory. Nevertheless this is not a problem since the matching probability for some of the vectors with memory is similar to that of the vectors without memory, as pointed out in section C. Also, if we choose the pruning strategy properly, the vectors with different matching properties tend to be discarded when the dictionary is updated (for example, we could discard the less used vector when updating the

dictionary).

In fact we expect the scaling process to be more useful with short input sequences since it speeds up the learning rate of the algorithm. It will also be more useful at low-rates since for these rates the matching probability with scales is even higher than that of an ordinary match. We also expect better performance of the matching with scales if the source is not memoryless.

The one-dimensional FD-MMP can be described as follows:

Let:

1. $\mathbf{X}$ be a sample of an $N \times 1$ memoryless Gaussian vector source.
2. $\mathcal{D} = \{\mathbf{S}^1, \mathbf{S}^2, \ldots, \mathbf{S}^M\}$ be a dictionary of M samples of an $N \times 1$ memoryless Gaussian source.

If there is a match, output a one bit flag '1' and the index i using $\log_2(M)$ bits.

If we don't have a match, split the vector $\mathbf{x} = (x_1 x_2 \ldots x_N)$ in two vectors $\mathbf{x}' = (x_1 x_2 \ldots x_{\frac{N}{2}})$ and $\mathbf{x}'' = (x_{\frac{N}{2}+1} x_{\frac{N}{2}+2} \ldots x_N)$ and repeat the procedure to each new vector.

Let p_i be the match probability for a Gaussian vector of dimension $2^{-i}N$ and $E\{b_i\}$ be the expected value for the number of bits used by FD-MMP to compress this vector. The expression for p_i is given in appendix C. Then:

$$\begin{aligned}
NE\{r\} = E\{b_0\} &= 1 + p_0 \log_2(M) + 2(1-p_0)E\{b_1\} \\
E\{b_1\} &= 1 + p_1 \log_2(M) + 2(1-p_1)E\{b_2\} \\
&\vdots \\
E\{b_i\} &= 1 + p_i \log_2(M) + 2(1-p_i)E\{b_{i+1}\} \\
&\vdots \\
E\{b_{\log_2(N)}\} &= \log_2(M)N
\end{aligned}$$

$$E\{r\} = \frac{1}{N}\left(1 + p_0 \log_2(M) + \left(\sum_{i=1}^{n-1}(1 + p_i \log_2(M)) \prod_{k=0}^{i-1} 2(1-p_k)\right) + \log_2(M) N \prod_{k=0}^{n-1}(1-p_k)\right) \tag{E.1}$$

where $n = \log_2(N)$, $p_i = P_m(2^{-i}N, M, d^*)$ as in equation C.23.

Let $E\{\xi_i\}$ be the distortion expected value for a vector of length $2^{-i}N$. Then the per-sample distortion expected value $E\{d\}$ for this code is:

$$\begin{aligned}
NE\{d\} = E\{\xi_0\} &= p_0 NE\{\xi_0|match\} + 2(1-p_0)E\{\xi_1\} \\
E\{\xi_1\} &= p_1 2^{-1} NE\{\xi_1|match\} + 2(1-p_1)E\{\xi_2\} \\
&\vdots \\
E\{\xi_i\} &= p_i 2^{-i} NE\{\xi_i|match\} + 2(1-p_i)E\{\xi_{i+1}\} \\
&\vdots \\
E\{\xi_{\log_2(N)}\} &= NDm(1, M, \infty)
\end{aligned}$$

$$E\{d\} = \frac{1}{N}\left(p_0 D_0^m + \left(\sum_{i=1}^{n-1}(p_i 2^{-i} N D_i^m)\prod_{k=0}^{i-1} 2(1-p_k)\right) + NDm(1, M, \infty)\prod_{k=0}^{n-1}(1-p_k)\right) \tag{E.2}$$

where $n = \log_2(N)$, $D_i^m = Dm(2^{-i}N, M, d^*)$ as in equation C.25 and p_i is the same as in equation E.1.

Figure E.2 shows the performance for FD-MMP with $N = 64$ and $M = 256$. We can see that the distortion is greater than the maximum theoretical distortion $R(0) = \sigma_x^2$ at the minimum rate achieved by the code. This is expected since for a random dictionary and finite M there is a nonzero probability that the difference vector has a variance greater than σ_x^2. The performance of FD-MMP can be improved by replacing one of the dictionary vectors by the zero vector $\mathbf{0}_N = (0 \ldots 0)^T$.

The performance for FD-MMP with $\mathbf{0}_N$ included in the dictionary is then given by the same equations E.1 and E.2, where $D_i = Dm(2^{-i}N, M, d^*)$ is given by equation C.29 and $p_i = P_m(2^{-i}N, M, d^*)$ is given by equation C.28.

Figure E.3 shows the performance for FD-MMP with $N = 64$ and $M = 256$, with the zero vector included in the dictionary. It also shows the $R(D)$ and performance for FD-MMP with random dictionary. We can see that the addition of the zero vector has improved the performance at the low rate range.

We considered in equations E.1 and E.2 that the dictionaries used to match vectors of different sizes were independent. In MMP this will not be true since there

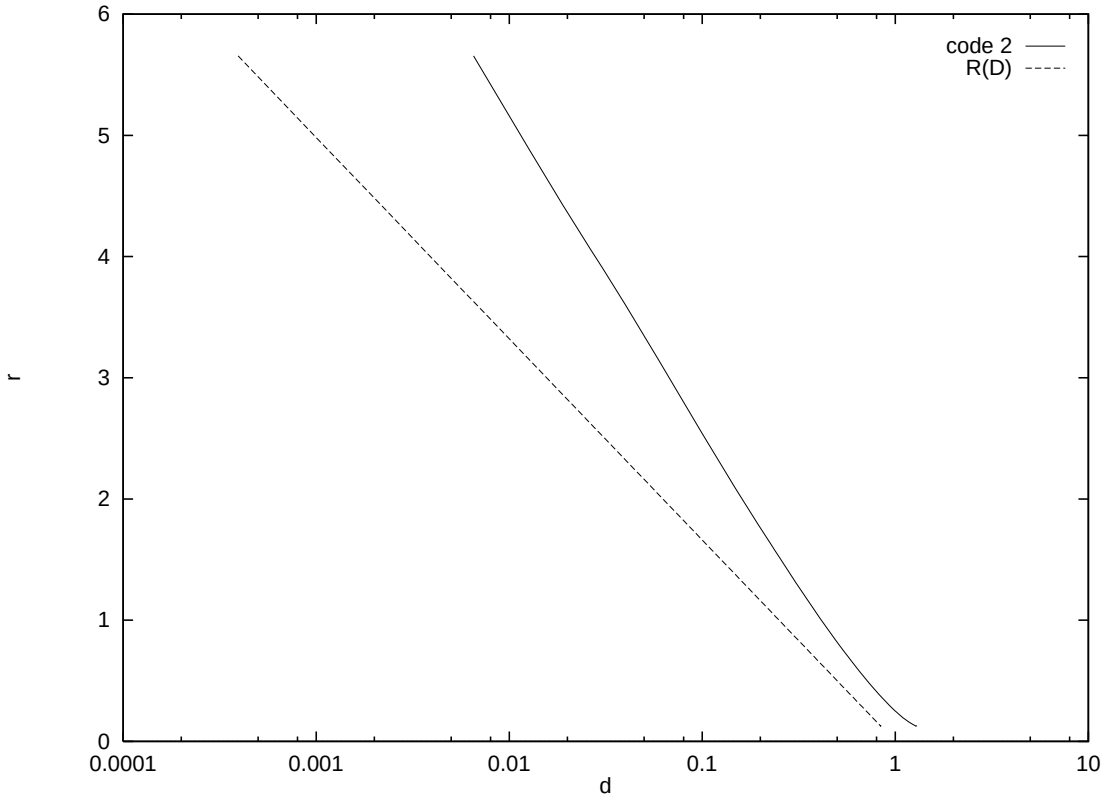


Figura E.2: Rate × distortion for FD-MMP.

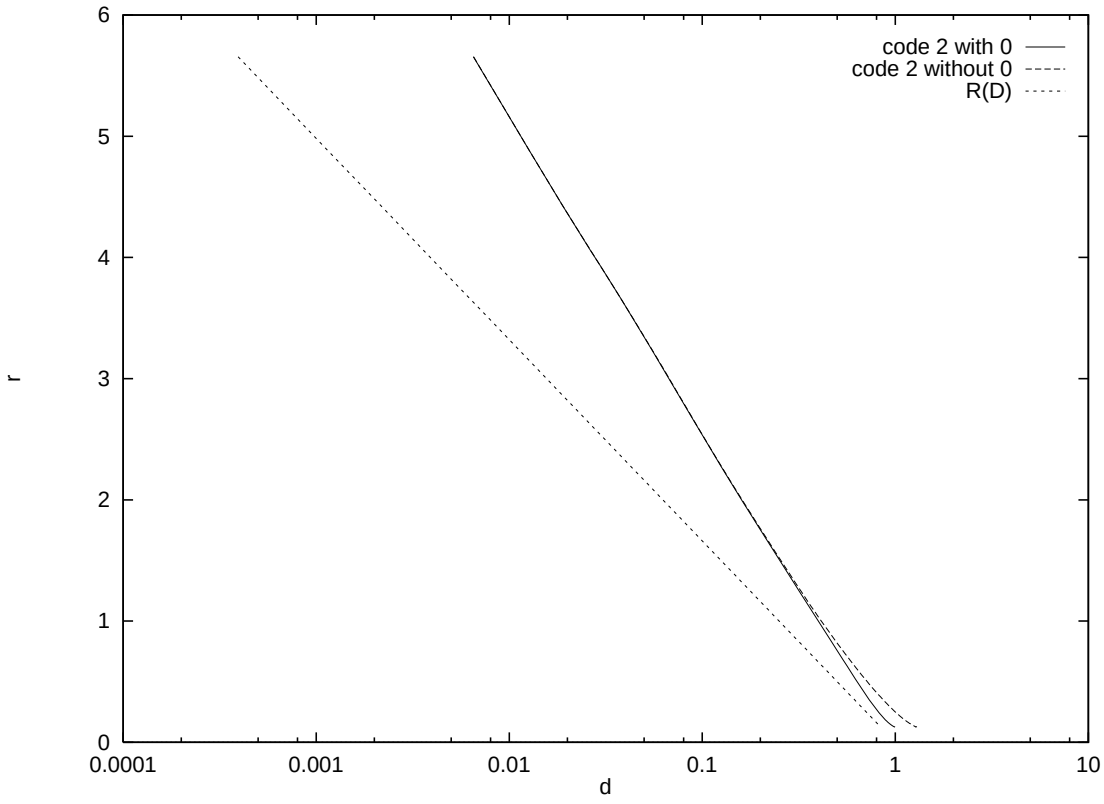


Figura E.3: Rate × distortion for FD-MMP with zero vector included.

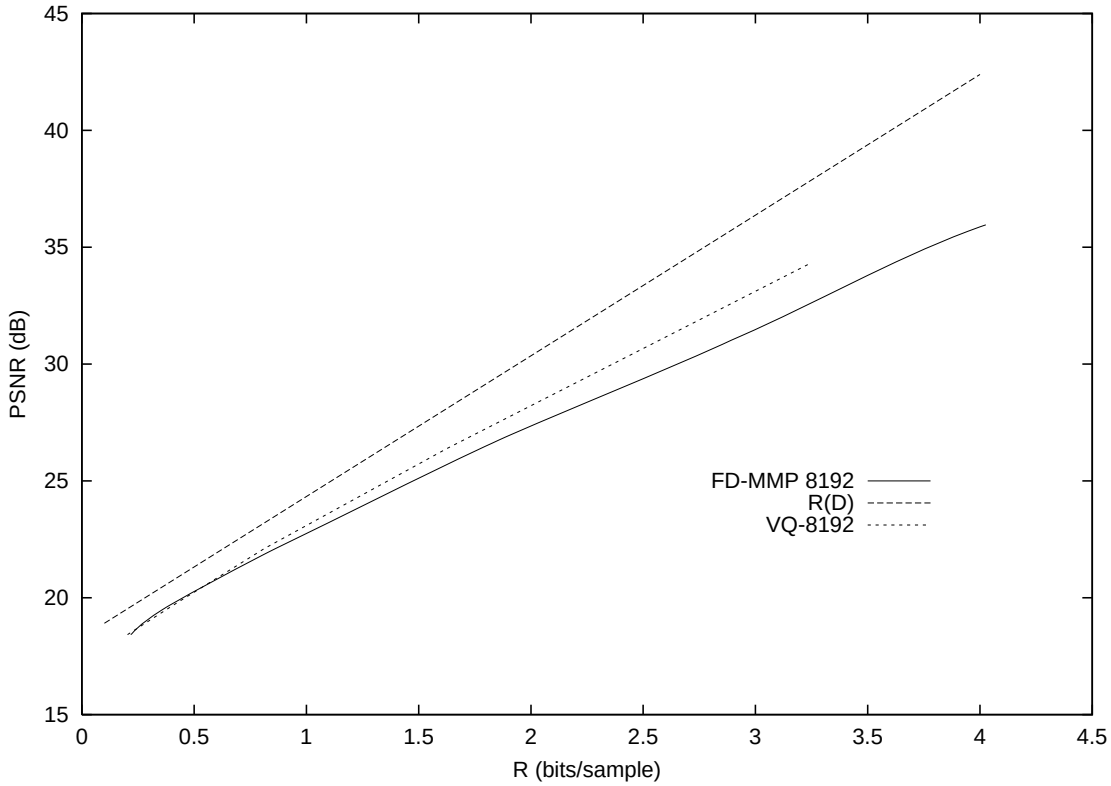


Figura E.4: Rate × distortion for VQ and FD-MMP.

are versions of all vectors in all scales. In fact, if a match in one scale fails, the probability of a match in a smaller scale tends to be lower (at most equal) than it would be if the dictionaries were independent. This implies that the FD-MMP rate-distortion performance will be an upper bound for the MMP rate-distortion performance. This effect can be attenuated, for long input sequences, if we keep separate dictionaries for each scale in MMP and allow different pruning decisions for each one. This is also useful to develop a faster version of MMP where the scaling is done only when we update the dictionaries.

Figure E.4 shows the theoretical performance of FD-MMP with a dictionary of size 8192, the expected performance of a vector quantizer with 8192 vectors and the optimum theoretical $R(D)$.

We can see in figure E.4 that the performance of FD-MMP is worse than that of a VQ. The difference is due to the "match/no match" flags. In the FD-MMP rate-distortion calculations the flags were coded using 1 bit and the indexes using $\log_2(M)$ bits, without entropy coding. We expect the entropy of the indexes to be close to this value, since the vectors tend to be equally probable, especially

for long blocklengths. However, we expect the entropy of the flags to be less than one for stationary sources. In fact, for a given target distortion d^* there is an optimum blocklength N^* that approaches the $R(D)$ function of the source. FD-MMP uses blocks with lengths that are powers of two, therefore it approximates the optimum blocklength N^* in the mean, that is, it switches between $N' = 2^k \leq N^*$ and $N'' = 2^{k+1} \geq N^*$, with k an integer. Only the flags associated with this two blocklengths will have an entropy significantly different from zero. Considering this, the performance of FD-MMP should be closer to that of the VQ if we use an appropriate entropy encoder for the flags.

In the next section we show experimental results obtained from computer experiments. In the simulations we used a variety of sources, both one and two-dimensional, and we compare the results to those of other popular lossy compression algorithms.

E.5 Experimental results

We have made some computer simulations of MMP. We wanted to evaluate the rate-distortion performance of UMMP and MMP when applied to two-dimensional data, specifically gray-scale digital images. So we implemented the two-dimensional versions of UMMP (2D-UMMP) and MMP (2D-MMP). The 2D-UMMP was described in section D and the 2D-MMP uses the same segmentation rule:

- `if` $\mathbf{X}^j$ is $N \times M$ where $N > M$, `then` $\mathbf{X}^{2j+1}$ and $\mathbf{X}^{2j+2}$ are $N/2 \times M$.
- `else` $\mathbf{X}^{2j+1}$ and $\mathbf{X}^{2j+2}$ are $N \times M/2$.

We have implemented both 2D-UMMP and 2D-MMP with this fixed segmentation rule. These implementations also used a one level scalar quantizer $\mathcal{Q} = \{1\}$ because, as in the one-dimensional case, this leads to better performance. The 2D-MMP and 2D-UMMP both used a multiple codebook approach.We have limited the matrix size to powers of two, that is, $\mathbf{X}$ must be of size $(N \times M) = (2^K \times 2^L)$ with K, L integers. To use the segmentation rule described above, we need $K+L+1$ dictionaries at the scales $(1 \times 1), (2 \times 1), (2 \times 2), \ldots, (N \times M)$. We denote each dictionary

$\mathcal{D}^{(k,l)}$, meaning that it contains matrices of size (scale) $2^k \times 2^l$. The 2D-MMP is described next:

Let:

- $\mathbf{X}$ be an $(N \times M) = (2^K \times 2^L)$ matrix.
- $\mathcal{D}^{(k,l)} = \{\mathbf{s}_0^{(k,l)}, \ldots, \mathbf{s}_{I-1}^{(k,l)}\}$,
  $(k,l) = (0,0), (1,0), (1,1) \ldots, (K,L)$ a set of $K + L + 1$ dictionaries with $I^{(k,l)}$ matrices of size $2^k \times 2^l$ each. This dictionaries must be initialized to $\mathcal{D}^{(k,l)} = \mathcal{D}_0^{(k,l)}$.
- d^* be a target distortion.
- $T_{N,M}[\mathbf{X}]$ be a scale transformation that maps the matrix $\mathbf{X}$ in a matrix of size $N \times M$ (see equation D.4 for an example).

Procedure $\hat{\mathbf{X}}^j = \text{encode}(\mathbf{X}^j, d^*)$:

step 1 find index i_j in the dictionary of the same scale (k, l) of $\mathbf{X}^j$
such that $\|\mathbf{X}^j - \mathbf{s}_{i_j}^{(k,l)}\|$ is minimum and make $\hat{\mathbf{X}}^j = \mathbf{s}_{i_j}^{(k,l)}$.

step 2 `if` $k = l = 0$, output index i_j and return $\hat{\mathbf{X}}^j$.
`else` got to step 3.

step 3 `if` $\|\mathbf{X}^j - \hat{\mathbf{X}}^j\|^2 \leq 2^{k+l} d^*$ `then` output flag '1', output index i_j and return $\hat{\mathbf{X}}^j$.
`else` go to step 4.

step 4 output flag '0'.

step 5 `if` $k > l$ split $\mathbf{X}^j = \begin{pmatrix} \mathbf{X}^{2j+1} \\ \mathbf{X}^{2j+2} \end{pmatrix}$,
where $\mathbf{X}^{2j+1}$ and $\mathbf{X}^{2j+2}$ are $2^{k-1} \times 2^l$
`else` split $\mathbf{X}^j = \begin{pmatrix} \mathbf{X}^{2j+1} & \mathbf{X}^{2j+1} \end{pmatrix}$
where $\mathbf{X}^{2j+1}$ and $\mathbf{X}^{2j+2}$ are $2^k \times 2^{l-1}$

step 6 compute $\hat{\mathbf{X}}^{2j+1} = \text{encode}(\mathbf{X}^{2j+1}, d^*)$

step 7 compute $\hat{\mathbf{X}}^{2j+2} = \text{encode}(\mathbf{X}^{2j+2}, d^*)$

step 8 `if` $k > l$ make $\hat{\mathbf{X}}^j = \begin{pmatrix} \hat{\mathbf{X}}^{2j+1} \\ \hat{\mathbf{X}}^{2j+2} \end{pmatrix}$,
`else` make $\hat{\mathbf{X}}^j = \begin{pmatrix} \hat{\mathbf{X}}^{2j+1} & \hat{\mathbf{X}}^{2j+2} \end{pmatrix}$

step 9 `for` $n = 0, 1, \ldots, K + L + 1$,
$p = \left\lfloor (\frac{n+1}{2}) \right\rfloor$, $q = \left\lfloor (\frac{n}{2}) \right\rfloor$,
$\mathcal{D}^{(p,q)} = \mathcal{D}^{(p,q)} \cup \left\{ T_{2^p, 2^q}[\hat{\mathbf{X}}^j] \right\}$
where $\lfloor x \rfloor$ is the largest integer that is smaller than or equal to x .

step 10 return $\hat{\mathbf{X}}^j$

The 2D-UMMP algorithm implemented with multiple codebooks is similar to the 2D-MMP and is described next:

$\mathbf{R}^{0,0}$ is initially set to $\mathbf{X}$.

Procedure $\hat{\mathbf{R}}^{r,j} = \text{encode}(\mathbf{R}^{r,j}, d^*)$:

step 1 find index $i_{r,j}$ in the dictionary of the same scale (k,l) of $\mathbf{R}^{r,j}$
such that $\|\mathbf{R}^{r,j} - \mathbf{s}^{(k,l)}_{i_{r,j}}\|$ is minimum and make $\hat{\mathbf{R}}^{r,j} = \mathbf{s}^{(k,l)}_{i_{r,j}}$.

step 2 output index $i_{r,j}$

step 3 `if` $k = l = 0$ return $\hat{\mathbf{R}}^{r,j}$.
`else` go to step 4.

step 4 `if` $\|\mathbf{R}^{r,j} - \hat{\mathbf{R}}^{r,j}\|^2 \le 2^{k+l} d^*$ `then` output flag '1' and return $\hat{\mathbf{R}}^{r,j}$.
`else` go to step 5.

step 5 make $\mathbf{R}^{r+1,j} = \mathbf{R}^{r,j} - \hat{\mathbf{R}}^{r,j}$ and output flag '0'.

step 6 `if` $k > l$ split $\mathbf{R}^{r+1,j} = \begin{pmatrix} \mathbf{R}^{r+1,2j+1} \\ \mathbf{R}^{r+1,2j+2} \end{pmatrix}$,
where $\mathbf{R}^{r+1,2j+1}$ and $\mathbf{R}^{r+1,2j+2}$ are $2^{k-1} \times 2^l$
`else` split $\mathbf{R}^{r+1,j} = \begin{pmatrix} \mathbf{R}^{r+1,2j+1} & \mathbf{R}^{r+1,2j+2} \end{pmatrix}$
where $\mathbf{R}^{r+1,2j+1}$ and $\mathbf{R}^{r+1,2j+2}$ are $2^k \times 2^{l-1}$

step 7 compute $\hat{\mathbf{R}}^{r+1,2j+1} = \text{encode}(\mathbf{R}^{r+1,2j+1}, d^*)$

step 8 compute $\hat{\mathbf{R}}^{r+1,2j+2} = \text{encode}(\mathbf{R}^{r+1,2j+2}, d^*)$

step 9 `if` $k > l$ make $\hat{\mathbf{R}}^{r+1,j} = \begin{pmatrix} \hat{\mathbf{R}}^{r+1,2j+1} \\ \hat{\mathbf{R}}^{r+1,2j+2} \end{pmatrix}$,
`else` make $\hat{\mathbf{R}}^{r+1,j} = \begin{pmatrix} \hat{\mathbf{R}}^{r+1,2j+1} & \hat{\mathbf{R}}^{r+1,2j+2} \end{pmatrix}$

step 10 make $\hat{\mathbf{R}}^{r,j} = \hat{\mathbf{R}}^{r,j} + \hat{\mathbf{R}}^{r+1,j}$.

step 11 `for` $n = 0, 1, \ldots, K + L + 1$,
$p = \left\lfloor (\frac{n+1}{2}) \right\rfloor$, $q = \left\lfloor (\frac{n}{2}) \right\rfloor$,
$\mathcal{D}^{(p,q)} = \mathcal{D}^{(p,q)} \cup \left\{ T_{2^p,2^q}[\hat{\mathbf{R}}^{r,j}] \right\} \cup \left\{ T_{2^p,2^q}[\hat{\mathbf{R}}^{r+1,j}] \right\}$
where $\lfloor x \rfloor$ is the largest integer that is smaller than or equal to x .

step 12 return $\hat{\mathbf{R}}^{r,j}$

We used the computer programs to lossy compress the images LENA, BABOON, F16, BRIDGE, AERIAL, BARBARA and GOLD, all of size 512×512.

These images were downloaded from http://sipi.www.usc.edu/services/database/, and are shown in figures J.1 to J.7. We also applied MMP to the images PP1209 and PP1205, both of size 512×512, shown in figures J.9 and J.8. PP1209 is a compound of a compressed grayscale Lena with text and graphics. This image was scanned from the *IEEE Transactions on Image Processing*, Volume 9, number 7, July 2000, page 1209. PP1205 is a pure text image scanned from page 1205 of the same magazine. The images were initially divided in 8×8 blocks that were processed in sequence by the algorithm. Figures E.5 to E.11 show the results for 2D-MMP and 2D-UMMP. The size of the dictionaries were limited by use of a pruning strategy. Whenever a new element is included in the dictionary, the least used element of that dictionary is discarded. The maximum size of the dictionaries was 32768. The initial dictionary for both algorithms at scale 1×1 was $\mathcal{D}_0^{(0,0)} = \{\min(X_{n,m}), \min(X_{n,m}) + \delta, \ldots, 0, \delta, \ldots, \max(X_{n,m})\}$, where $\min(X_{n,m}), \max(X_{n,m})$ are respectively the minimum and the maximum values for the pixels in the image and $\delta = \frac{2(\max(X_{n,m}) - \min(X_{n,m}))}{\mathrm{sqrt}(d^*)}$. The initial dictionaries at the other scales were obtained from $\mathcal{D}_0^{(0,0)}$ by the use of the scale transformation. The scale transformation used was the same defined in appendix D, equations D.2, D.3 and D.4. The integers output by the algorithms were encoded using an adaptive arithmetic coder with different models for the dictionary indexes at each scale. The flags were also encoded using the adaptive arithmetic coder with independent models for the flags corresponding to each scale. Also shown are the results for LLZ [8], JPEG [10] and SPIHT [13].

Figure E.14 shows the maximum size of the dictionaries of MMP as a function of the bit rate, for all the test images. The variation is quite linear and the dictionary size is almost independent of the source.

We studied the variation of the performance of the MMP algorithm versus the size of the blocks. We tested the 2D-MMP applied to the test images initially divided in blocks of sizes $1 \times 1, 2 \times 2, \ldots, 16 \times 16$. The results for the image LENA are shown in figure E.15. These results for LENA are quite typical. Similar behavior was observed for the other test images. We can see that the performance improves as the size of the block increases. This is expected for a VQ scheme and for the 8×8 case the PSNR is almost 20 dB higher than the performance for the scalar

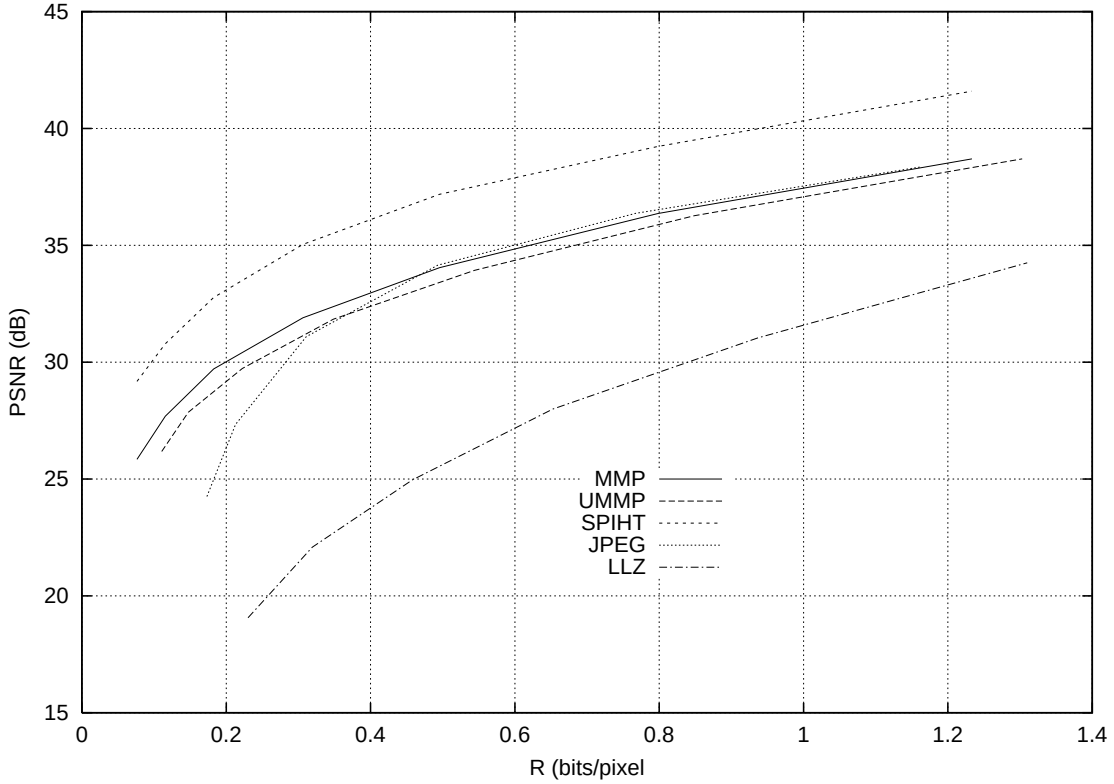


Figura E.5: Rate × distortion for 2D-UMMP and 2D-MMP with LENA 512×512.

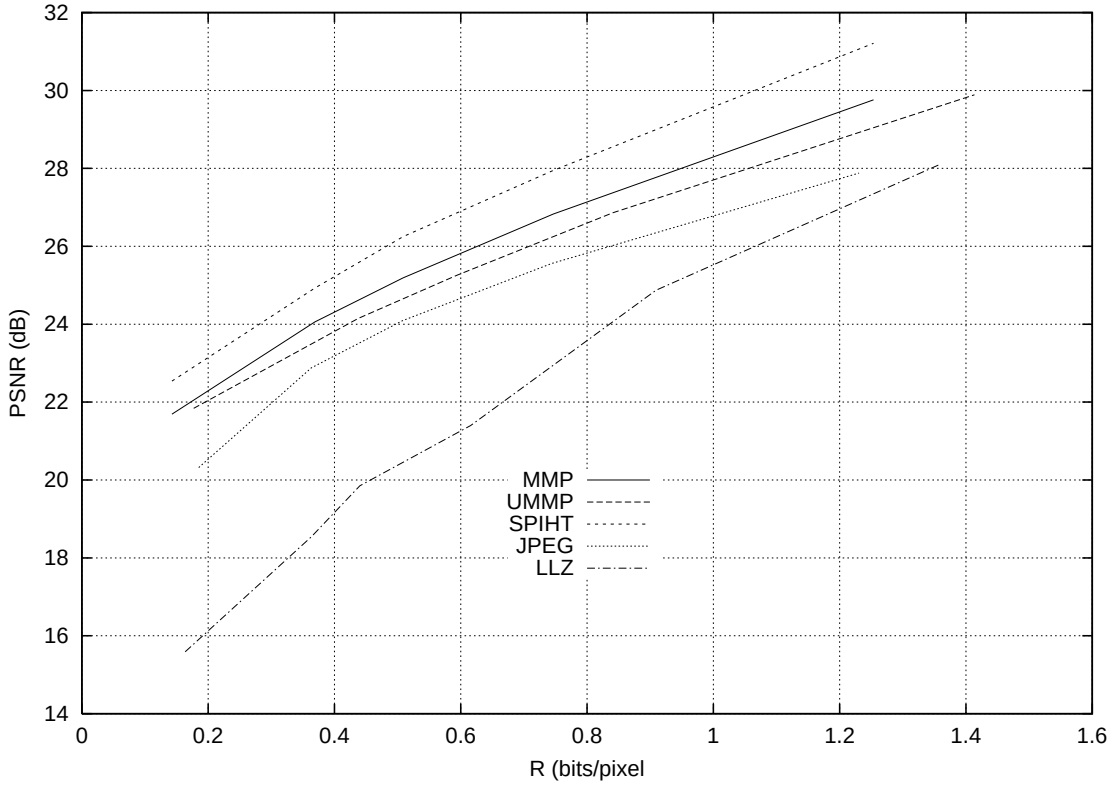


Figura E.6: Rate × distortion for 2D-UMMP and 2D-MMP with BABOON 512×512.

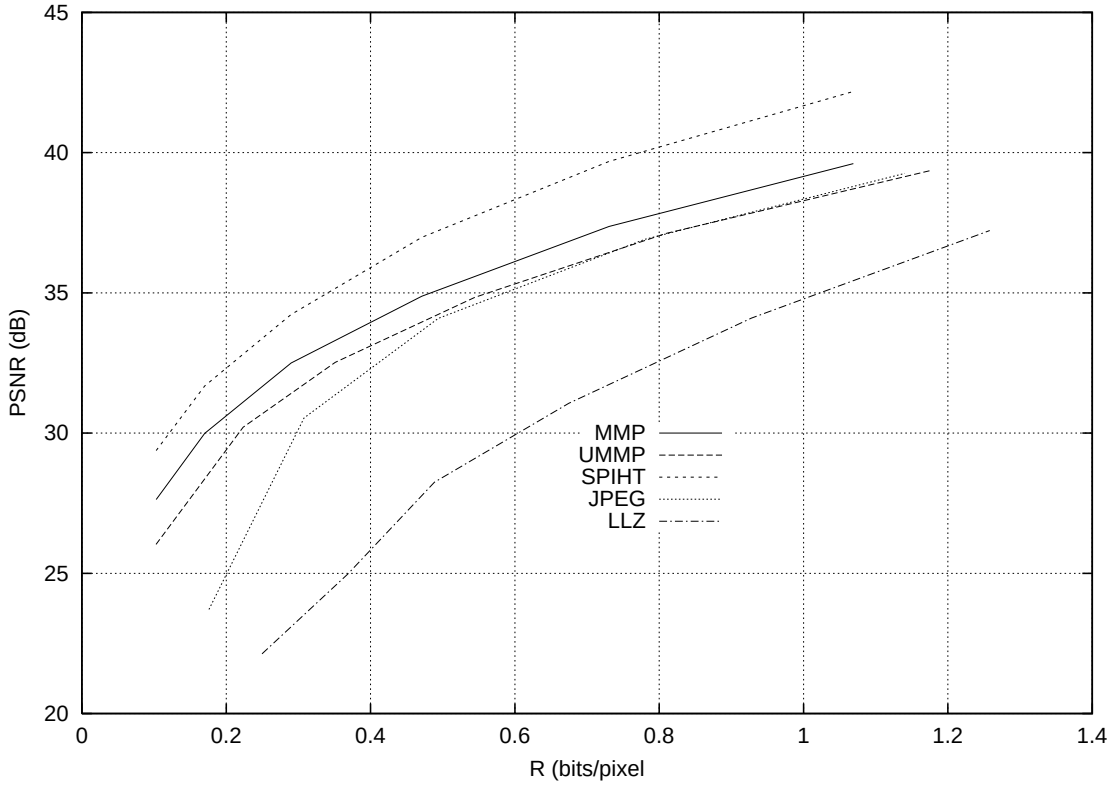


Figura E.7: Rate $\times$ distortion for 2D-UMMP and 2D-MMP with F16 512×512.

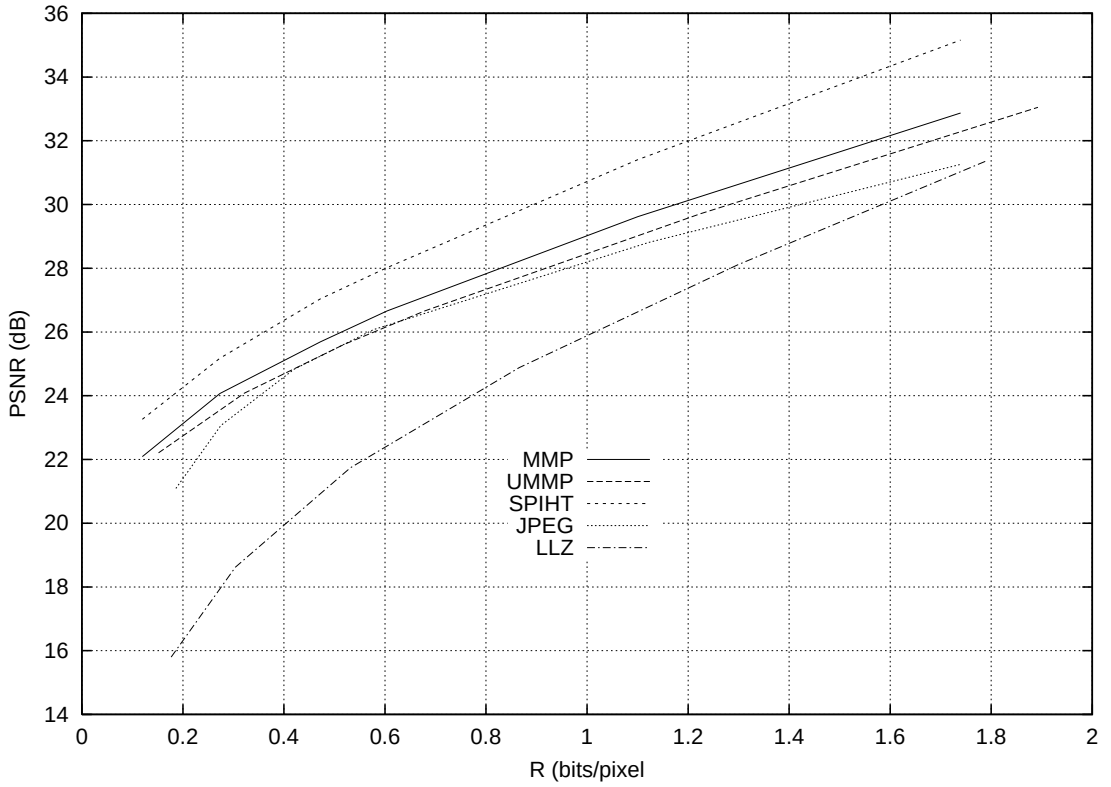


Figura E.8: Rate $\times$ distortion for 2D-UMMP and 2D-MMP with BRIDGE 512×512.

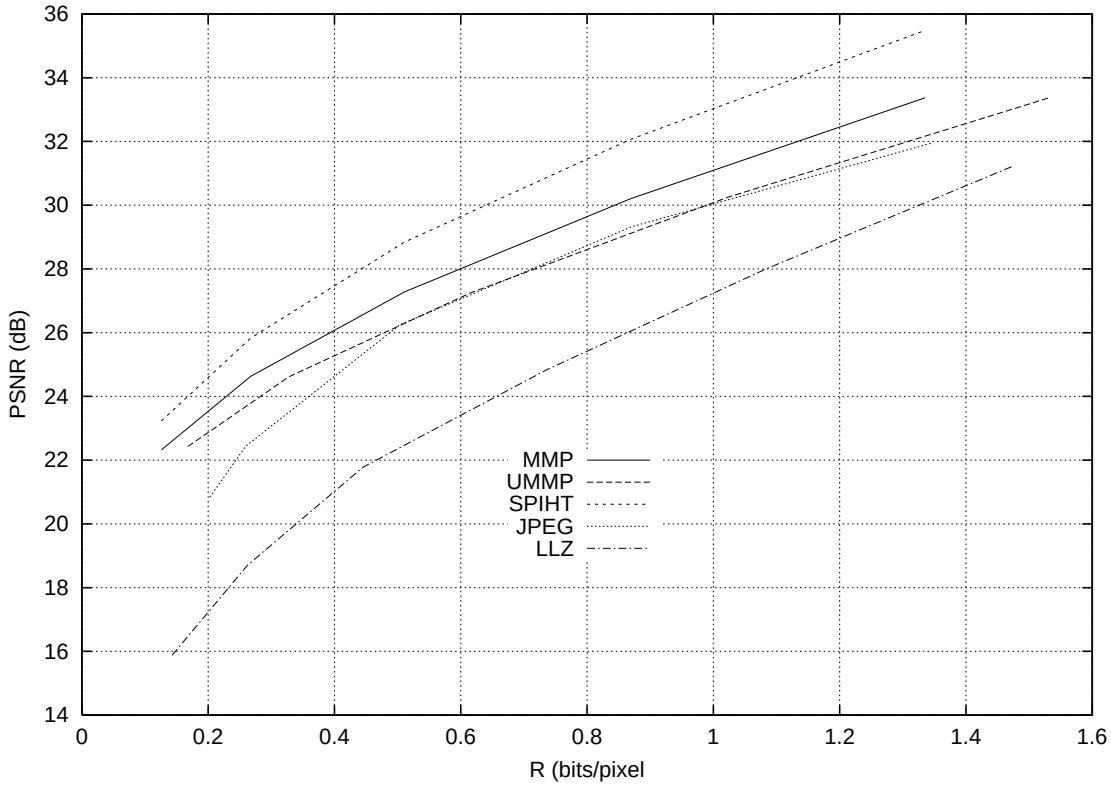


Figura E.9: Rate × distortion for 2D-UMMP and 2D-MMP with AERIAL 512×512.

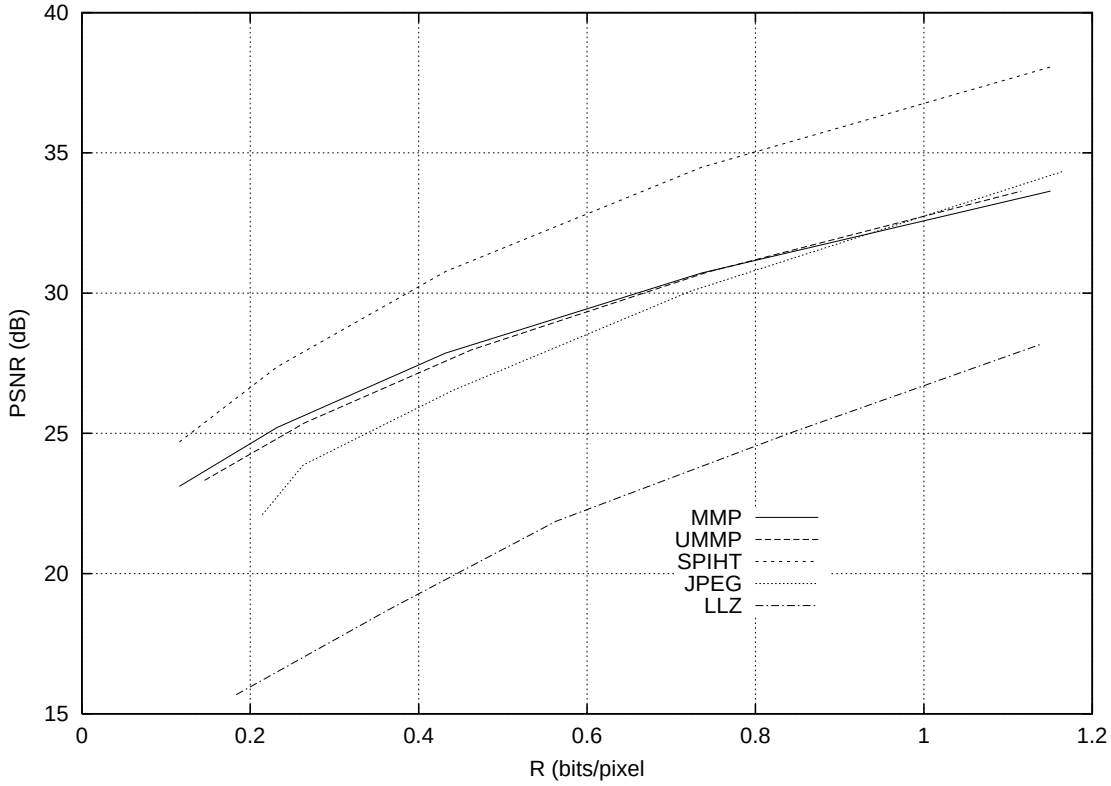


Figura E.10: Rate × distortion for 2D-UMMP and 2D-MMP with BARBARA 512×512.

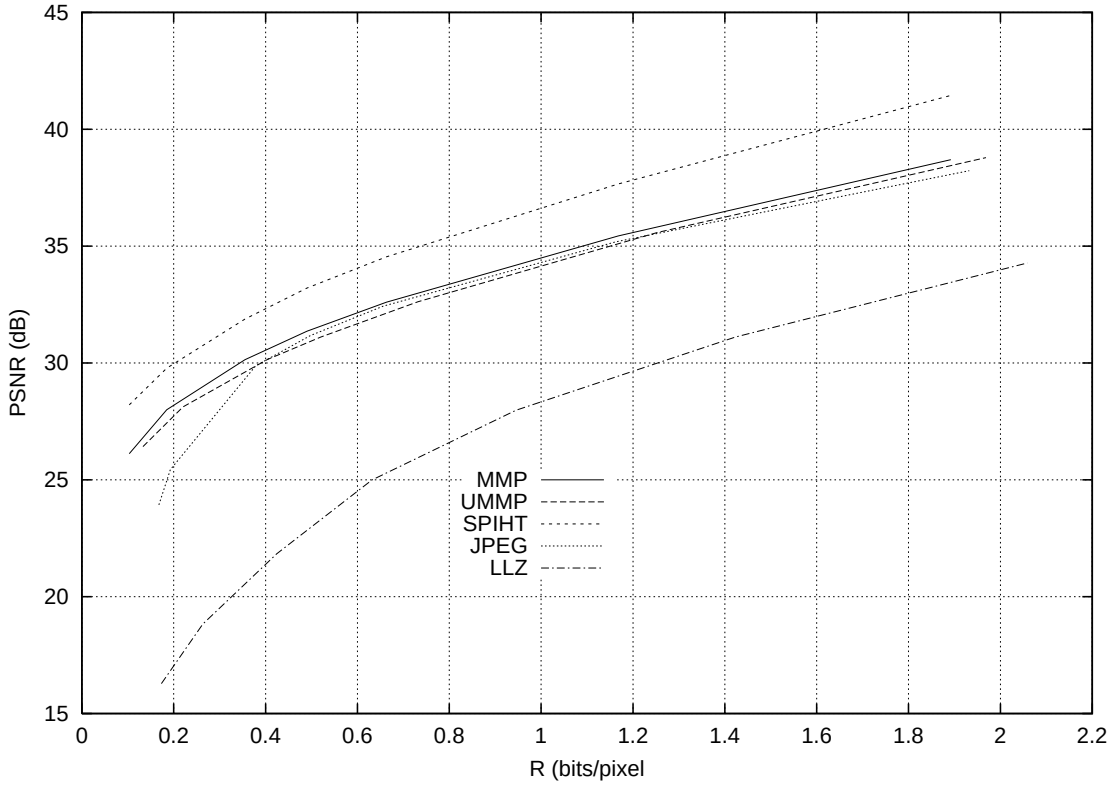


Figura E.11: Rate × distortion for 2D-UMMP and 2D-MMP with GOLD 512×512.

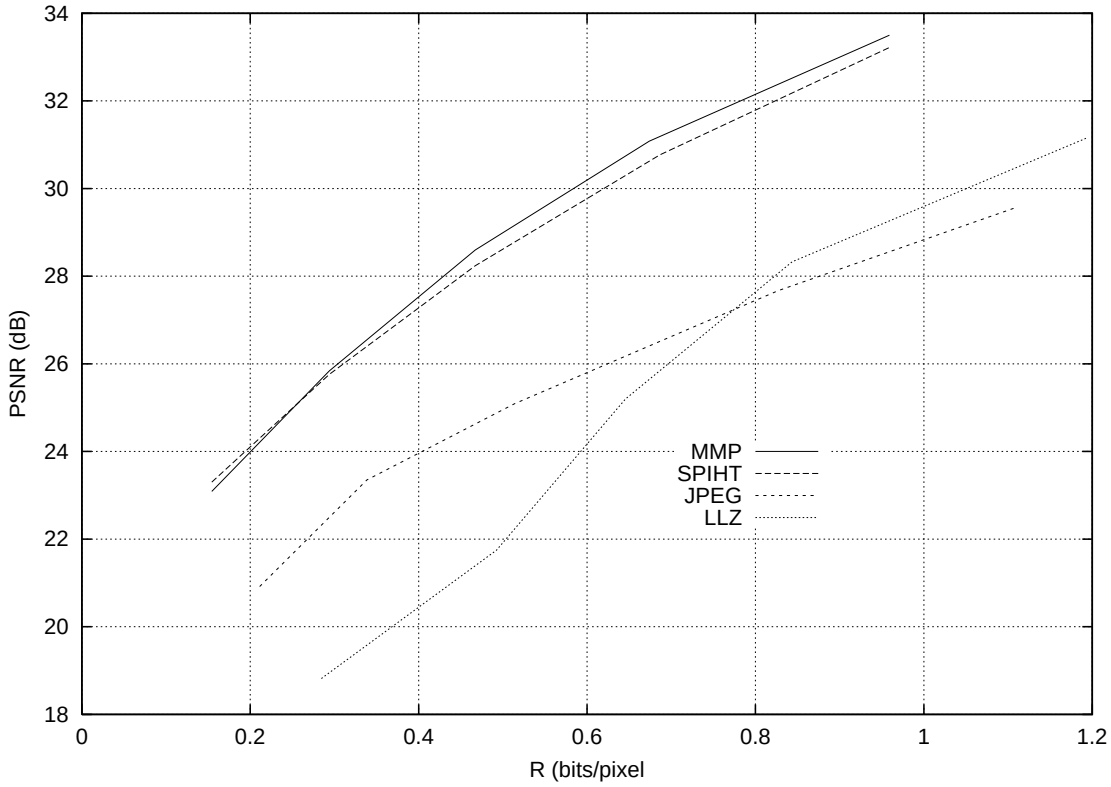


Figura E.12: Rate × distortion for 2D-UMMP and 2D-MMP with PP1209 512×512.

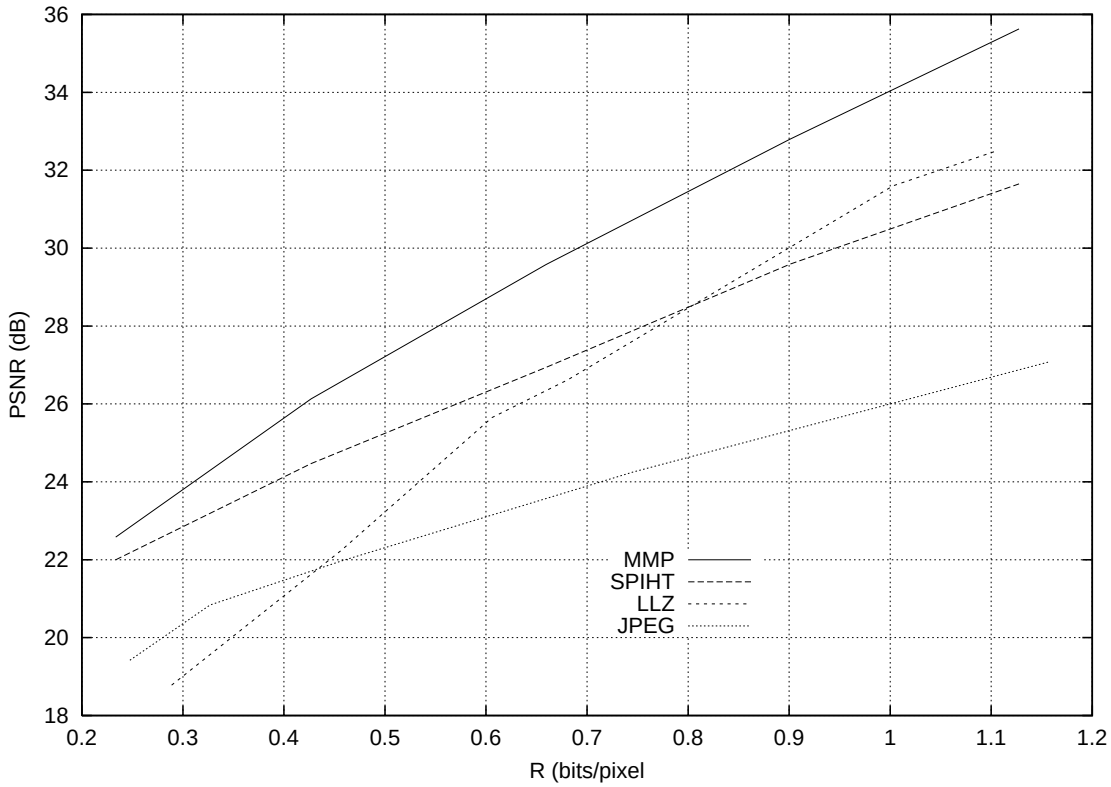


Figura E.13: Rate × distortion for 2D-UMMP and 2D-MMP with PP1205 512×512.

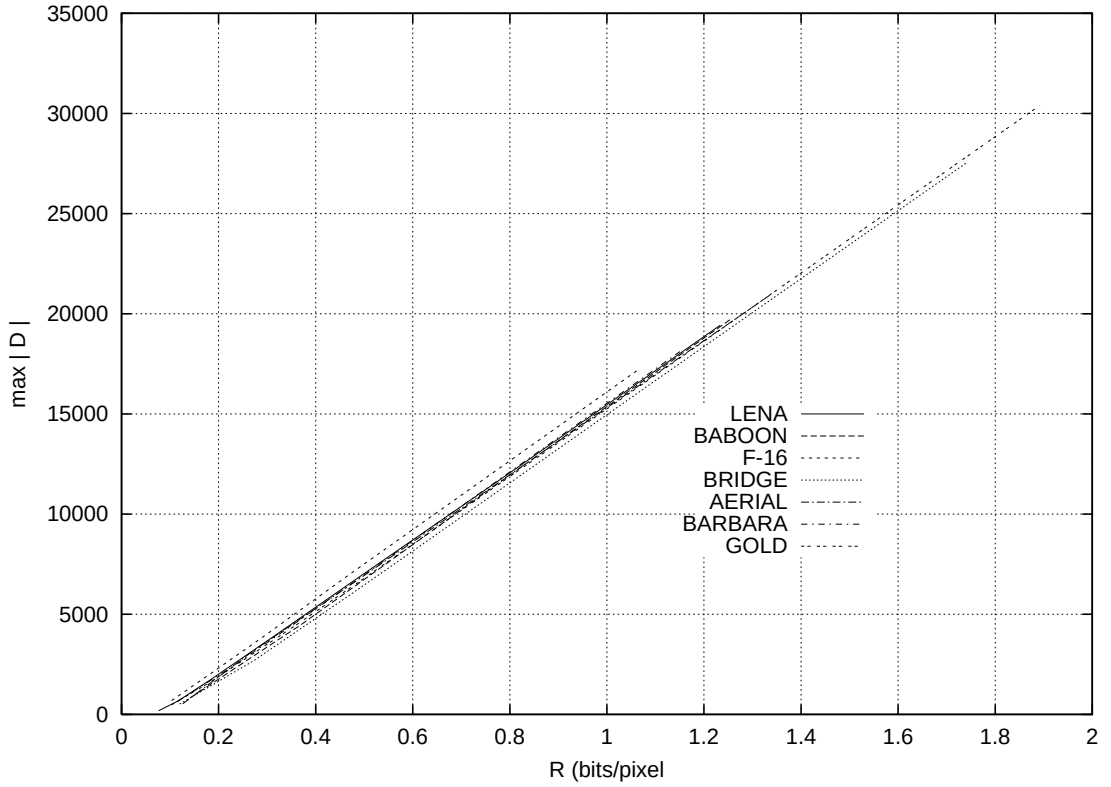


Figura E.14: Maximum dictionary size × rate for 2D-MMP.

(1×1) case. However, there is no further improvement after the block reaches 8×8 except for very low rates, below 0.15 bits/pixel. In fact the performance for 16×16 blocks is a little worse than the performance for 8×8 blocks. This can be partially explained if we consider that the matching probability for the matrices in the dictionaries at the larger scales decreases as the target distortion d^* decreases. Therefore, at high rates, there is nothing to gain by increasing the size of the block since most of the matchings will occur at scales where the matrices are smaller than the block. The additional cost in bit rate expended to indicate segmentation until the algorithm reaches these smaller scales will reduce the efficiency of the code. This effect explains why the performance decreases at high rates but increases at low rates when the block size is above a threshold. However, since the flags are entropy encoded, the difference in performance at high rates, despite being small, should be much smaller. We believe that the main cause of the decrease of the performance in this case is the non-optimality of the segmentation tree created by the MMP. In appendix F we develop a rate distortion optimized version of MMP. In our simulations, the performance of this enhanced algorithm actually improved as we increased the size of the blocks from 8×8 to 16×16.

Figure E.16 shows the performance of 2D-MMP and 2D-UMMP with a Gaussian memoryless source. The parameters used in the algorithms were the same as in the case of the image LENA. The source was a 512×512 matrix of samples of a memoryless Gaussian process with mean 128.0 and variance 960.0. Also shown are the expected performance of a vector quantizer with 8192 vectors, the theoretical performance of FD-MMP with a dictionary of size 8192 and the optimum theoretical $R(D)$.

As we can see from figures E.5 to E.11, the best performance with all the pure gray-scale images was obtained by the wavelet coder SPIHT, followed by 2D-MMP. After that, came the 2D-UMMP, then the transform coder JPEG, and finally the LLZ. An exception occurred for LENA where JPEG is slightly better (within 0.1 dB) than 2D-MMP at rates in the interval 0.5 to 1.2 bits/pixel. We can get some insight on the advantage of 2D-MMP over 2D-UMMP by considering the following example: Suppose we try to approximate the input vector $\mathbf{X}$ with UMMP in one step using a equal-components vector from the initial dictionary $\mathcal{D}_0$ and fail. UMMP

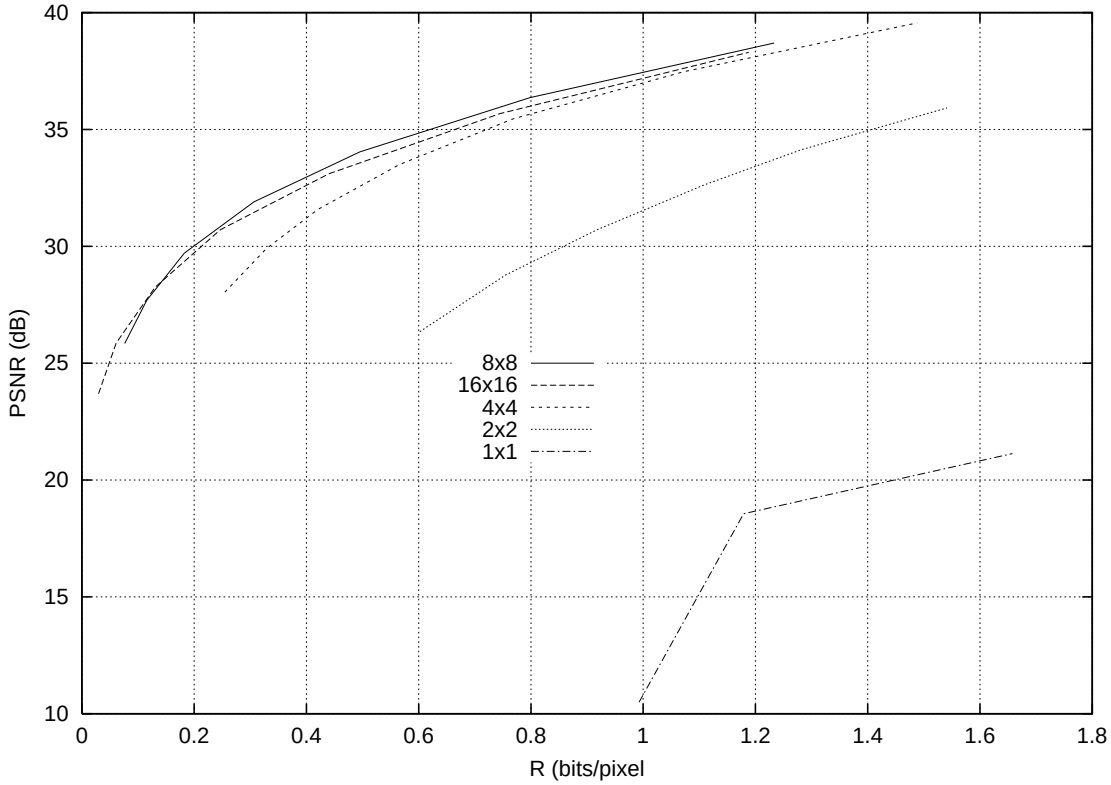


Figura E.15: Performance for several block-sizes.

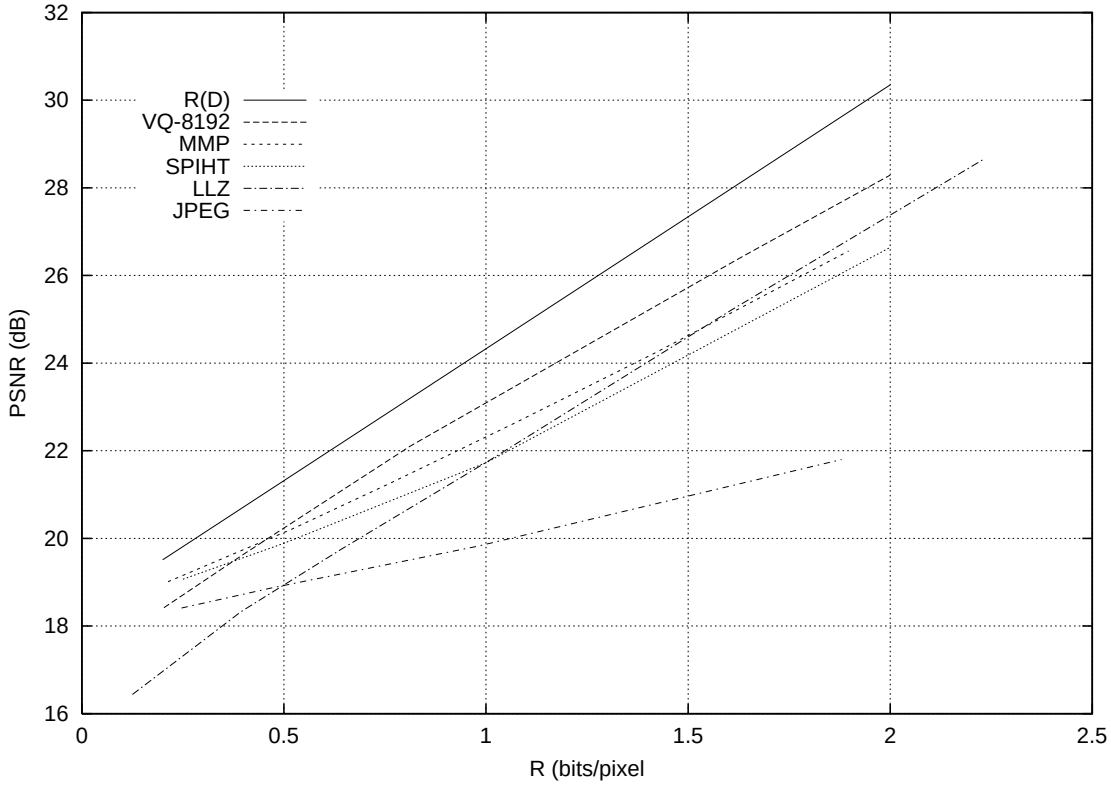


Figura E.16: R(D) curve for 2D-MMP and 2D-UMMP with Gaussian memoryless source.

will subtract this approximation from $\mathbf{X}$ and the resulting residue $\mathbf{R}$ will have near zero mean. UMMP then parses the residue in two segments. Now let's suppose that UMMP successfully approximates this two segments in one step each. We then have an approximation using three vectors from the dictionaries. MMP on the other hand will also successfully approximate the input in two segments, but will use only two vectors. Considering that *both use the same initial dictionary* there may be no advantage in the mean-removal step performed by UMMP. UMMP could possibly perform better if we take into consideration that the power of the residue vector tends to decrease, and use this information to modify the power of the vectors to be included in the dictionary. It might also be that the reconstruction at the UMMP decoder using the same vectors used at the encoder is sub-optimal. The UMMP performs an expansion of the input vector in a *frame*. It is known that the optimal frame for reconstruction is the *dual frame*. Therefore, the performance of UMMP might be improved if we used the dual frame to reconstruct, maybe using *inverse frame computations* as in [17]. LLZ is not expected to perform well with this type of source because it is a one-dimensional algorithm but it is used as a reference of another adaptive lossy compression scheme based on approximate pattern matching. Figures E.17, E.18 and E.19 ilustrtes the performance with LENA at 0.5 bits/pixel for the 2D-MMP, 2D-UMMP and SPIHT algorithms respectively.

With the text image PP1205, the MMP algorithm outperforms all other coders, as ilustrated in figure E.13. At 0.5 bits/pixel, it is 2 dB above the second best performer. Both JPEG and SPIHT are transform based coders that decompose the input image in several bands, prior to the quantization performed on the transformed domain. They assume that natural images have a high concentration of energy in the low frequency bands, with an approximately exponencial decay towards the higher frequency bands. Also, the SPIHT coder assumes that the remaining high frequency energy is spatially concentrated in a few clusters around the regions in each band that correspond to the location of the edges in the original image. These assumptions are not true with text images like PP1205. However, the MMP algorithm assumes nothing with respect to the image statistics. Instead, it learns from the data itself everything it needs to build its dictionary. The segmentation procedure allows MMP to perform very well at the edges in an image. Therefore it performs very well with

Figura E.17: LENA compressed at 0.5 bpp by 2D-MMP. PSNR = 34.0 dB.

Figura E.18: LENA compressed at 0.5 bpp by 2D-UMMP. PSNR = 33.5 dB.

Figura E.19: LENA compressed at 0.5 bpp by SPIHT. PSNR = 37.2 dB.

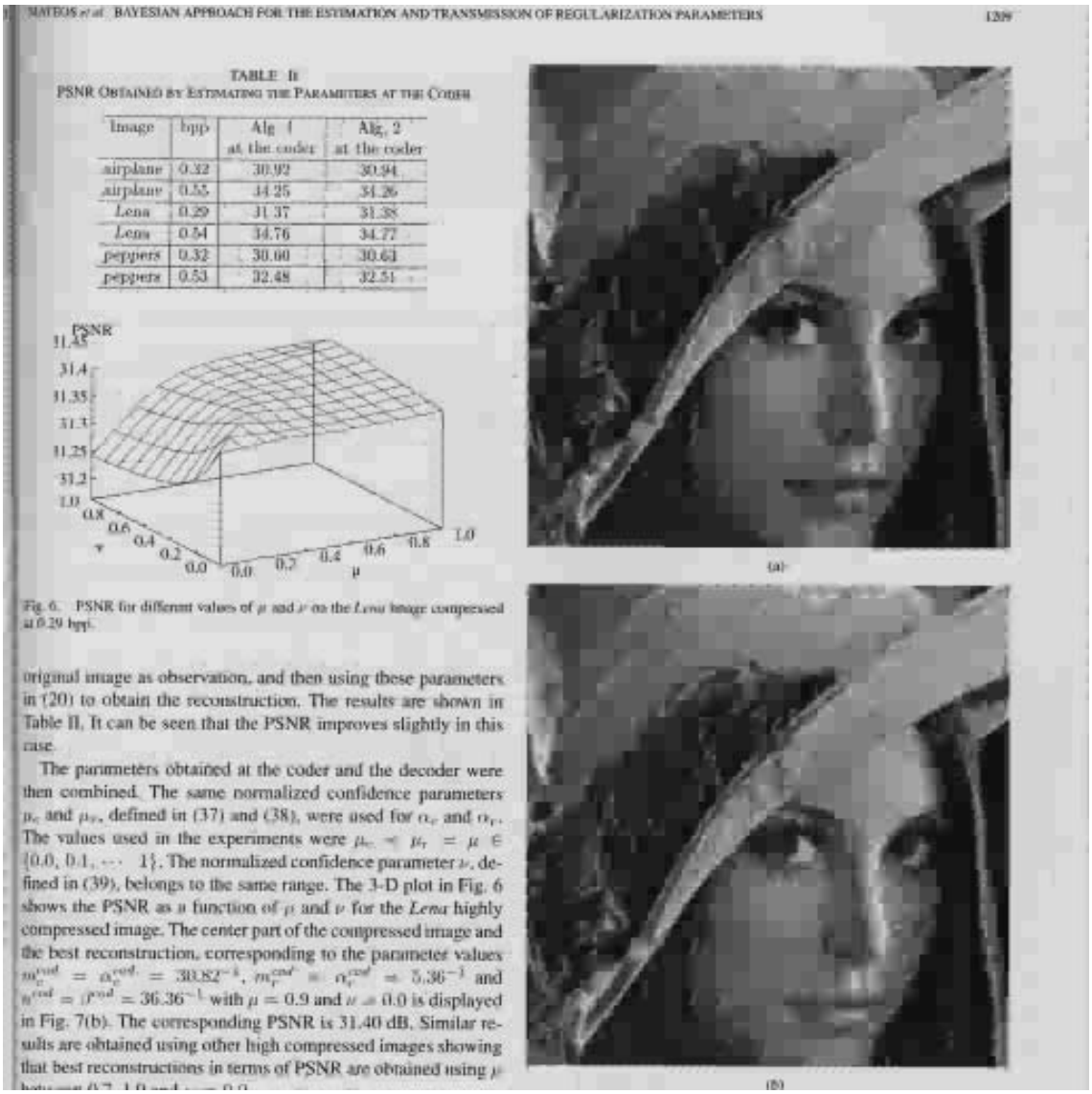
Figura E.20: PP1205 compressed at 0.5 bpp by 2D-MMP. PSNR = 27.3 dB.

text images, that can be viwed as an image almost uniformely populated with edges. Figures E.20, E.21 and E.22 ilustrtes the performance with PP1205 at 0.5 bits/pixel for the 2D-MMP, 2D-UMMP and SPIHT algorithms respectively.

With the compound image, PP1209, the MMP also outperforms all other coders, as can be seen in figure E.12. This image is half text and grafics and half continuous gray scale images (two compressed versions of LENA). From the results obtained for the other images we would expect that MMP would outperform SPIHT in the text regions but not in the regions of the two LENAs. Nevertheless, the advantage in performance in the text region overcomes the loss in the images region, and the net result is that MMP performs better with the compound. Figures

Figura E.21: PP1205 compressed at 0.5 bpp by 2D-UMMP. PSNR = 24.3 dB.

Figura E.22: PP1205 compressed at 0.5 bpp by SPIHT. PSNR = 25.2 dB.



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

TABLE II
PSNR OBTAINED BY ESTIMATING THE PARAMETERS AT THE CODER

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

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

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

The parameters obtained at the coder and the decoder were then combined. The same normalized confidence parameters μ_c and μ_r, defined in (37) and (38), were used for α_c and α_r. The values used in the experiments were $\mu_c = \mu_r = \mu \in \{0.0, 0.1, \cdots 1\}$. The normalized confidence parameter ν, defined in (39), belongs to the same range. The 3-D plot in Fig. 6 shows the PSNR as a function of μ and ν for the *Lena* highly compressed image. The center part of the compressed image and the best reconstruction, corresponding to the parameter values $m_c^{cod} = \alpha_c^{cod} = 30.82^{-1}$, $m_r^{cod} = \alpha_r^{cod} = 5.36^{-1}$ and $\eta^{cod} = \beta^{cod} = 36.36^{-1}$ 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 μ

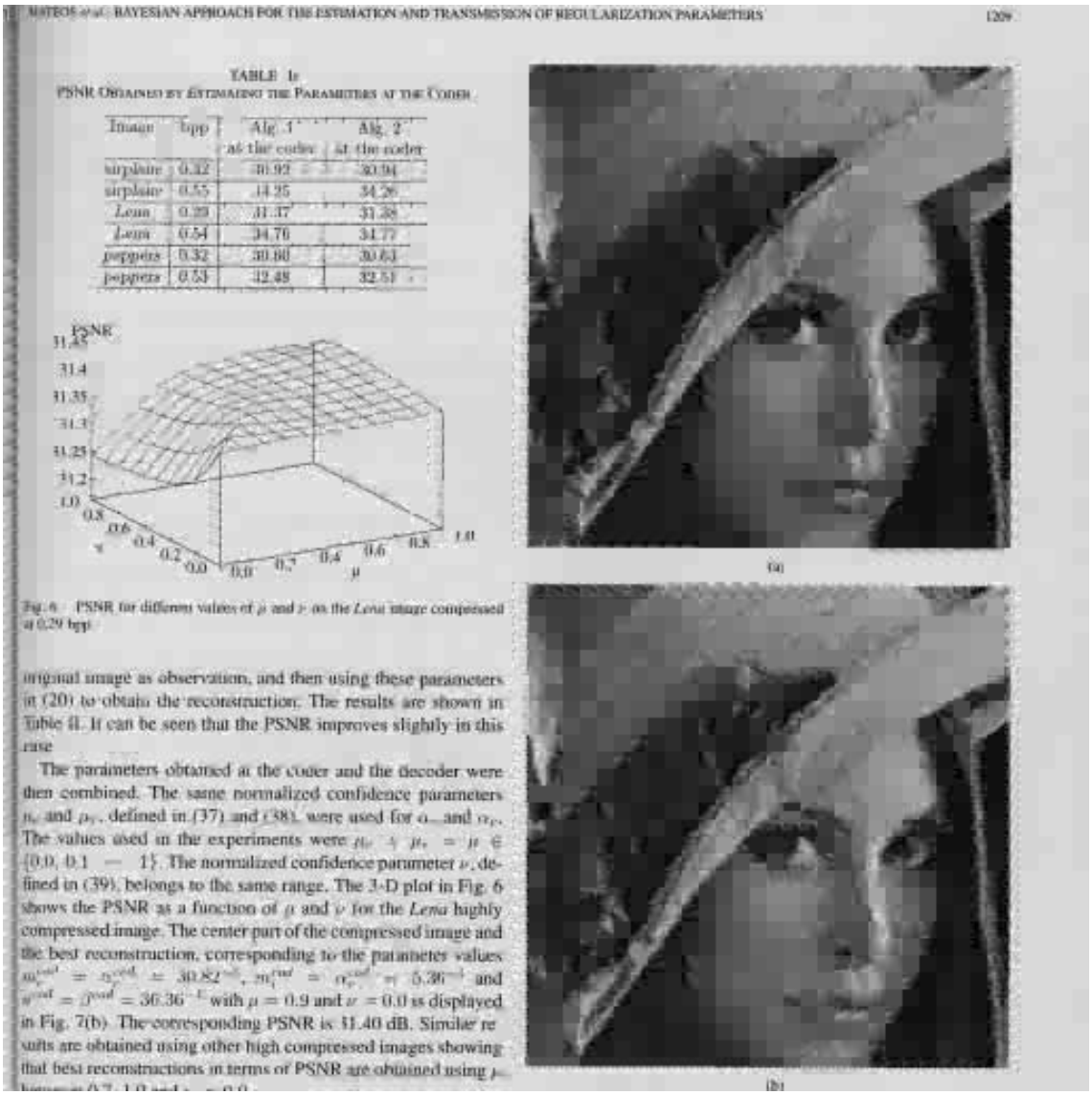
(a)

(b)

Figura E.23: PP1209 compressed at 0.5 bpp by 2D-MMP. PSNR = 29.0 dB.

E.23, E.24 and E.25 ilustrtes the performance with PP1209 at 0.5 bits/pixel for the 2D-MMP, 2D-UMMP and SPIHT algorithms respectively.

In the memoryless Gaussian case, we can see from figure E.16 that the best performer is the VQ with random codebook of size 8192. We expect the performance of MMP to become closer to this if we can improve the entropy encoding of the flags, which provide information about the segmentation tree (as was pointed out in the FD-MMP algorithm analysis). Although this source is one-dimensional, we used the 2D-versions of MMP and UMMP to highlight the universal character of the algorithms, that perform well with these two very different sources. LLZ outperforms MMP at rates above 1.5 bits/symbol, but its performance is below that



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

TABLE Ic
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	34.29
Lena	0.29	31.37	31.38
Lena	0.54	34.76	34.77
peppers	0.32	30.60	30.63
peppers	0.53	32.48	32.51

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

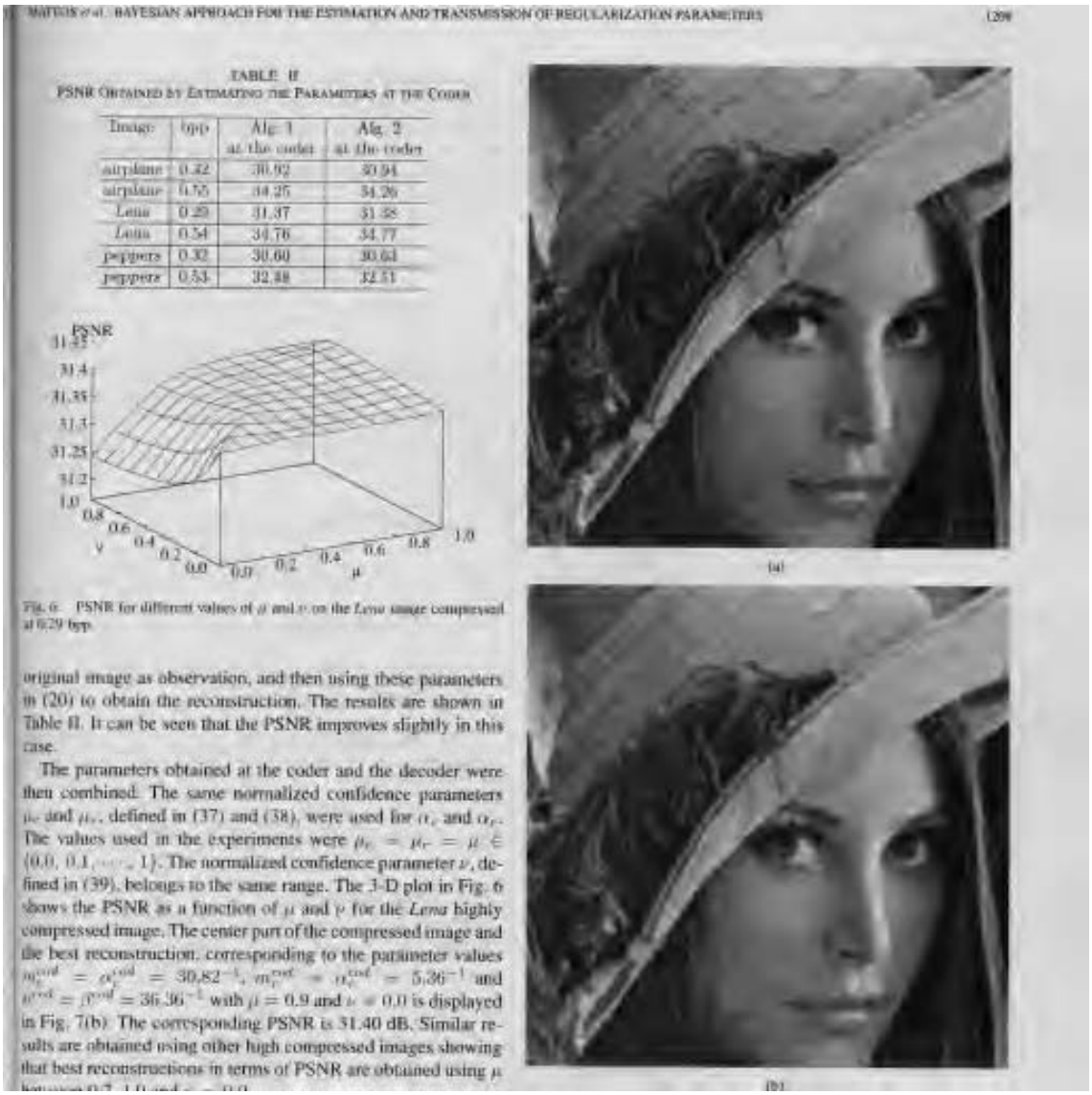
original image as observation, and then using these parameters in (20) to obtain the reconstruction. The results are shown in Table II. It can be seen that the PSNR improves slightly in this case.

The parameters obtained at the coder and the decoder were then combined. The same normalized confidence parameters μ_s and μ_β, defined in (37) and (38), were used for α_s and α_β. The values used in the experiments were $\mu_\alpha = \mu_s = \mu \in \{0.0, 0.1, \ldots, 1\}$. The normalized confidence parameter ν, defined in (39), belongs to the same range. The 3-D plot in Fig. 6 shows the PSNR as a function of μ and ν for the *Lena* highly compressed image. The center part of the compressed image and the best reconstruction, corresponding to the parameter values $\alpha_s^{cod} = \alpha_s^{dec} = 30.82^{-1}$, $m_1^{cod} = \alpha_\beta^{cod} = 5.36^{-1}$ and $\beta^{cod} = \beta^{cod} = 36.36^{-1}$ with $\mu = 0.9$ and $\nu = 0.0$ is displayed in Fig. 7(b). The corresponding PSNR is 31.40 dB. Similar results are obtained using other high compressed images showing that best reconstructions in terms of PSNR are obtained using μ [illegible]

(a)

(b)

Figura E.24: PP1209 compressed at 0.5 bpp by 2D-UMMP. PSNR = 26.7 dB.



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

TABLE II
PSNR OBTAINED BY ESTIMATING THE PARAMETERS AT THE CODER

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

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

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

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

(a)

(b)

Figura E.25: PP1209 compressed at 0.5 bpp by SPIHT. PSNR = 28.7 dB.

of the random VQ at all rates. Therefore as the performance of MMP approaches the VQ we expect it to outperform LLZ even at high rates. We believe that the good performance of MMP at low rates, when compared to the LLZ, is due to the multiscale approach.

The MMP and UMMP algorithms described so far create their segmentation trees based on local distortion decisions. There is no guarantee that the corresponding trees are optimal in a rate-distortion sense. In the next appendix we address this issue, studying methods to optimize the segmentation tree.

Apêndice F

Rate-distortion optimization on MMP

The MMP algorithm, introduced in appendix E, was controlled by d^*, a target distortion parameter. The segmentation tree was built based on local decisions regarding distortion calculations. That is, the subdivision of a segment stopped when the distortion incurred by the approximation was not greater than d^* times the length of the segment. We can expect to improve the performance if we make decisions based on global criteria rather than local ones. In this appendix we describe an R-D optimization technique applied to the MMP algorithm.

F.1 Optimization of the segmentation tree

The MMP is a lossy data compression algorithm that uses a recursive parsing procedure and a set of $K = \log_2(N)$ dictionaries $\mathcal{D}^{(k)}$ to encode the data. An input vector $\mathbf{X} = \begin{pmatrix} x_0 & \dots & x_{N-1} \end{pmatrix}$ is parsed in L segments, $\mathbf{X} = \begin{pmatrix} \mathbf{X}^{l_0} & \dots & \mathbf{X}^{l_{L-1}} \end{pmatrix}$ of lengths $\ell(\mathbf{X}^{l_m})$, $m = 0, 1, \dots, L-1$. Each segment $\mathbf{X}^{l_m}$ is encoded using one element of the corresponding dictionary $\mathcal{D}^{(k_m)} = \left\{ \mathbf{S}_0^{(k_m)}, \dots, \mathbf{S}_{M_{k_m}-1}^{(k_m)} \right\}$, where $k_m = \log_2\left(\ell(\mathbf{X}^{l_m})\right)$. That is, the input vector is approximated as $\hat{\mathbf{X}} = \begin{pmatrix} \mathbf{S}_{i_0}^{(k_0)} & \dots & \mathbf{S}_{i_{L-1}}^{(k_{L-1})} \end{pmatrix}$. The lengths $\ell(\mathbf{X}^{l_m})$ of the segments can be specified by a binary segmentation tree $\mathcal{S}$ as in figure F.1.

Each node of the tree, denoted by n_l, can have either zero or two children. The two children of node n_l are n_{2l+1} and n_{2l+2}. A node with no child is a leaf.

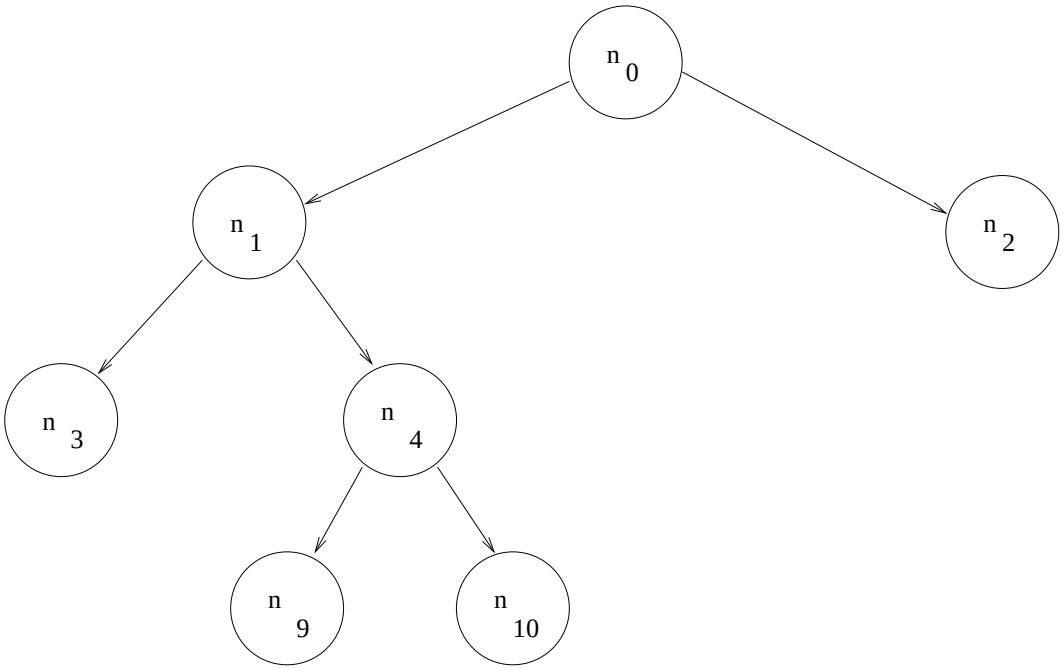


Figura F.1: A binary segmentation tree.

The root node n_0 of the segmentation tree corresponds to a segment of length N. Its two children, n_1 and n_2 are associated to the two segments of length $N/2$. A node at depth p represents a segment of length $2^{-p}N$. Therefore, the segmentation is given by the leaves, and the length $\ell(\mathbf{X}^{l_m})$ of the segment $\mathbf{X}^{l_m}$ is the length of the corresponding leaf node in the tree. For example, the segmentation represented by the tree in figure F.1 is $\mathbf{X} = \left(\begin{array}{cccc} \mathbf{X}^3 & \mathbf{X}^9 & \mathbf{X}^{10} & \mathbf{X}^2 \end{array} \right)$ and the lengths of the segments are respectively: $N/4, N/8, N/8, N/2$.

Each node n_l is associated to a segment of the input vector $\mathbf{X}^l$ that is best represented by an element $\mathbf{S}_{i_l}^{(k_l)}$, where $k_l = \log_2(\ell(\mathbf{X}^l))$. Therefore we can evaluate the distortion:

$$\begin{aligned} D(n_l) &= \|\mathbf{X}^l - \mathbf{S}_{i_l}^{(k_l)}\|, \\ k_l &= \log_2(\ell(\mathbf{X}^l)) \end{aligned} \tag{F.1}$$

The rate $R(n_l)$ is the rate needed to specify the index i_l, that is:

$$R(n_l) = -\log_2(Pr(i_l|k_l)) \tag{F.2}$$

Where $Pr(i_l|k_l)$ is the probability of occurrence of index i_l in the dictionary of scale k_l.

The overall distortion is:

$$D(\mathcal{S}) = \sum_{n_l \in \mathcal{S}_\mathcal{L}} D(n_l) \tag{F.3}$$

Where $\mathcal{S}_\mathcal{L}$ is the set of leaf nodes of $\mathcal{S}$. The amount of bits needed to encode this approximation is the rate $R(\mathcal{S})$, and is given by:

$$R(\mathcal{S}) = R_t(\mathcal{S}) + \sum_{n_l \in \mathcal{S}_\mathcal{L}} R(n_l) \tag{F.4}$$

where $R_t(\mathcal{S})$ is the rate required to specify the segmentation tree.

The best segmentation $\mathcal{S}^*$, in an R-D sense, leads to the minimum rate $R(\mathcal{S})$ given that the distortion $D(\mathcal{S})$ is no greater than a target distortion D^* or, alternatively, the minimum distortion $D(\mathcal{S})$ at rate R^*. This is a constrained minimization problem stated as:

$$\begin{aligned} \mathcal{S}^* &= \arg\min_{\mathcal{S} \in \mathcal{S}_{R^*}} D(\mathcal{S}), \\ \mathcal{S}_{R^*} &= \{\mathcal{S} : R(\mathcal{S}) = R^*\} \end{aligned} \tag{F.5}$$

To find $\mathcal{S}^*$ we can find the solution to a related unconstrained problem introducing a Lagrange multiplier λ. It is well known that if we find the minimum of the Lagrangian cost $J(\mathcal{S}) = D(\mathcal{S}) + \lambda R(\mathcal{S})$ [2], we also find the solution to the constrained problem when we choose $R(\lambda) = R^*$. That is:

$$\begin{aligned} \mathcal{S}^* &= \arg\min_{\mathcal{S}} J(\mathcal{S}) \\ &= \arg\min_{\mathcal{S}} \left(\sum_{n_l \in \mathcal{S}_\mathcal{L}} D(n_l) \right) + \lambda \left(R_t(\mathcal{S}) + \sum_{n_l \in \mathcal{S}_\mathcal{L}} R(n_l) \right) \\ &= \arg\min_{\mathcal{S}} \lambda R_t(\mathcal{S}) + \sum_{n_l \in \mathcal{S}_\mathcal{L}} (D(n_l) + \lambda R(n_l)) \\ &= \arg\min_{\mathcal{S}} \lambda R_t(\mathcal{S}) + \sum_{n_l \in \mathcal{S}_\mathcal{L}} J(n_l) \end{aligned} \tag{F.6}$$

where $J(n_l) = D(n_l) + \lambda R(n_l)$.

A sub-tree $\mathcal{S}(n_l)$ of $\mathcal{S}$ at node n_l is the binary tree with all the nodes of $\mathcal{S}$ having n_l as the root node. Figure F.2 illustrates the sub-tree $\mathcal{S}(n_4)$ of the binary

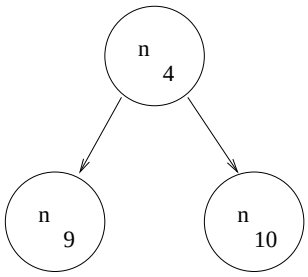


Figura F.2: The sub-tree $\mathcal{S}(n_4)$.

tree in figure F.1. We denote $\mathcal{S} - \mathcal{S}(n_l)$ the tree obtained from $\mathcal{S}$ by pruning the sub-tree $\mathcal{S}(n_l)$.

If the Lagrangian costs $J(n_l)$, associated with the approximation of each segment $\mathbf{X}^l$, are independent, then the Lagrangian cost of two sub-trees $J(\mathcal{S}(n_l))$ and $J(\mathcal{S}(n_m))$ are also independent, as long as all nodes of both sub-trees are different. Then a fast search algorithm can be implemented considering that if $J(n_l) \leq J(\mathcal{S}(n_{2l+1})) + J(\mathcal{S}(n_{2l+2}))$ then the sub-trees $\mathcal{S}(n_{2l+1})$ and $\mathcal{S}(n_{2l+2})$ must be pruned from $\mathcal{S}$ in order to decrease the cost. Unfortunately this is not the case with MMP, since the costs $J(n_l)$ are coupled by the dictionary updating procedure. However, if the initial dictionaries are large enough, the contribution to the minimization of $J(n_l)$ due to the dictionary update can be negligible. In practical MMP implementations, we tend to use an upper limit to the size M of the input vector $\mathbf{X}$ to deal with the finite amount of memory available. Therefore the input data is broken in blocks of size M that are sequentially processed by MMP. Although not true for the first blocks, the dictionary eventually grows large enough for the Lagrangian costs $J(n_l)$ to be almost decoupled.

We can find an approximate R-D-optimal solution considering that the costs $J(n_l)$ are independent. The search can be implemented by iteratively pruning the binary segmentation tree as follows [27]:

step 1 Initialize $\mathcal{S}$ as the full tree of depth $\log_2(M) + 1$.

step 2 Make $J_i = \infty, i = M - 1, M, \ldots, 2M - 2$.

step 3 Make $p = \log_2(M)$.

step 4 For each node $n_l \in \mathcal{S}$ at depth p, that is $l \in [2^{p-1} - 1, 2^p - 2]$, evaluate $J_l = J(n_l) + \lambda R_{1_l}$ where $J(n_l)$ is the cost to represent the input segment

associated to the node n_l and R_{1_l} is the rate needed to indicate that the node n_l is a leaf. If $J_l \leq J_{2l+1} + J_{2l+2} + \lambda R_{0_l}$ then prune nodes n_{2l+1} and n_{2l+2} (R_{0_l} is the rate needed to indicate splitting, and J_{2l+1}, J_{2l+2} were evaluated in the previous iteration with $p+1$). Otherwise, the cost of node n_l is updated with $J_{2l+1} + J_{2l+2} + \lambda R_{0_l}$. Make $p = p - 1$.

step 5 Repeat step 4 until $p = 0$.

The algorithm above finds the optimal segmentation tree for a block of input data and is consistent with the way the MMP algorithm, as described in section E, outputs the information regarding the segmentation (with flags to indicate "match" and "split").

If we want to use relatively large blocklengths or the dictionary is too small (as happens at very low rates), we can modify the algorithm to compute the impact of the dictionary-updating procedure. The dictionary is updated by the inclusion of the concatenation of previously encoded segments. Therefore, if we choose to prune a sub-tree, the impact in the cost is not restricted to that sub-tree, but can affect all nodes that are to the right of the node. That is, if we prune a sub-tree, we might remove from the dictionary an element that would otherwise be used later to approximate an input segment, therefore increasing the cost. The idea is to prune a sub-tree only if the potential increase in the cost of subsequent nodes, due to the removal of some vectors from the dictionary, is not greater than the reduction in the cost provided by the pruning. The modified algorithm is described below.

step 1 Initialize $\mathcal{S}$ as the full tree of depth $\log_2(M) + 1$.

step 2 Make $J_i = \infty, i = M-1, M, \ldots, 2M-2$.

step 3 Make $p = \log_2(M)$.

step 4 For each node $n_l \in \mathcal{S}$ at depth p, that is $l \in [2^{p-1} - 1, 2^p - 2]$, evaluate $J_l = J(n_l) + \lambda R_{1_l}$ where $J(n_l)$ is the cost to represent the input segment associated to the node n_l and R_{1_l} is the rate needed to indicate that the node n_l is a leaf. Evaluate $\Delta J_l = \sum_{n_r \in \mathcal{S} - \mathcal{S}(n_l)} J(n_r) - \sum_{n_r \in \mathcal{S} - \mathcal{S}(n_l)} J'(n_r)$, where $J'(n_l)$ is computed using the dictionary without $\hat{\mathbf{X}}^l = \begin{pmatrix} \hat{\mathbf{X}}^{2j+1} & \hat{\mathbf{X}}^{2j+2} \end{pmatrix}$. If $J_l - J_{2l+1} - J_{2l+2} - \lambda R_{0_l} \leq \Delta J_l$ then prune nodes n_{2l+1} and n_{2l+2} (R_{0_l} is

the rate needed to indicate splitting, and J_{2l+1}, J_{2l+2} were evaluated in the previous iteration with $p+1$). Otherwise, the cost of node n_l is updated with $J_{2l+1} + J_{2l+2} + \lambda R_{0_l}$. Make $p = p - 1$.

step 5 Repeat step 4 until $p = 0$.

step 6 If $\mathcal{S}_0 = \mathcal{S}$ then the optimization is done. Otherwise, make $\mathcal{S}_0 = \mathcal{S}$ and go to step 3.

It is interesting to consider why ΔJ_l is evaluated with all the nodes, since only the leaf nodes contribute to the total cost. The idea of the extended algorithm is to prune only when we are sure that the cost will not increase. The computation of ΔJ_l must be conservative because we don't know, at the time we are making a decision at node n_l, which nodes will be the leaves (the current leaves may be pruned later). When we evaluate $\Delta J_l < J_l - J_{2l+1} - J_{2l+2} - \lambda R_{0_l}$, we decide not to prune. However, the nodes that affected the decision might be pruned later. Therefore, the whole procedure must be repeated to improve the segmentation, until convergence.

F.2 Experimental results

The algorithm for R-D optimization of the segmentation tree described in the previous section was implemented with 2D-MMP and we call the resultant algorithm 2D-MMP-RD.

In our implementation, a segmentation tree $\mathcal{S}$ is represented by a segmentation code $\mathbf{C}^{(\mathcal{S})} \in \{0,1\}^*$ that is a sequence of segmentation flags. For example, the segmentation tree in figure F.1 corresponds to the segmentation code $\mathbf{C}^{(\mathcal{S})} = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$. This code is generated as follows: Firstly we use $C_0^{(\mathcal{S})} = 0$ to indicate that the node n_0 has two children n_1 and n_2. Then we append to this bit the segmentation code to the sub-tree $\mathcal{S}(n_1)$ followed by the segmentation code for the sub-tree $\mathcal{S}(n_2)$, that is, $\mathbf{C}^{(\mathcal{S})} = \begin{pmatrix} 0 & \mathbf{C}^{(\mathcal{S}(n_1))} & \mathbf{C}^{(\mathcal{S}(n_2))} \end{pmatrix}$. The code for $\mathbf{C}^{(\mathcal{S}(n_1))}$ is built in a similar fashion, that is: We use $C_0^{(\mathcal{S}(n_1))} = 0$ to indicate that the node n_1 has two children n_3 and n_4. Then we append to this bit the segmentation code to the sub-tree $\mathcal{S}(n_3)$ followed by the segmentation code for the sub-tree $\mathcal{S}(n_4)$, that is, $\mathbf{C}^{(\mathcal{S}(n_1))} = \begin{pmatrix} 0 & \mathbf{C}^{(\mathcal{S}(n_3))} & \mathbf{C}^{(\mathcal{S}(n_4))} \end{pmatrix}$. The node n_3 is a leaf

therefore the segmentation code for $\mathcal{S}(n_3)$ is $\mathbf{C}^{(\mathcal{S}(n_3))} = 1$. The code for $\mathcal{S}(n_4)$ is $\mathbf{C}^{(\mathcal{S}(n_4))} = \begin{pmatrix} 0 & \mathbf{C}^{(\mathcal{S}(n_9))} & \mathbf{C}^{(\mathcal{S}(n_{10}))} \end{pmatrix}$. The nodes n_9, n_{10}, and n_2 are leaves, therefore their respective segmentation codes are $\mathbf{C}^{(\mathcal{S}(n_9))} = \mathbf{C}^{(\mathcal{S}(n_{10}))} = \mathbf{C}^{(\mathcal{S}(n_2))} = 1$. By substitution we can easily determine the segmentation code for the complete tree $\mathcal{S}$.

The optimized 2D-MMP-RD algorithm is:

Let:

- $\mathbf{X}$ be an $(N \times M) = (2^K \times 2^L)$ matrix.
- $\mathcal{D}^{(k,l)} = \{\mathbf{s}_0^{(k,l)}, \ldots, \mathbf{s}_{I-1}^{(k,l)}\}$,
  $(k,l) = (0,0), (1,0), (1,1) \ldots, (K,L)$ a set of $K + L + 1$ dictionaries with $I^{(k,l)}$ matrices of size $2^k \times 2^l$ each. This dictionaries must be initialized to $\mathcal{D}^{(k,l)} = \mathcal{D}_0^{(k,l)}$.
- d^* be a target distortion.
- $T_{N,M}[\mathbf{X}]$ be a scale transformation that maps the matrix $\mathbf{X}$ in a matrix of size $N \times M$.
- $\mathcal{S}$ be the optimum segmentation tree and $\mathbf{C}^{(\mathcal{S})}$ its segmentation code.
- m be a pointer initialized to $m = 0$.

Procedure $\hat{\mathbf{X}}^j = \text{encode}(\mathbf{X}^j, \mathbf{C}^{(\mathcal{S})})$:

step 1 find index i_j in the dictionary of the same scale (k, l) of $\mathbf{X}^j$
such that $\|\mathbf{X}^j - \mathbf{s}_{i_j}^{(k,l)}\|$ is minimum and make $\hat{\mathbf{X}}^j = \mathbf{s}_{i_j}^{(k,l)}$
make $m = m + 1$.

step 2 `if` $k = l = 0$, output index i_j and return $\hat{\mathbf{X}}^j$.
`else` got to step 3.

step 3 `if` $C_{m-1}^{(\mathcal{S})} = 1$ `then` output flag '1', output index i_j and return $\hat{\mathbf{X}}^j$.
`else` go to step 4. .

step 4 output flag '0'.

step 5 `if` $k > l$ split $\mathbf{X}^j = \begin{pmatrix} \mathbf{X}^{2j+1} \\ \mathbf{X}^{2j+2} \end{pmatrix}$,
where $\mathbf{X}^{2j+1}$ and $\mathbf{X}^{2j+2}$ are $2^{k-1} \times 2^l$
`else` split $\mathbf{X}^j = \begin{pmatrix} \mathbf{X}^{2j+1} & \mathbf{X}^{2j+1} \end{pmatrix}$
where $\mathbf{X}^{2j+1}$ and $\mathbf{X}^{2j+2}$ are $2^k \times 2^{l-1}$

step 6 compute $\hat{\mathbf{X}}^{2j+1} = \text{encode}(\mathbf{X}^{2j+1}, \mathbf{C}^{(\mathcal{S})})$

step 7 compute $\hat{\mathbf{X}}^{2j+2} = \text{encode}(\mathbf{X}^{2j+2}, \mathbf{C}^{(\mathcal{S})})$

step 8 `if` $k > l$ make $\hat{\mathbf{X}}^j = \begin{pmatrix} \hat{\mathbf{X}}^{2j+1} \\ \hat{\mathbf{X}}^{2j+2} \end{pmatrix}$,
`else` make $\hat{\mathbf{X}}^j = \begin{pmatrix} \hat{\mathbf{X}}^{2j+1} & \hat{\mathbf{X}}^{2j+2} \end{pmatrix}$

step 9 `for` $n = 0, 1, \ldots, K + L + 1$,
$p = \left\lfloor (\frac{n+1}{2}) \right\rfloor$, $q = \left\lfloor (\frac{n}{2}) \right\rfloor$,
$\mathcal{D}^{(p,q)} = \mathcal{D}^{(p,q)} \cup \left\{ T_{2^p, 2^q}[\hat{\mathbf{X}}^j] \right\}$
where $\lfloor x \rfloor$ is the largest integer that is smaller than or equal to x .

step 10 return $\hat{\mathbf{X}}^j$

This R-D optimized algorithm is completely compatible with the decoder of the MMP without R-D optimization, since only the decision of whether to split or not is different, not the syntax of the bitstream.

The 2D-MMP-RD was applied to lossy compress the gray-scale still images LENA, BABOON, F16, BRIDGE, AERIAL, BARBARA and GOLD 512×512. As in appendix E the images were initially divided in blocks that were sequentially processed by the algorithm. However, unlike 2D-MMP, the results always improved as we increased the size of the blocks. In our experiments, we were able to use 16×16 blocks. The maximum size of the dictionaries was limited to $|\mathcal{D}^{(k,l)}| \leq 32768$. The initial dictionary at scale 1×1 was $\mathcal{D}^{(0,0)} = \{\min(X_{n,m}), \min(X_{n,m}) + 1 \ldots, \max(X_{n,m})\}$ and the initial dictionaries at all other scales were obtained from this dictionary by the use of the scale transformation. The scale transformation was the same described in appendix D, equations D.2, D.3 and D.4. The indexes output by the algorithm and the flags were encoded by an arithmetic coder with independent models for each scale. The rates $R(n_l)$, R_{0_l} and R_{1_l} were estimated using the logarithm of the relative frequencies of occurrence of the symbols, computed by the arithmetic coder.

Figures F.3 to F.9 show the results for the test images LENA, BABOON, F16, BRIDGE, AERIAL, BARBARA, GOLD, PP1205 and PP1209 512×512. Results for the MMP without R-D optimization and for the transform coders SPIHT and JPEG are also shown.

As can be seen from the figures F.3 to F.11 the performance improved dramatically with the R-D optimization. With the pure gray-scale images, the best performer is still the SPIHT algorithm. For example, at 0.5 bits/pixel, the smallest difference in PSNR is 0.81 dB for the image AERIAL, and the largest difference is 2.70 dB for the image BARBARA. These are not bad results at all, for a pure VQ compression scheme compared to the transform coder SPIHT. For the images PP1205, pure text, and PP1209, a compound, the 2D-MMP-RD algorithm outperformed all other methods. The advantage in PSNR, at 0.5 bits/pixel is 3.23 dB for the image PP1205 and 1.23 dB for the image PP1209 when compared to the SPIHT. Figures F.12, F.13 and F.14 show the images LENA, PP1205 and pp1209 respectively, compressed by 2D-MMP-RD at 0.5 bits/pixel.

Its interesting to compare the image LENA compressed by 2D-MMP-RD, shown in figure F.12, and the version compressed by 2D-MMP in appendix E, shown in figure E.17, both at 0.50 bpp. The subjective quality of the image compressed

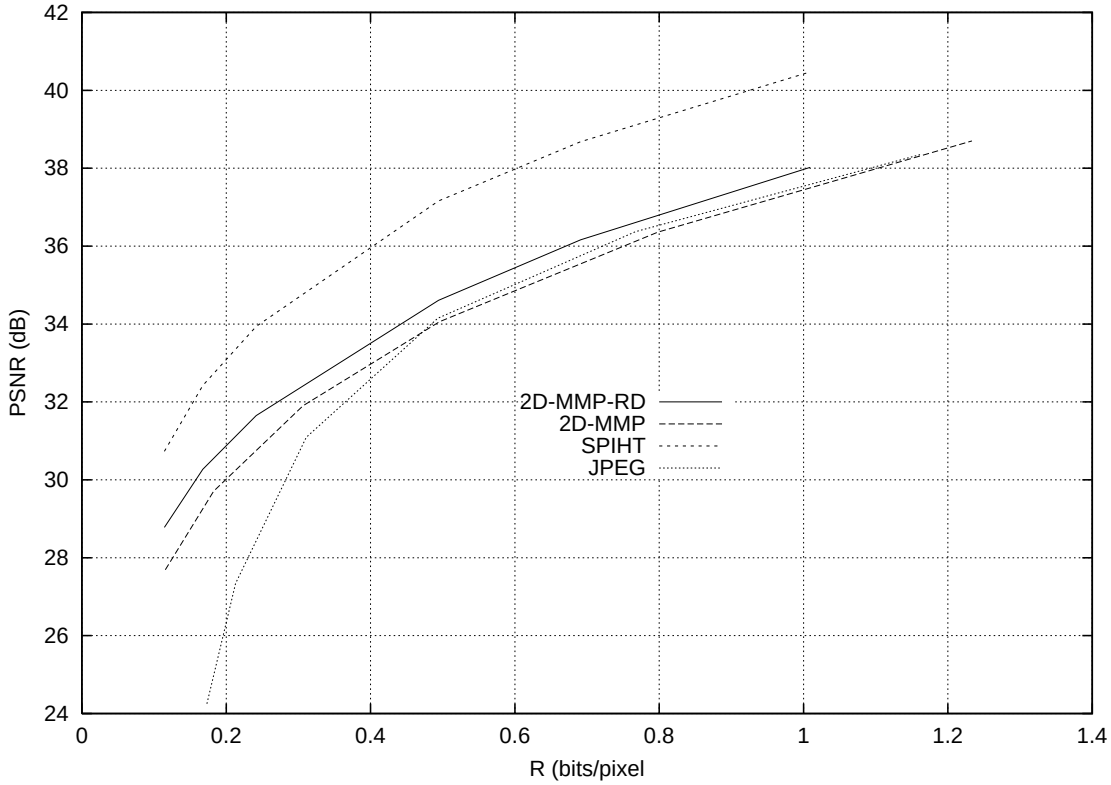


Figura F.3: R-D performance with LENA 512×512.

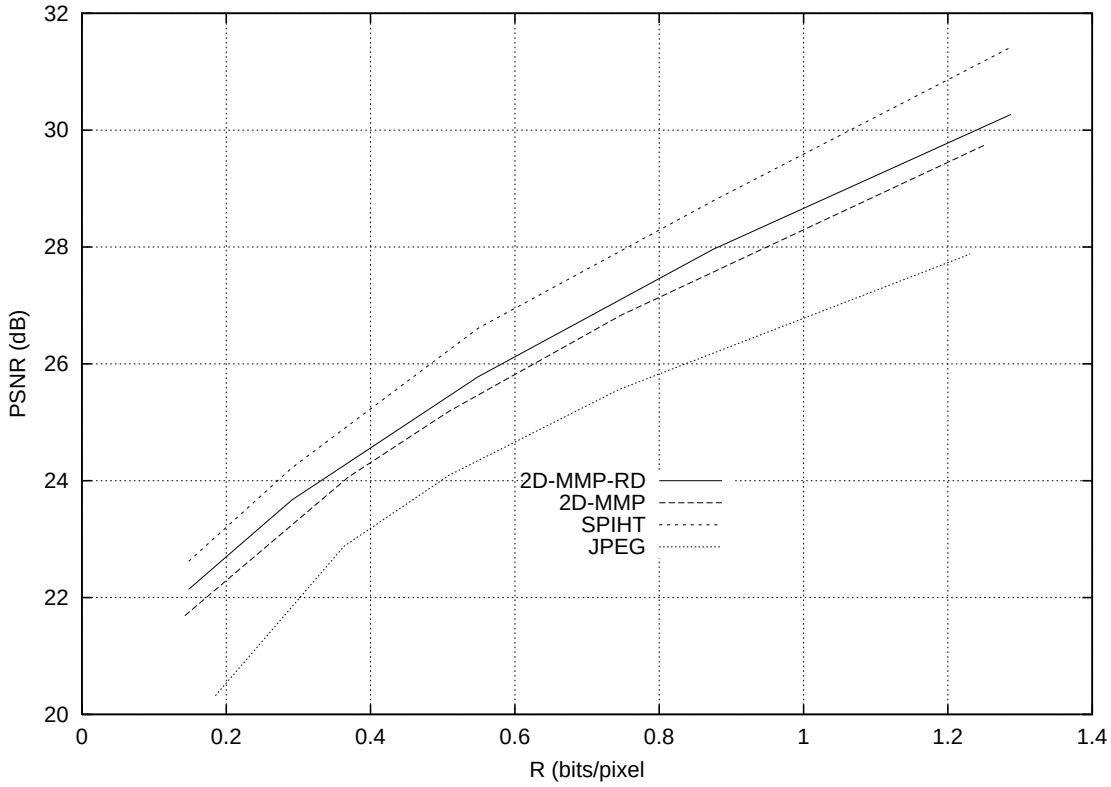


Figura F.4: R-D performance with BABOON 512×512.

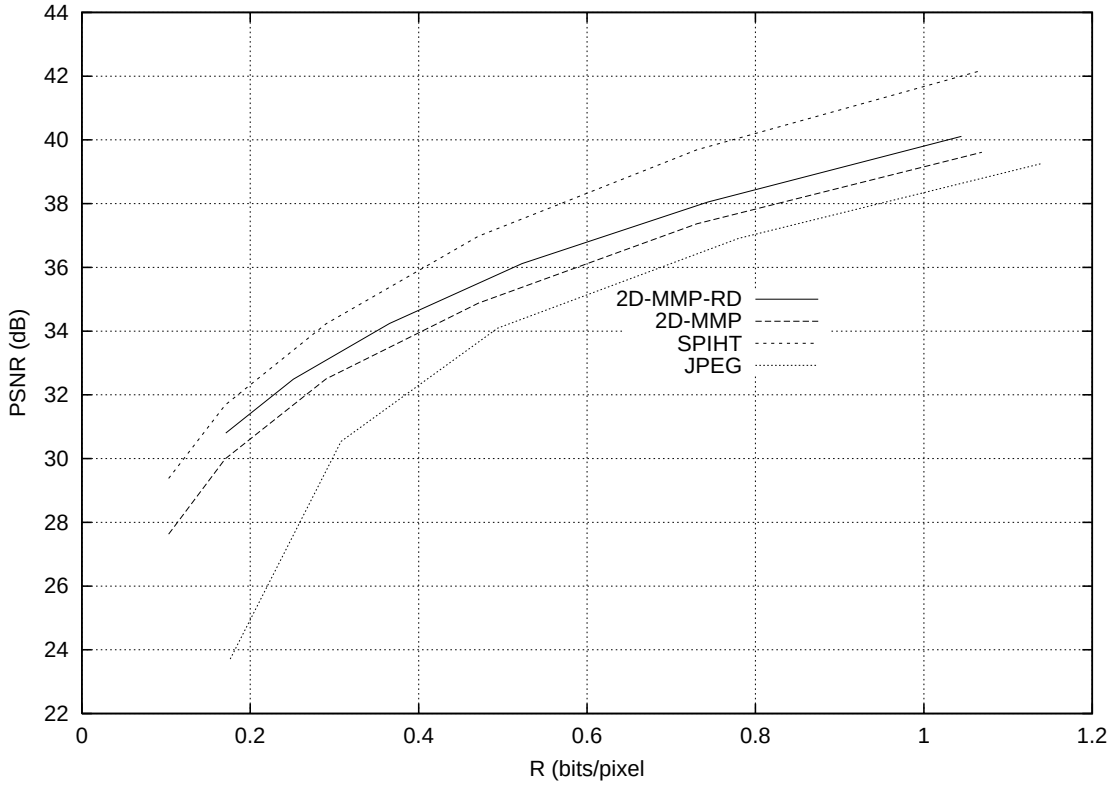


Figura F.5: R-D performance with F16 512×512.

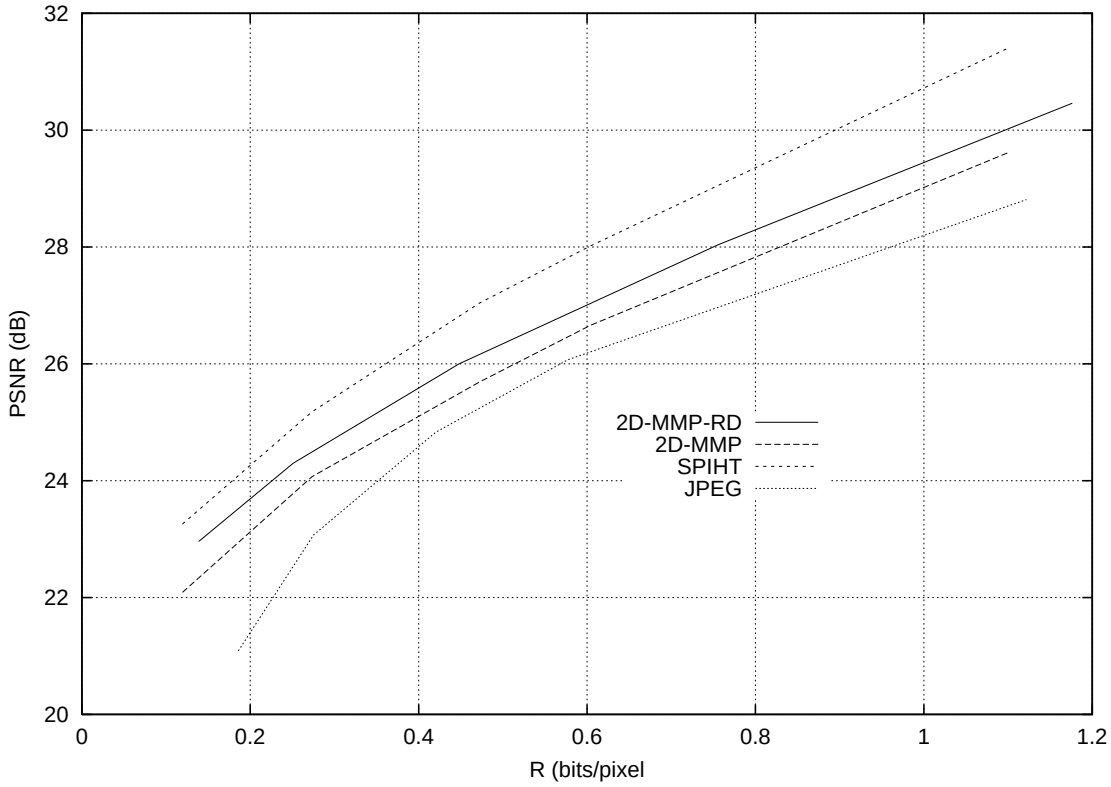


Figura F.6: R-D performance with BRIDGE 512×512.

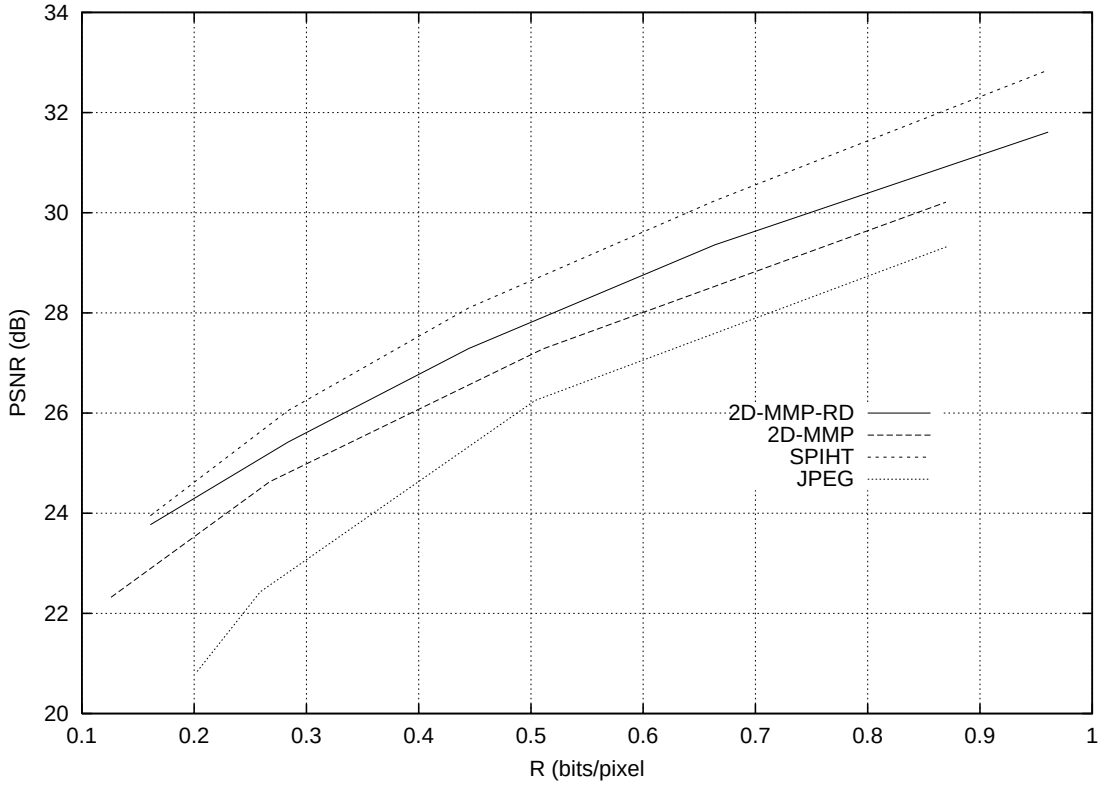


Figura F.7: R-D performance with AERIAL 512×512.

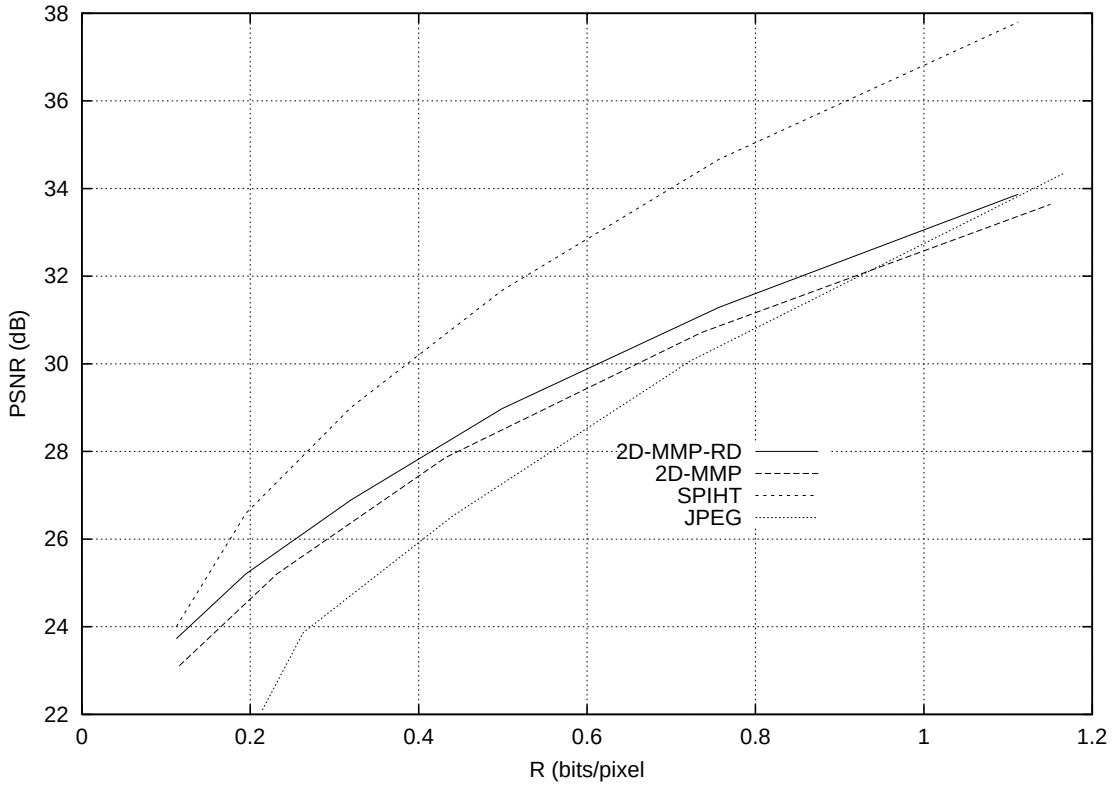


Figura F.8: R-D performance with BARBARA 512×512.

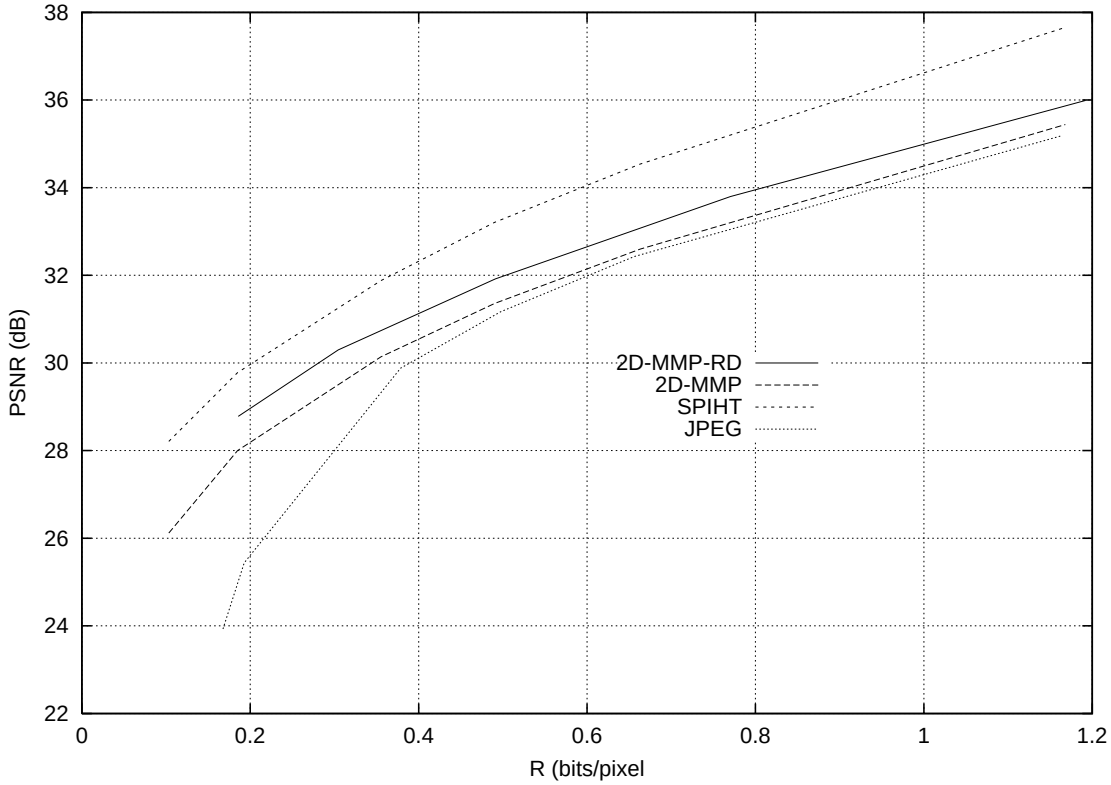


Figura F.9: R-D performance with GOLD 512×512.

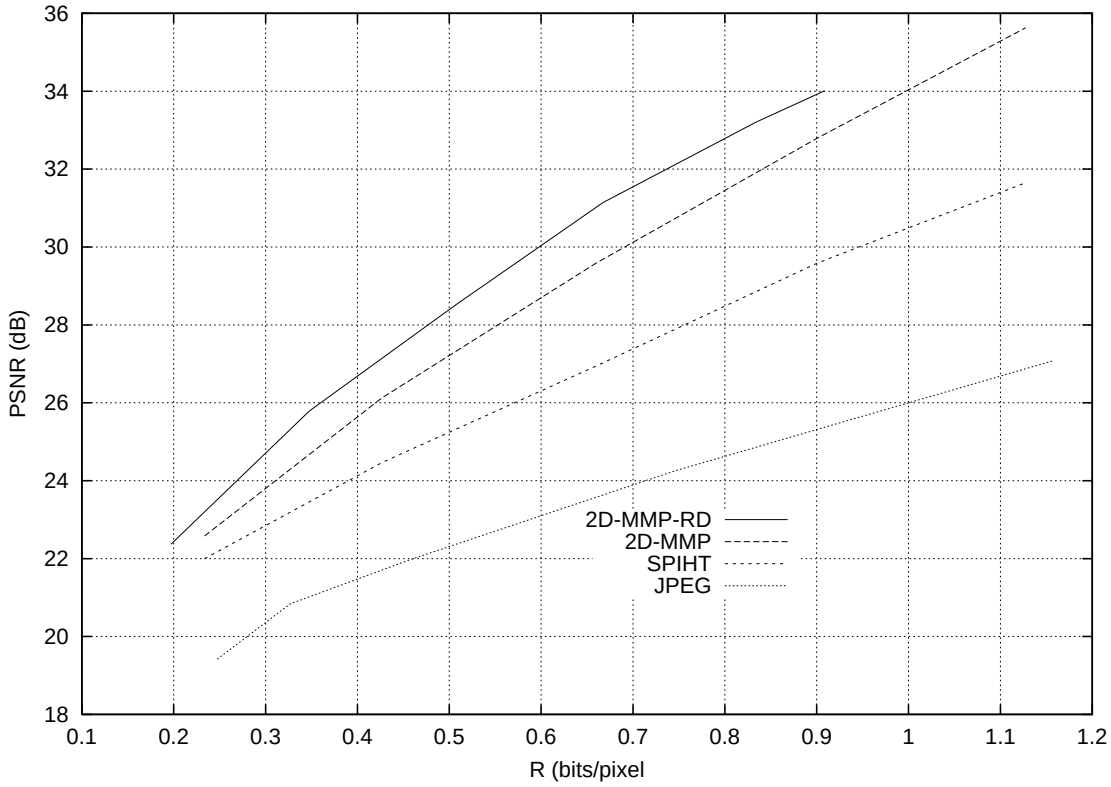


Figura F.10: R-D performance with PP1205 512×512.

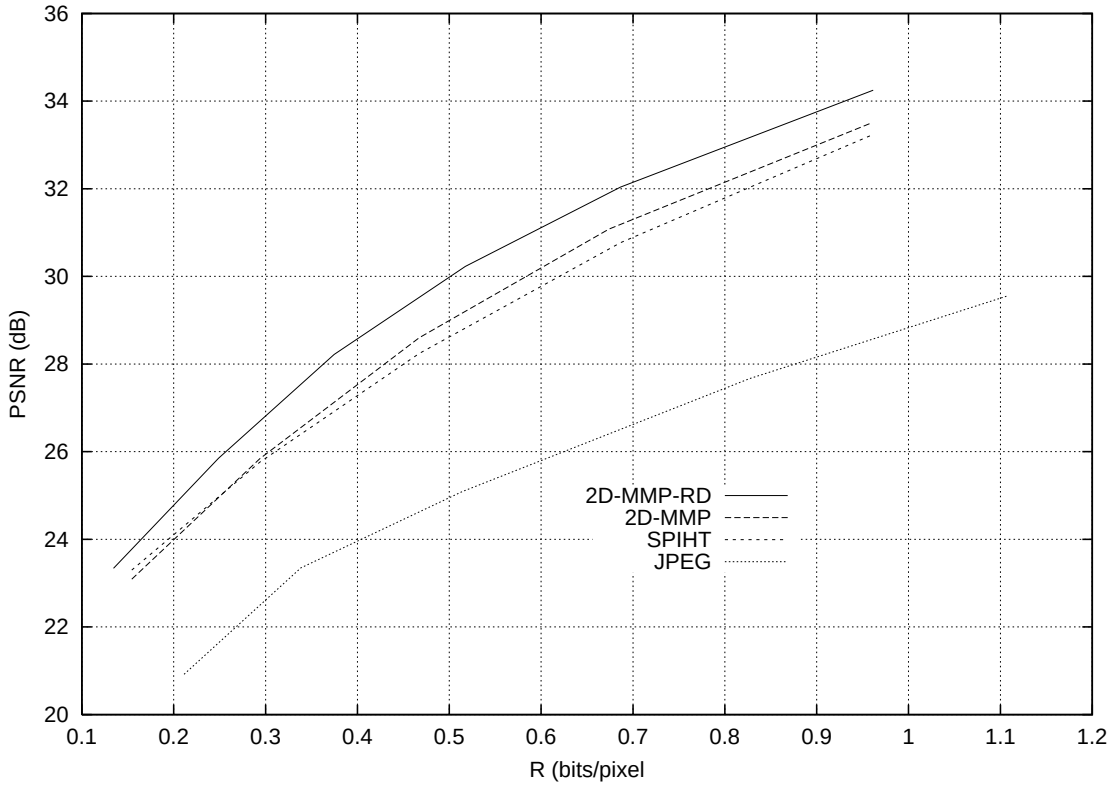


Figura F.11: R-D performance with PP1209 512×512.

by 2D-MMP-RD is clearly superior. Part of the superiority is due to the higher PSNR but the better segmentation of 2D-MMP-RD, that allowed the use of a larger maximum block-size, also lead to an image with reduced *blockiness*. In appendix G we discuss in detail the blocking effect in MMP.

Figura F.12: LENA compressed by 2D-MMP-RD at 0.5 bpp. PSNR = 34.7 dB.

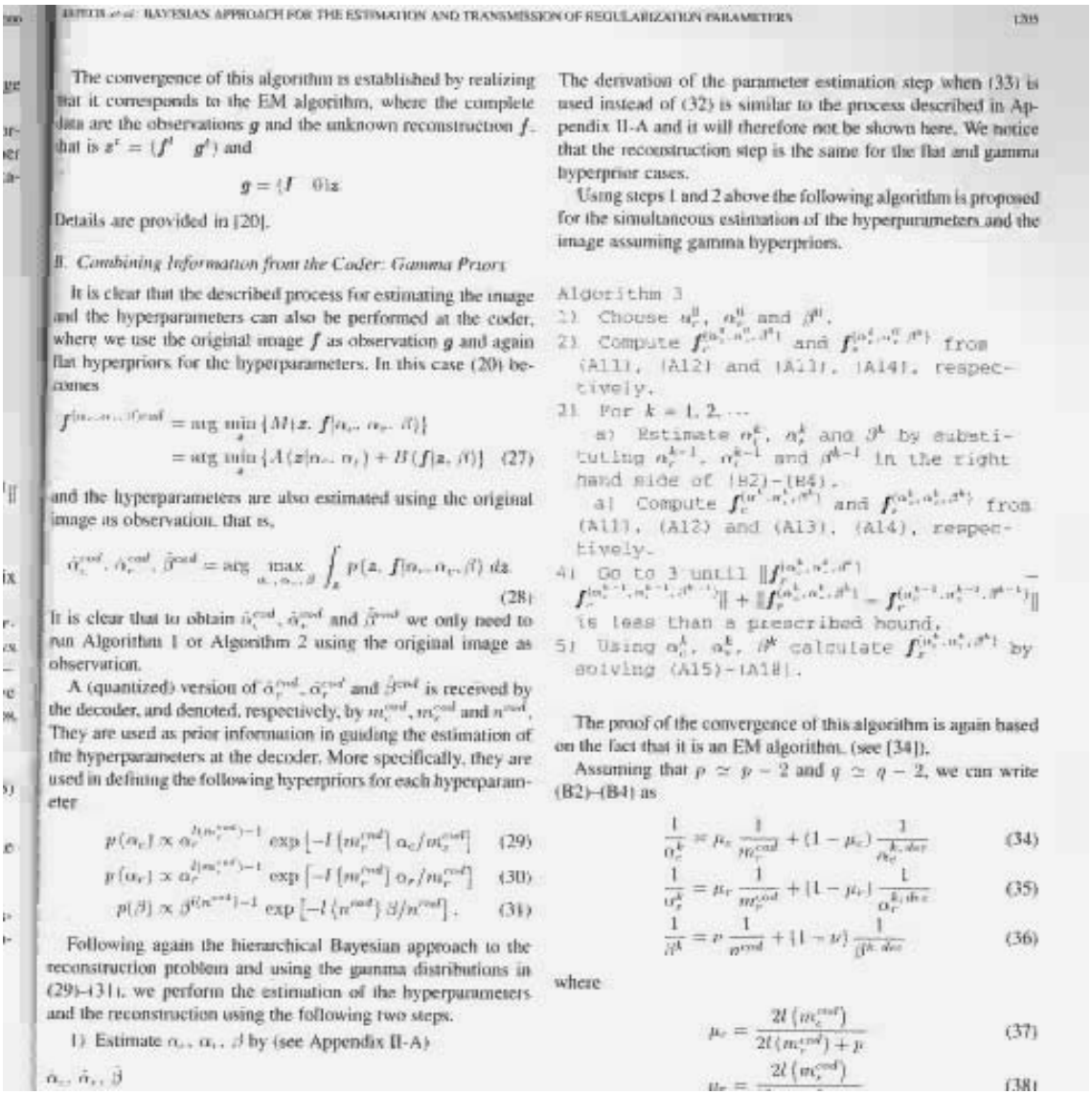
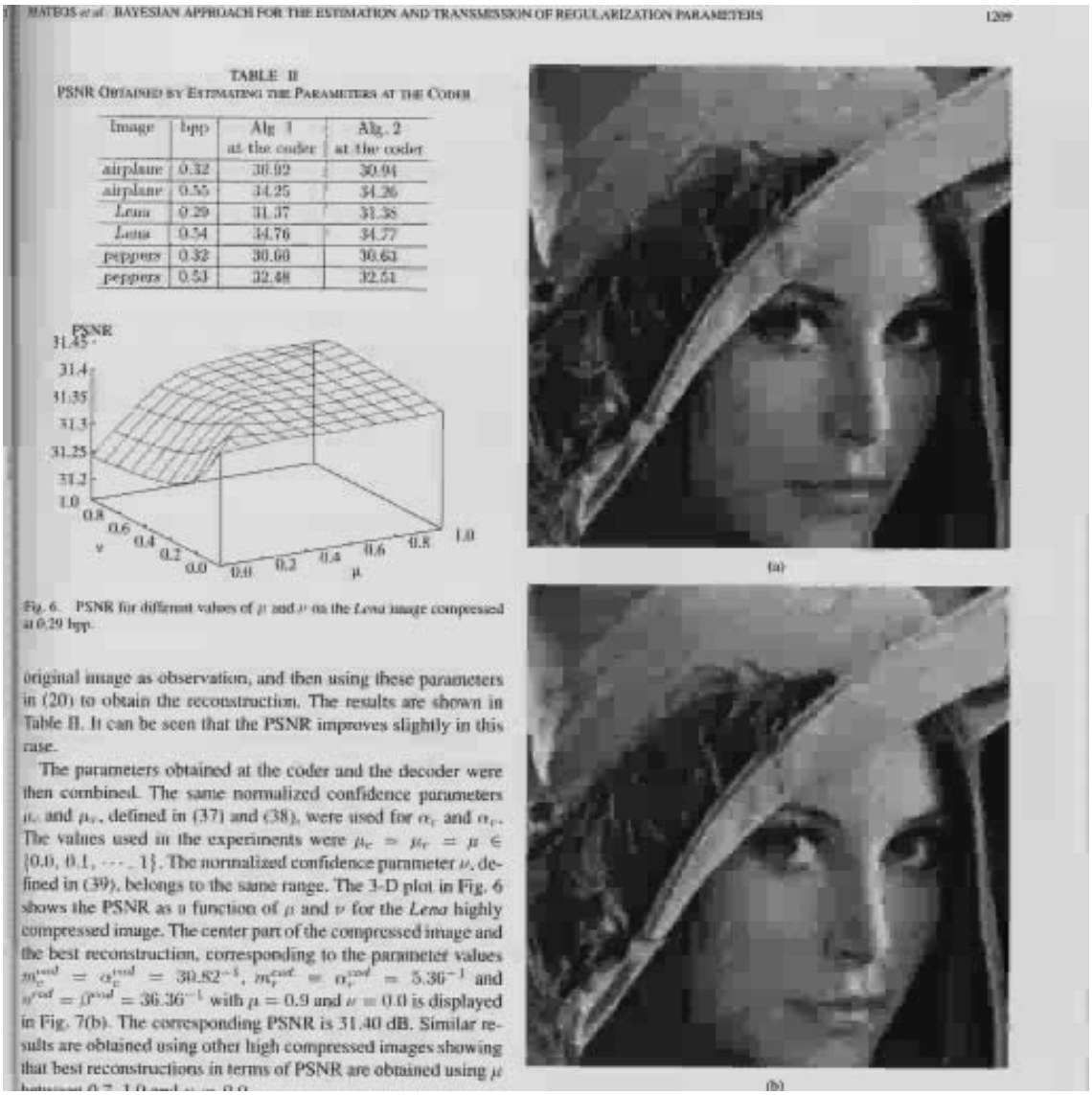


Figura F.13: PP1205 compressed by 2D-MMP-RD at 0.5 bpp. PSNR = 28.5 dB.



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

TABLE II
PSNR OBTAINED BY ESTIMATING THE PARAMETERS AT THE CODER

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

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

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

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

(a)

(b)

Figura F.14: PP1209 compressed by 2D-MMP-RD at 0.5 bpp. PSNR = 29.9 dB.

Apêndice G

Blocking effect on MMP

In this appendix we focus on the control of the blocking effect in MMP. The blocking-effect is a rather annoying artifact that block-coders, like MMP, tend to present. It is caused by artificial discontinuities at the edges of a block, usually a consequence of the independent processing of the blocks. This effect is more pronounced at low-rates. In fact, the discontinuities at the boundaries of the blocks tend to increase at low rates due to the greater variation admissible in the values reproduced.

Figure G.2 shows the image LENA 512×512 compressed by 2D-MMP-RD (with maximum block size of 16×16) to 0.2415 bpp and PSNR $= 31.65$ dB. The blocking effect is clear.

G.1 The source of blockiness in MMP.

MMP is based on a segmentation procedure that parses the input vector $\mathbf{X}$ in L segments $\mathbf{X} = \left(\mathbf{X}^{l_0} \quad \dots \quad \mathbf{X}^{l_{L-1}} \right)$. The procedure can guarantee that the mean distortion in the approximation of each segment $\hat{\mathbf{X}}^{l_m}$, and therefore in the concatenation $\hat{\mathbf{X}}$, is less than or equal to a target distortion d^*. Alternatively, the segmentation tree can be optimized to minimize $D + \lambda R$. However, in either case the algorithms have no control regarding the smoothness of the approximation at the boundaries of the segments, unless the target distortion d^* or the value of the λ parameter is zero. This gives rise to blocking effects. The concatenation of two segments can alternatively be viewed as the sum of two non-overlapping vectors

$\hat{\mathbf{X}}_s^{l_m} = \begin{pmatrix} \mathbf{X}^{l_m} & \mathbf{0}_{\ell(\mathbf{X}^{l_{m+1}})} \end{pmatrix}$ and $\hat{\mathbf{X}}_s^{l_{m+1}} = \begin{pmatrix} \mathbf{0}_{\ell(\mathbf{X}^{l_m})} & \mathbf{X}^{l_{m+1}} \end{pmatrix}$. One way to improve the smoothness at the boundary, and consequentially reduce the blocking effects, is to allow overlapping of the segments. However, if we use overlapping vectors, we can no longer guarantee that the sum of two segments, each one with mean distortion below the target d^* will not have mean distortion above d^*. We have made some experiments using overlapping vectors in the dictionaries of both the encoder and the decoder and found it difficult to optimize the choice of bases at the encoder side.

G.2 Blocking effect reduction by adaptive post-filtering.

To circumvent the difficult optimization problem posed by the use of overlapped vectors at the encoder side, we can use one basis, with non-overlapped vectors, to analyze the input data at the encoder and a different basis, with overlapped functions, to reconstruct the data at the decoder side. For example, lets consider that MMP parses the input vector $\mathbf{X}$ in four segments of same length and approximates each one by an equal components vector. That is, the segmentation tree is a binary tree of depth 2 and $\mathbf{X} = \begin{pmatrix} \mathbf{X}^3 & \mathbf{X}^4 & \mathbf{X}^5 & \mathbf{X}^6 \end{pmatrix}$. The approximation of each segment by a equal-components vector gives $\hat{\mathbf{X}} = \begin{pmatrix} a_0\mathbf{1}_{N/4} & a_1\mathbf{1}_{N/4} & a_2\mathbf{1}_{N/4} & a_3\mathbf{1}_{N/4} \end{pmatrix}$. The encoder has then the task of reconstructing the input vector from the four dictionary indexes representing these equal components vectors. This is analog to recovering an approximation to $\mathbf{X}$ from a downsampled by $N/4$ version of it, $\hat{\mathbf{X}}_d = \begin{pmatrix} a_0 & a_1 & a_2 & a_3 \end{pmatrix}$. It's well known from multirate filter banks theory [20] that we can recover a low-pass version of $\mathbf{X}$ from its downsampled version using a synthesis filter that can be different from the analysis filter. So we can tailor the impulse response of the synthesis filter, or, on the MMP decoder, the corresponding basis vector, to match the smoothness requirement.

We used an adaptive FIR post-filtering technique to modify the shape of the reconstruction vectors at the decoder, as follows:

- Firstly, we find an approximation for $\mathbf{X}$ using the standard decoder, $\hat{\mathbf{X}} = \begin{pmatrix} \mathbf{s}_{i_0}^{(k_0)} & \mathbf{s}_{i_1}^{(k_1)} & \dots & \mathbf{s}_{i_{L-1}}^{(k_{L-1})} \end{pmatrix}$. That is, $\hat{\mathbf{X}}$ is composed by the concatenation of L segments, each one a vector $\mathbf{s}_{i_m}^{(k_m)}$ in the dictionary of scale k_m.

- Next, we replace each $\mathbf{s}_{i_m}^{(k_m)}$ by a concatenation of vectors in the initial dictionaries. That is, $\hat{\mathbf{X}} = \left(\begin{array}{cccc} \mathbf{s}_{i'_0}^{(k'_0)} & \mathbf{s}_{i'_1}^{(k'_1)} & \dots & \mathbf{s}_{i'_{q-1}}^{(k'_{q-1})} \end{array} \right)$, where the vectors $\mathbf{s}_{i'_m}^{(k'_m)}$ are in the initial dictionaries $\mathcal{D}_0^{(k)}$. This creates a new segmentation of the reconstructed vector $\hat{\mathbf{X}}$, where each segment corresponds to an element of the initial dictionaries, $\hat{\mathbf{X}} = \left(\begin{array}{cccc} \hat{\mathbf{X}}^{l_0} & \hat{\mathbf{X}}^{l_1} & \dots & \hat{\mathbf{X}}^{l_{q-1}} \end{array} \right)$.

- Finally, we apply a zero-delay moving-average filter to $\hat{\mathbf{X}}$. The length of the filter at each segment $\hat{X}^{l_m}$ is proportional to the length of the segment $\ell(\hat{\mathbf{X}}^{l_m})$. Therefore, the size of the filter is adjusted in a sample-by-sample basis. Considering again, for example, that the reconstructed $\hat{\mathbf{X}}$ has four segments of equal-components vectors, and the filter is a moving average filter of length $L_k + 1$, this procedure replaces the four non-overlapped flat vectors of size $N/4$ by four overlapped triangle-shaped vectors of size $N/2$.

Figure G.1 illustrates the process. In this figure, the output vector is composed of three segments. The component being filtered is indicated by the arrow. As can be seen, the length of the FIR filter is proportional to the size of the original reconstruction vector.

G.3 Experimental results

Figure G.2 shows the results of the original MMP applied to the image LENA size 512×512. Figure G.3 shows the results of the modified MMP applied to the same image. Both images were coded at 0.2415 bits/pixel. The effectiveness of the method proposed in reducing the blockiness is clear.

The PSNR values for the original and modified MMP applied to LENA 512×512 at 0.2415 bits/pixel are 31.65 and 31.31 dB, respectively. In figure G.4 we show the objective performance of both decoders at other rates, and results for the SPIHT algorithm for reference. The penalty in objective performance reaches a maximum of 0.34 dB at 0.2415 bits/pixel. However, the subjective performance of the modified decoder is by far superior to that of the original decoder at all rates.

We have made no attempt to optimize the impulse response of the adaptive filter used at the decoder. However, in one interesting experiment, we changed the

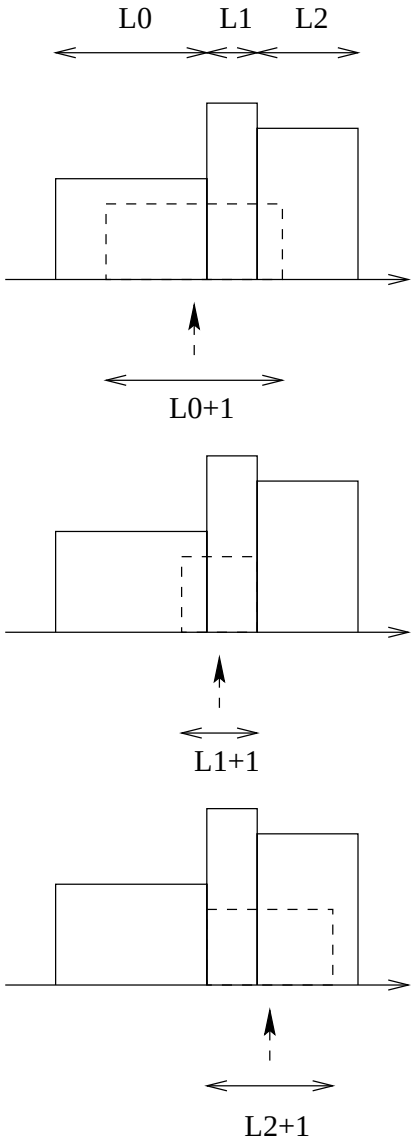


Figura G.1: FIR Post-filtering process.

Figura G.2: Original MMP. LENA at 0.2415 bpp

Figura G.3: Modified MMP with moving-average adaptive filter. LENA at 0.2415 bpp.

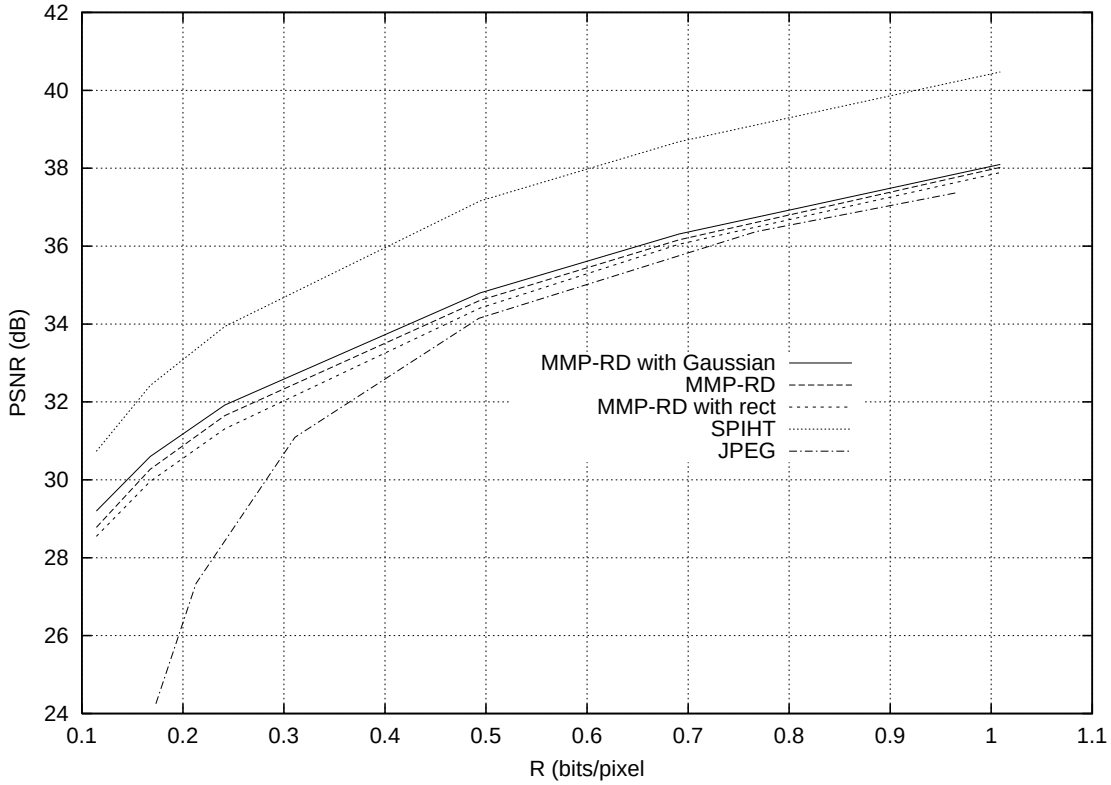


Figura G.4: Rate-distortion of MMP and modified MMP with LENA.

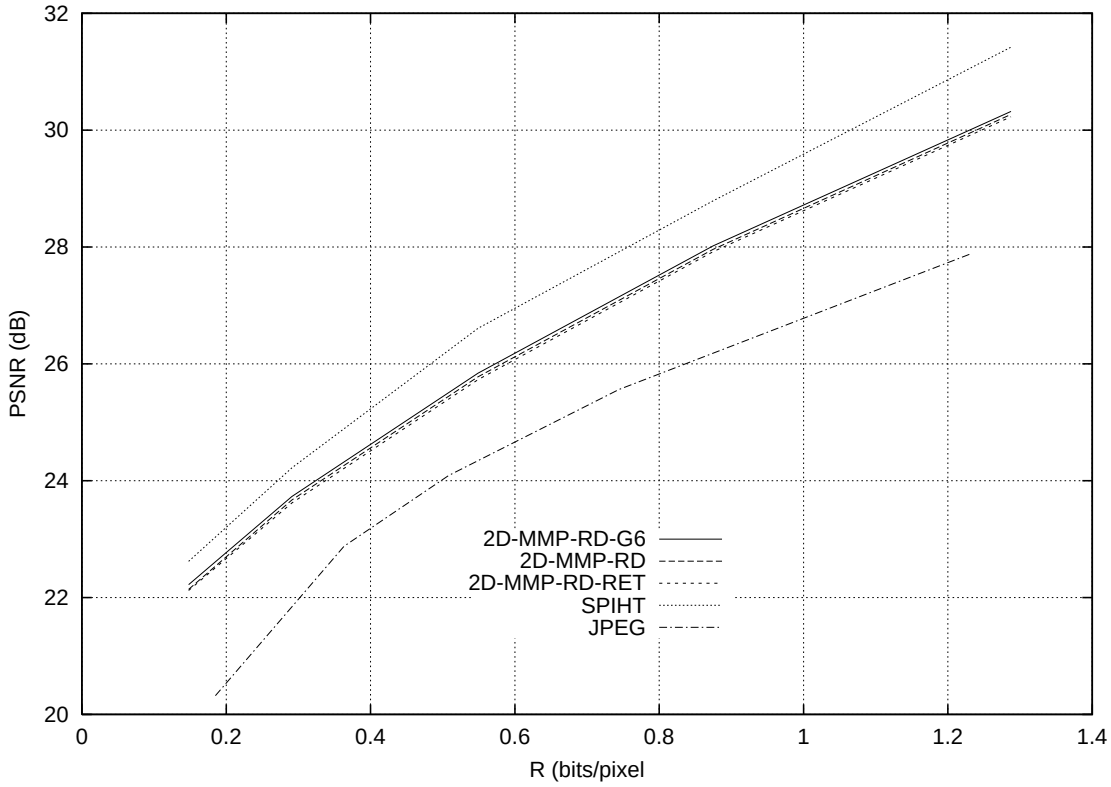


Figura G.5: Rate-distortion of MMP and modified MMP with BABOON.

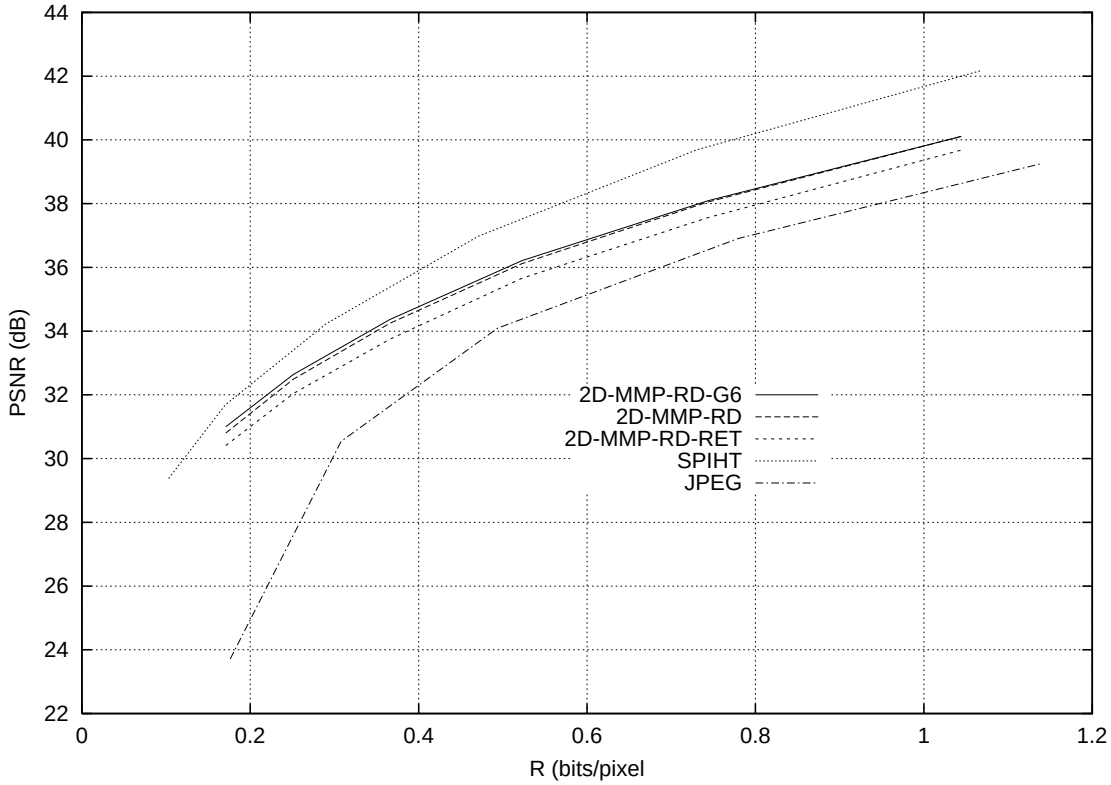


Figura G.6: Rate-distortion of MMP and modified MMP with F16.

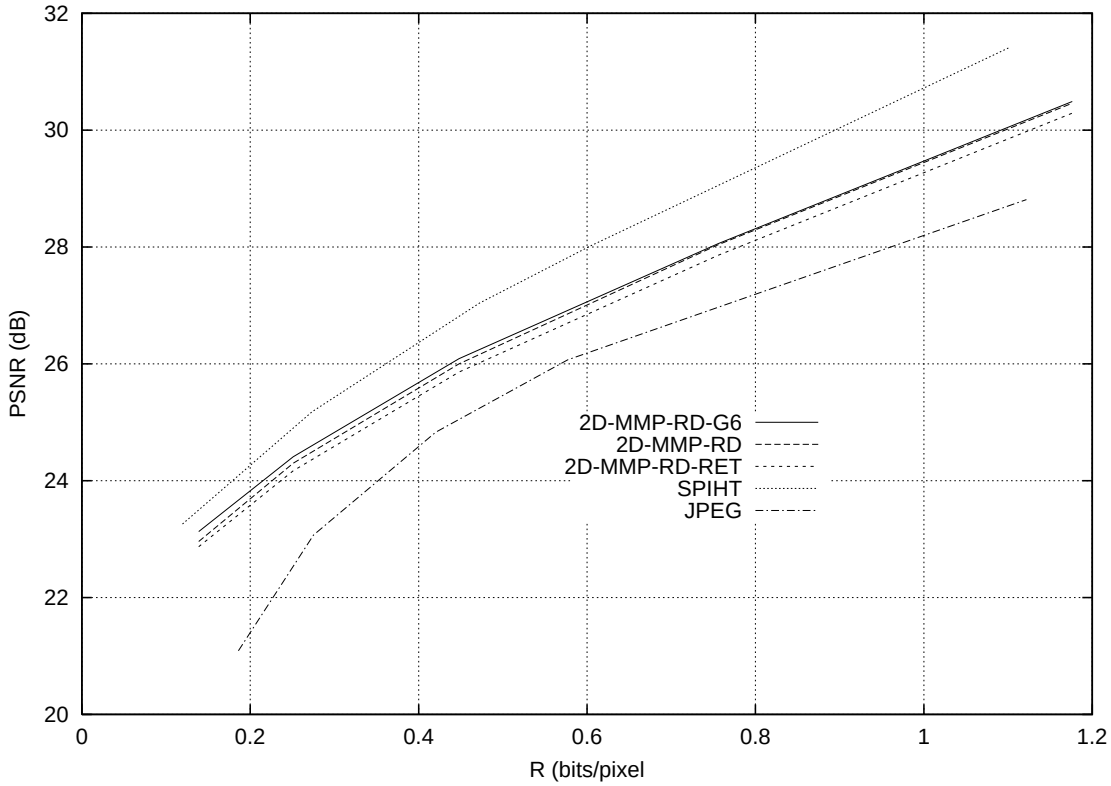


Figura G.7: Rate-distortion of MMP and modified MMP with BRIDGE.

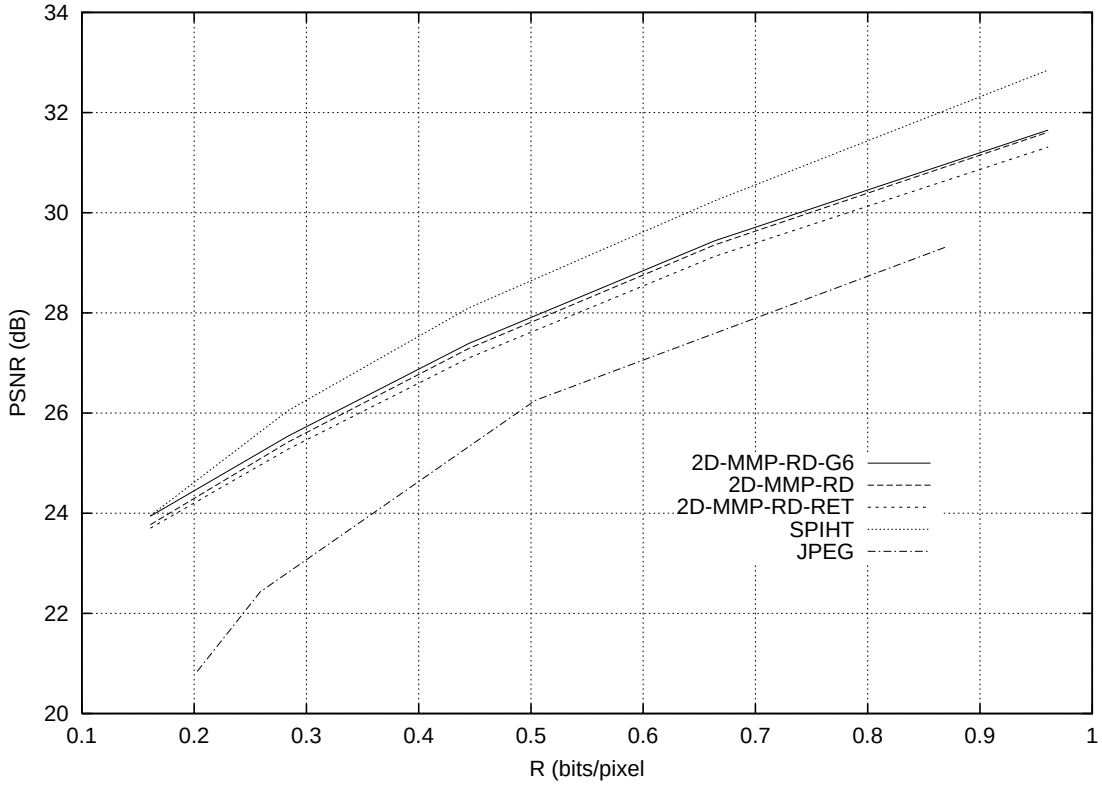


Figura G.8: Rate-distortion of MMP and modified MMP with AERIAL.

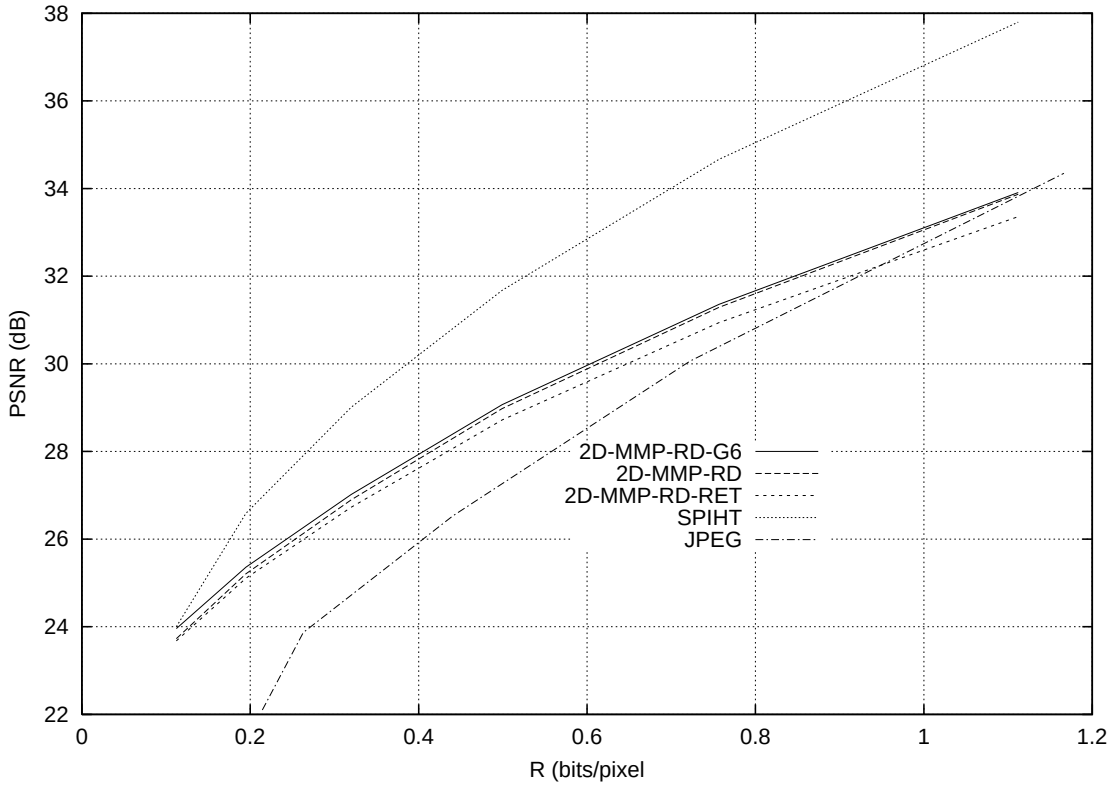


Figura G.9: Rate-distortion of MMP and modified MMP with BARBARA.

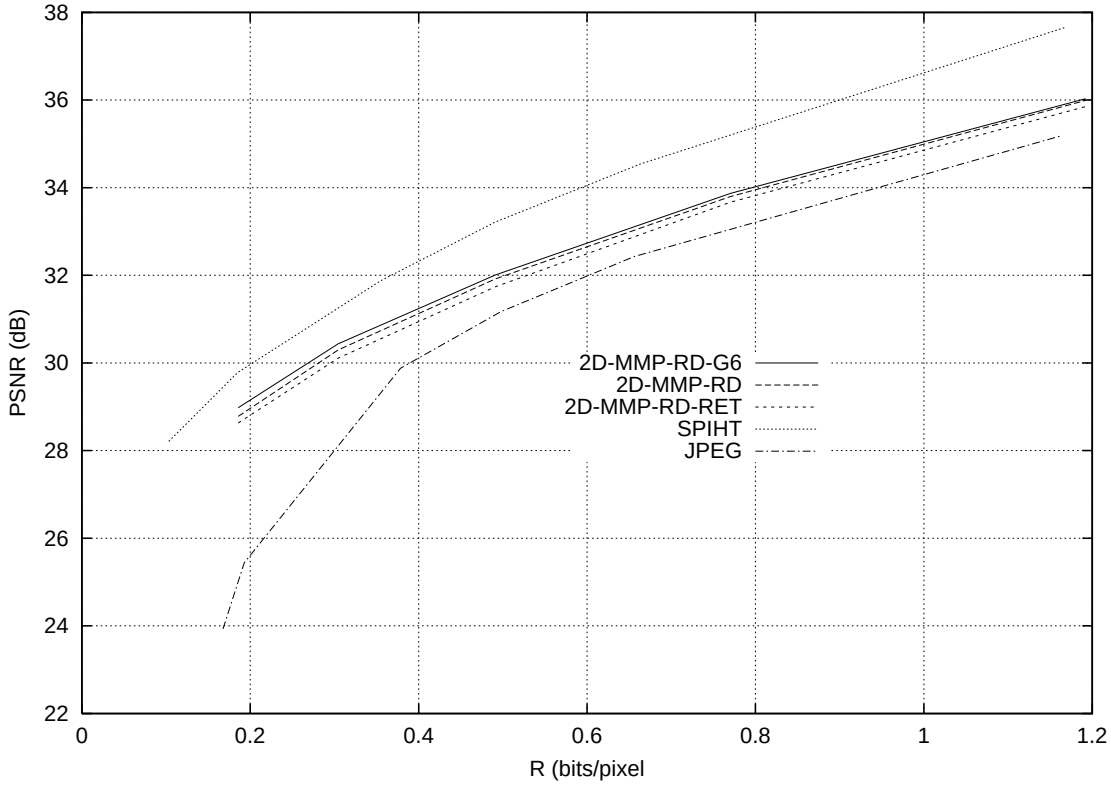


Figura G.10: Rate-distortion of MMP and modified MMP with GOLD.

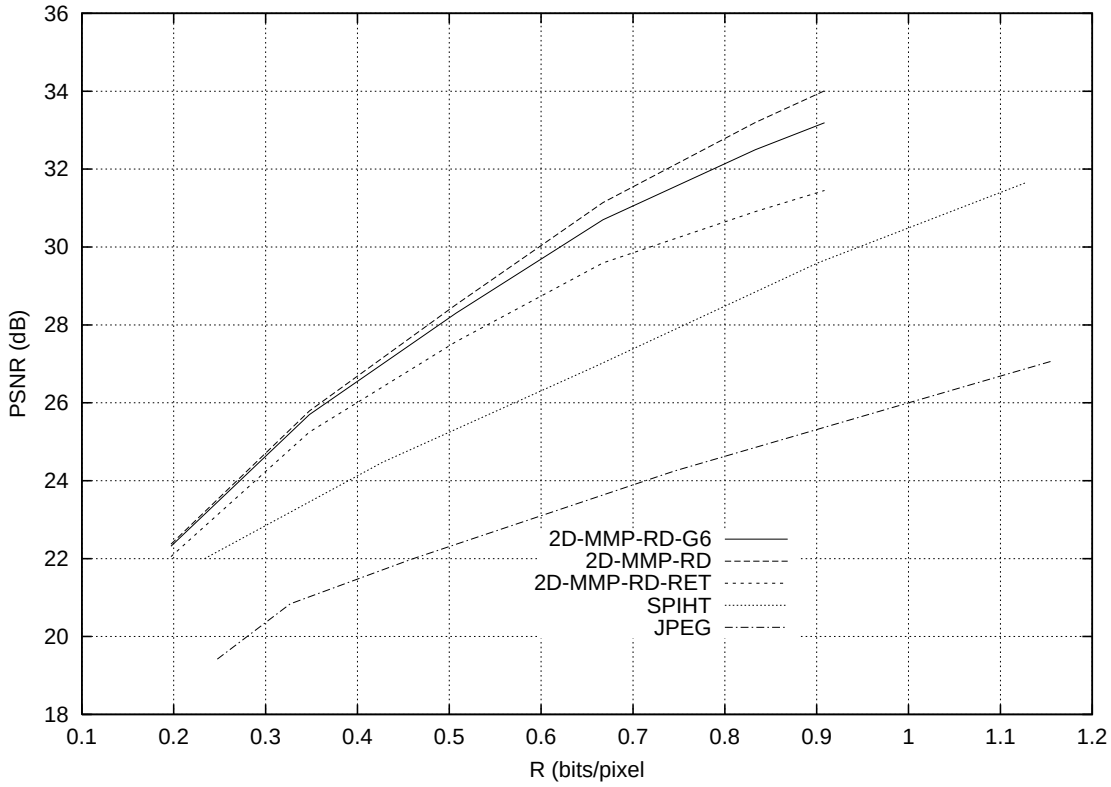


Figura G.11: Rate-distortion of MMP and modified MMP with PP1205.

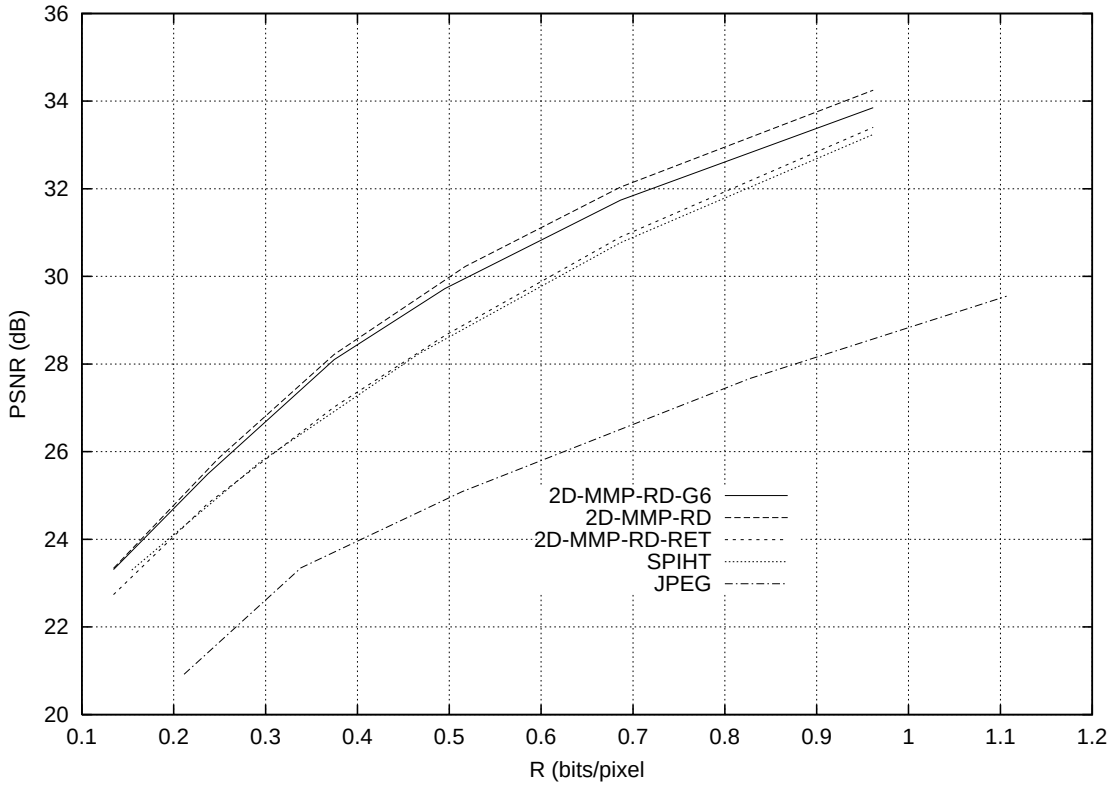


Figura G.12: Rate-distortion of MMP and modified MMP with PP1209.

shape of the impulse response of the adaptive filter from flat to Gaussian. The inpulse response of this filter, for a $N \times M$ block was given by:

$$h_{n,m} = \begin{cases} C_0 e^{-C_1\left(\left(\frac{n}{N}\right)^2+\left(\frac{m}{M}\right)^2\right)}, & 1-N \le n \le N-1, 1-M \le m \le M-1 \\ 0, & \texttt{outside} \end{cases}$$

$$C_0 = \sum_{n=1-N}^{N-1} \sum_{m=1-M}^{M-1} e^{-C_1\left(\left(\frac{n}{N}\right)^2+\left(\frac{m}{M}\right)^2\right)} \qquad \text{(G.1)}$$

Although not as efficient as the rectangular filter to reduce the blocking effect, one of the Gaussian filters, with $C_1 = 6$, actually improved the objective performance at all rates, as illustrated in figure G.4. This happened with all test images except for those containing text, the PP1205 and PP1209. The image LENA reconstructed with the MMP with adaptive Gaussian filter at 0.2415 bpp and PSNR 31.92 dB is shown in figure G.13.

Figura G.13: Modified MMP with adaptive Gaussian filter. LENA at 0.2415 bpp.

Apêndice H

MMP applied to image data in the DWT domain

In this appendix we apply the MMP algorithm to compress image data previously transformed by a DWT (Discrete Wavelet Transform). This transformation was introduced in appendix B, and is used in many state-of-the-art wavelet coders. For images with high low-frequency content, the wavelet transform compacts most of the energy into a few low-frequency coefficients. If the image have different objects delimited by edges, the corresponding energy is concentrated in a few spatially clustered high-frequency coefficients. Complex-textured regions however, tend to be represented by many coefficients spread throughout the entire image in the transformed domain. Many wavelet-based lossy compression algorithms, like the EZW [12], the SPIHT [13] the MGE [14], the SWEET [15] and the JPEG2000 [11], are based on simple scalar quantization strategies associated with efficient methods to localize the coefficients of highest energy and encode them. The efficiency of these methods depends on the type of input image. It is usually assumed that the input images have strong low-frequency content, and that the high-frequency content is mostly concentrated around the edges of the objects in the scene, not in textures. When processing this kind of image, the MMP algorithm can explore this energy clustering property of the DWT to encode more efficiently. Also, the most annoying artifact produced by MMP when directly applied to image data, that is the blocking effect, is eliminated when using the DWT.

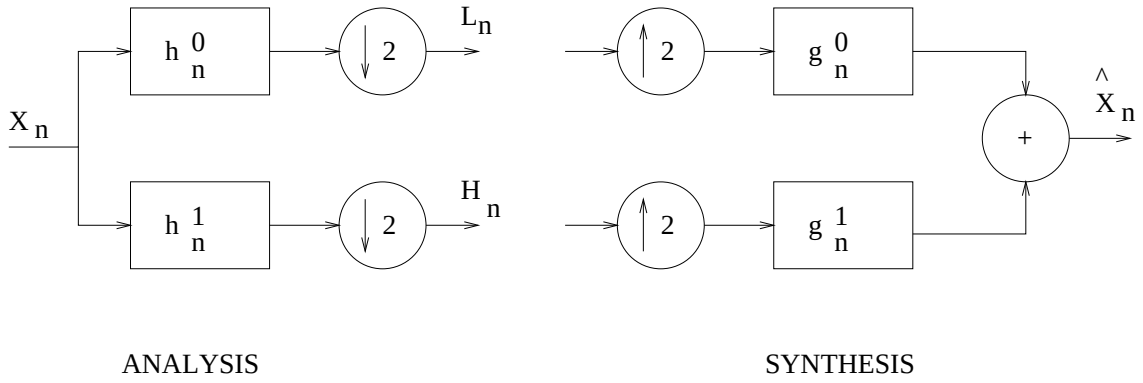


Figura H.1: Two-band filter bank.

H.1 The DWT

The DWT can be implemented with iterated filter banks. The fundamental building block of this construction is the two-band perfect reconstruction filter bank, illustrated in figure H.1.

The two channel bank is composed by a pair of analysis filters, with impulse responses h_n^0 and h_n^1, and a pair of synthesis filters, with impulse responses g_n^0 and g_n^1. The responses are such that the channel 0, that is the output of h^0, is related to the lower half of the spectrum of the input X_n, while the channel 1 contains high frequency information. The output of both analysis filters are critically downsampled by two, generating the two outputs of the analysis section of the filter bank, L_n (low-pass) and H_n (high-pass). Since the filters are not ideal, the output of the decimators have an aliasing component. However, the four filters in the system can be designed to provide alias cancellation, a necessary condition for perfect reconstruction. The outputs of the two channels of the analysis section feed two upsamplers, that are followed by the synthesis filters g_n^0 and g_n^1.

The two channel filter bank can be used in an iterated fashion to build a larger filter bank that implement the DWT. The one-dimensional version is illustrated in figure H.2.

In the filter bank of figure H.2 the input signal X_n is applied to a two-band filter bank. The output of the low-pass channel is subdivided in two additional bands by another two-band filter bank and so on. The number of iterations is also called the number of stages of the filter bank. The example illustrated in figure H.2 is the analysis section of a three-stage filter bank. Since the two-band filter

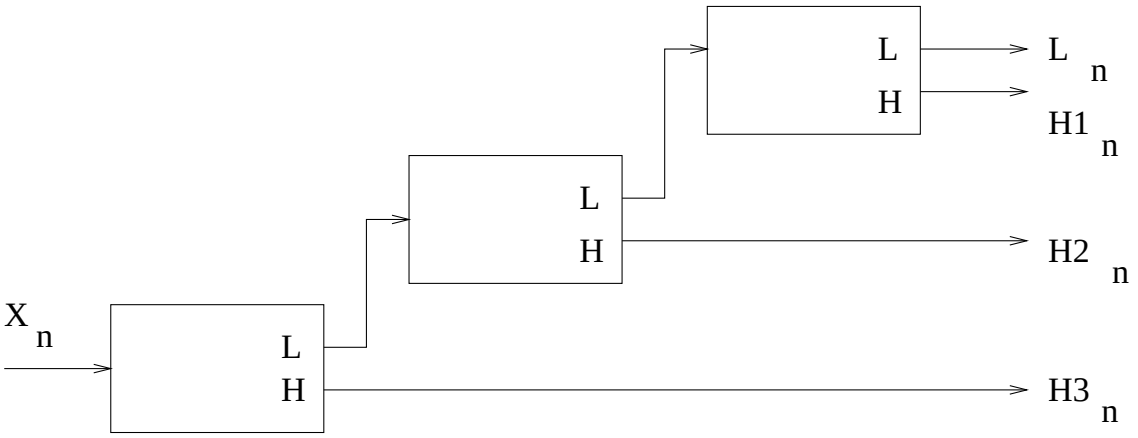


Figura H.2: Four-band iterated filter bank.

bank divides the spectrum of the input signal in two halves, the iterated filter bank conveys a dyadic decomposition of the spectrum, that is, the system performs an octave-band decomposition.

In the two-dimensional case, a two-dimensional input signal $X_{n,m}$ is input to a two-band filter bank that operates in the first variable n. Then the outputs of both channels are subdivided in four bands by a pair of two-band filter banks, but operating in the second variable m. At this point we have four channels: The first is low-pass in both variables n and m, the second is low-pass in n and high-pass in m, the third is high-pass in n and low-pass in m and the fourth is high-pass in both n and m. We denote the outputs of these channels respectively by $LL_{n,m}$, $LH_{n,m}$, $HL_{n,m}$ and $HH_{n,m}$. The next iteration proceeds in the output of the LL channel and so on. Figure H.3 illustrates the analysis section of a two-dimensional two-stage iterated filter bank, and figure H.4 shows the frequency decomposition obtained with this filter bank.

H.2 Experimental results

The DWT was implemented using the 9-7 biorthogonal set of linear phase FIR filters of [30]. The coefficients of these filters are listed in table H.1.

The two-dimensional input signal $\mathbf{X} = \{X_{n,m}\}$ has finite length $N \times M$, therefore it must be properly extended at the borders to ensure perfect reconstruction by the iterated filter bank. For these filters, the proper extension to $X_{n,m}$ is: when filter-

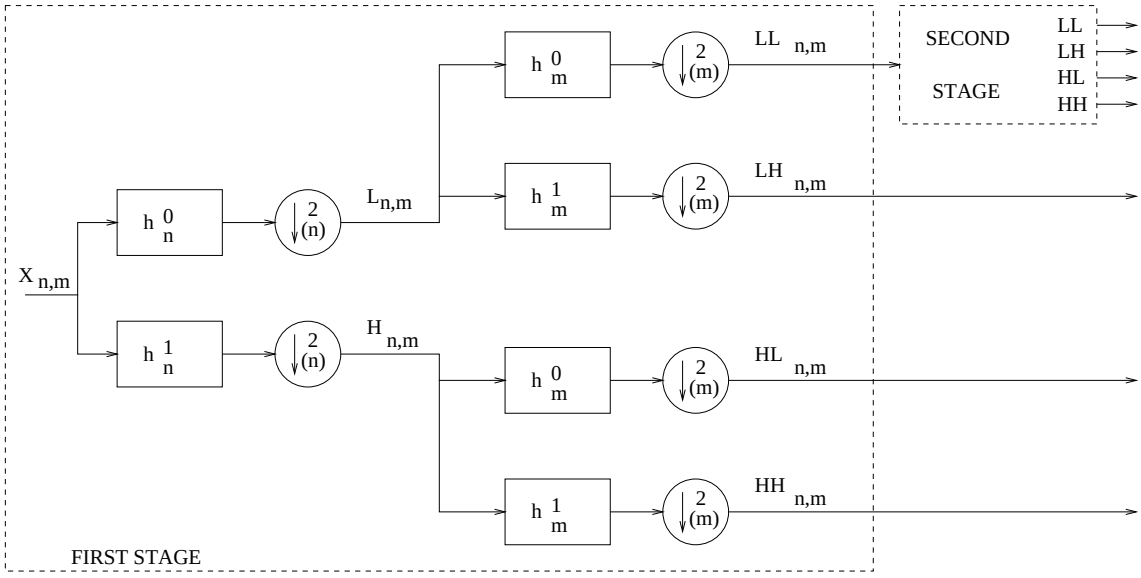


Figura H.3: Two-dimensional iterated filter bank.

n	h_n^0	h_n^1	g_n^0	g_n^1
0	0.02674875736	-0.09127176308	-0.09127176308	-0.02674875736
1	-0.0168641184	0.0575435262	-0.0575435262	-0.0168641184
2	-0.0782232663	0.5912717632	0.5912717632	0.0782232663
3	0.2668641183	-1.115087052	1.115087052	0.2668641183
4	0.6029490175	0.5912717632	0.5912717632	-0.6029490175
5	0.2668641183	0.0575435262	-0.0575435262	0.2668641183
6	-0.0782232663	-0.09127176308	-0.09127176308	0.0782232663
7	-0.0168641184			-0.0168641184
8	0.02674875736			-0.02674875736

Tabela H.1: Coefficients of the filters D9-7.

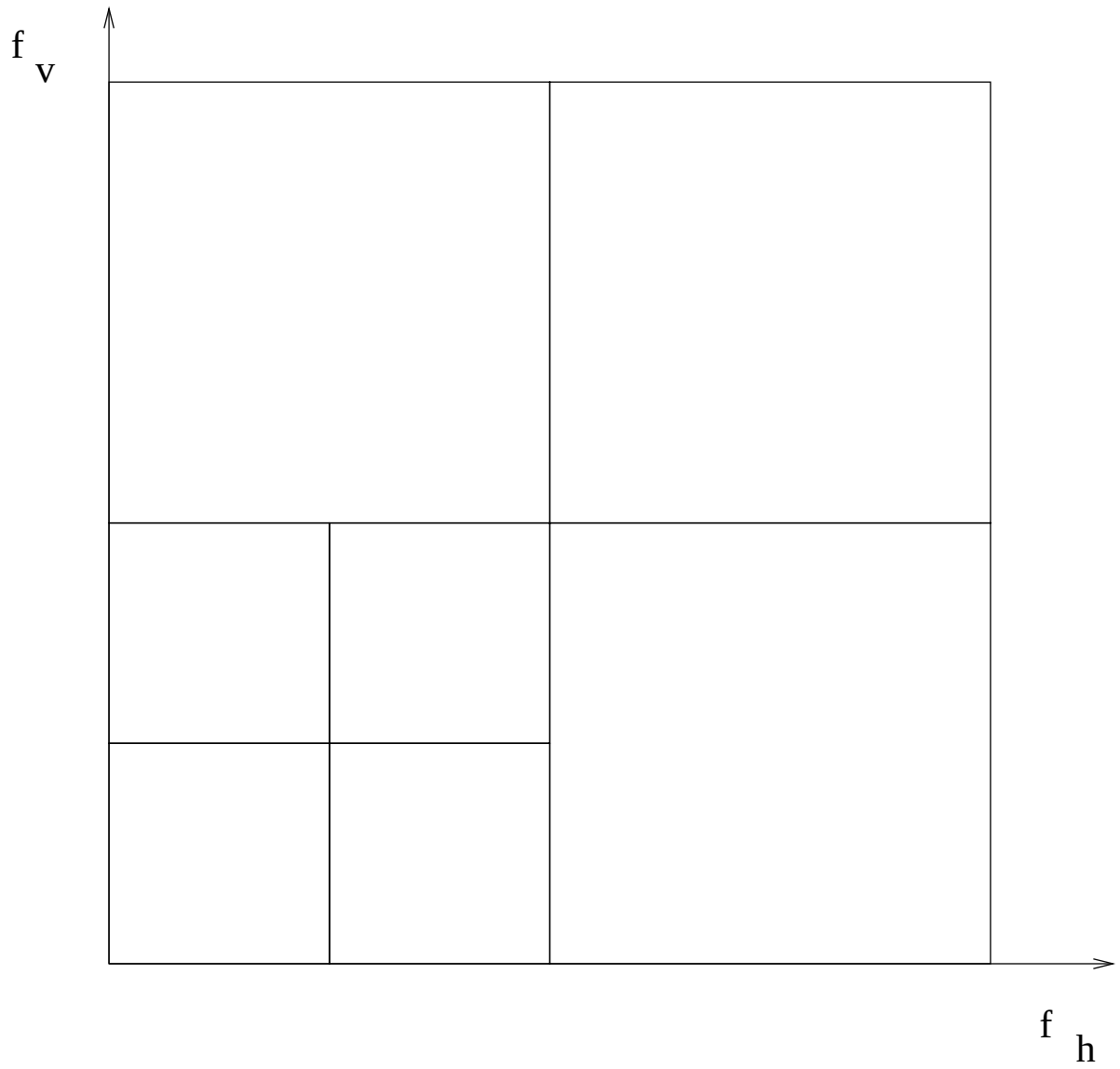


Figura H.4: Spectrum subdivision of the DWT.

ing in the n variable, the rows of $\mathbf{X}$ must be extended accordingly to $X^e_{n,m} = X_{-n,m}$ for $n < 0, 0 \leq m < M$ and $X^e_{n,m} = X_{2N-n-2,m}$ for $n \geq N, 0 \leq m < M$. When filtering in the m variable, the columns of $\mathbf{X}$ must be extended accordingly to $X^e_{n,m} = X_{n,-m}$ for $0 \leq n < N, m < 0$ and $X^e_{n,m} = X_{n,2M-m-2}$ for $0 \leq n < N, m > M$.

The output of the channels have also finite length, $N/2 \times M/2$, and must be properly extended prior to the synthesis filtering. The extension used is: When filtering in the n variable, the rows of $\mathbf{LL}$ (or $\mathbf{LH}$, or $\mathbf{HL}$, or $\mathbf{HH}$) must be extended accordingly to $LL^e_{n,m} = LL_{-n,m}$ for $n < 0, 0 \leq m < M$ and $LL^e_{n,m} = LL_{2N-n-2,m}$ for $n \geq N, 0 \leq m < M$. When filtering in the m variable, the columns of $\mathbf{LL}$ must be extended accordingly to $LL^e_{n,m} = LL_{n,-m}$ for $0 \leq n < N, m < 0$ and $LL^e_{n,m} = LL_{n,2M-m-2}$ for $0 \leq n < N, m > M$.

The dynamic range of the subband channels are usually much higher than that of the input signal, and increases with the number of stages. In a 6-stage decomposition the dynamic range of a typical 256 gray-levels image can increase to 8192 levels. In the previous appendixs we used an initial dictionary of almost the size of the alphabet. If we try to do this with the image transformed by the DWT the initial dictionary would be too large. Therefore we start with dictionaries with just one element, the all-zeroes matrix. We made a small modification in MMP to allow the algorithm to learn the extra symbols it needs as they occur. This modified 2D-MMP algorithm is described next.

Let:

- $\mathbf{X}$ be an $(N \times M) = (2^K \times 2^L)$ matrix.
- $\mathcal{D}^{(k,l)} = \{\mathbf{s}_0^{(k,l)}, \ldots, \mathbf{s}_{I-1}^{(k,l)}\}$,
  $(k,l) = (0,0), (1,0), (1,1) \ldots, (K,L)$ a set of $K + L + 1$ dictionaries with $I^{(k,l)}$ matrices of size $2^k \times 2^l$ each. These dictionaries must be initialized to $\mathcal{D}^{(k,l)} = \{\mathbf{0}^{(k,l)}\}$.
- d^* be a target distortion.
- $T_{N,M}[\mathbf{X}]$ be a scale transformation that maps the matrix $\mathbf{X}$ in a matrix of size $N \times M$.

Procedure $\hat{\mathbf{X}}^j = \text{encode}(\mathbf{X}^j, d^*)$:

step 1 find index i_j in the dictionary of the same scale (k,l) of $\mathbf{X}^j$
such that $\|\mathbf{X}^j - \mathbf{s}_{i_j}^{(k,l)}\|$ is minimum and make $\hat{\mathbf{X}}^j = \mathbf{s}_{i_j}^{(k,l)}$.

step 2 `if` $k = l = 0$, output $X = \mathbf{X}^j$ and return $X = \mathbf{X}^j$. make $\hat{\mathbf{X}}^j = \mathbf{X}^j$
`else` go to step 3.

step 3 `if` $\|\mathbf{X}^j - \hat{\mathbf{X}}^j\|^2 \leq 2^{k+l} d^*$ `then` output flag '1', output index i_j and return $\hat{\mathbf{X}}^j$.
`else` go to step 4.

step 4 output flag '0'.

step 5 `if` $k > l$ split $\mathbf{X}^j = \begin{pmatrix} \mathbf{X}^{2j+1} \\ \mathbf{X}^{2j+2} \end{pmatrix}$,
where $\mathbf{X}^{2j+1}$ and $\mathbf{X}^{2j+2}$ are $2^{k-1} \times 2^l$
`else` split $\mathbf{X}^j = \begin{pmatrix} \mathbf{X}^{2j+1} & \mathbf{X}^{2j+1} \end{pmatrix}$
where $\mathbf{X}^{2j+1}$ and $\mathbf{X}^{2j+2}$ are $2^k \times 2^{l-1}$

step 6 compute $\hat{\mathbf{X}}^{2j+1} = \text{encode}(\mathbf{X}^{2j+1}, d^*)$

step 7 compute $\hat{\mathbf{X}}^{2j+2} = \text{encode}(\mathbf{X}^{2j+2}, d^*)$

step 8 `if` $k > l$ make $\hat{\mathbf{X}}^j = \begin{pmatrix} \hat{\mathbf{X}}^{2j+1} \\ \hat{\mathbf{X}}^{2j+2} \end{pmatrix}$,
`else` make $\hat{\mathbf{X}}^j = \begin{pmatrix} \hat{\mathbf{X}}^{2j+1} & \hat{\mathbf{X}}^{2j+2} \end{pmatrix}$

step 9 `for` $n = 0, 1, \ldots, K + L + 1$,
$p = \left\lfloor (\frac{n+1}{2}) \right\rfloor$, $q = \left\lfloor (\frac{n}{2}) \right\rfloor$,
$\mathcal{D}^{(p,q)} = \mathcal{D}^{(p,q)} \cup \left\{ T_{2^p,2^q}[\hat{\mathbf{X}}^j] \right\}$
where $\lfloor x \rfloor$ is the largest integer that is smaller than or equal to x .

step 10 return $\hat{\mathbf{X}}^j$

We used an R-D optimized version of the algorithm to compress the transformed images. The algorithm is basically the same but the splitting decisions are made based on a previously R-D optimized segmentation code $\mathbf{C}^{(\mathcal{S})}$, as described in appendix F. This 2D-MMP-RD was applied to gray scale images transformed by a 6-stage DWT. Each band of the DWT was initially divided in $N \times M$ blocks that were processed in sequence by the algorithm. In our implementation, the size of the

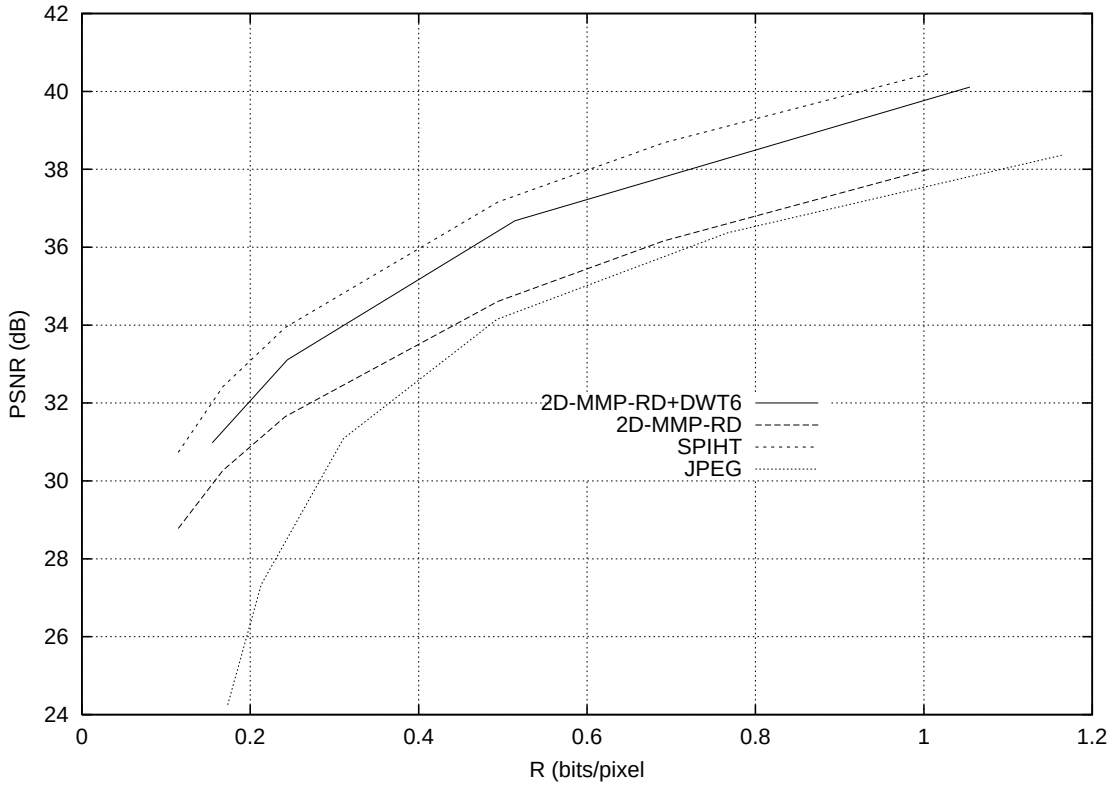


Figura H.5: R-D performance with LENA 512×512.

blocks cannot be larger than the size of the smaller low pass band. Since the test images are 512×512, in a 6-stage decomposition the low-frequency band is 8×8. Therefore we used $(N \times M) = (8 \times 8)$ blocks. The maximum value $\max(X_{n,m})$ and the minimum value $\min(X_{n,m})$ of the pixels of each band were computed and transmitted to the decoder. This values were used to specify the number of levels used by MMP to output a sample value $X_{n,m}$ (step 2 of the algorithm). The lower-energy higher-frequency bands were processed first.

Figures H.5 to H.11 show the rate-distortion performance for the test images LENA, BABOON, F-16, BRIDGE, AERIAL, BARBARA and GOLD. The performances of SPIHT and the MMP-RD directly applied to these images are also shown for comparison purposes.

As can be seen from figures H.5 to H.11, the R-D performance improved with the DWT for all pure gray-scale images. The larger improvement was for the image BARBARA, for wich the PSNR increased 2.2 dB at 0.5 bits/pixel. The smallest improvement was for the image AERIAL, with less than 0.1 dB of increase in PSNR at the same rate. For these images, the coeficients in the transformed domain

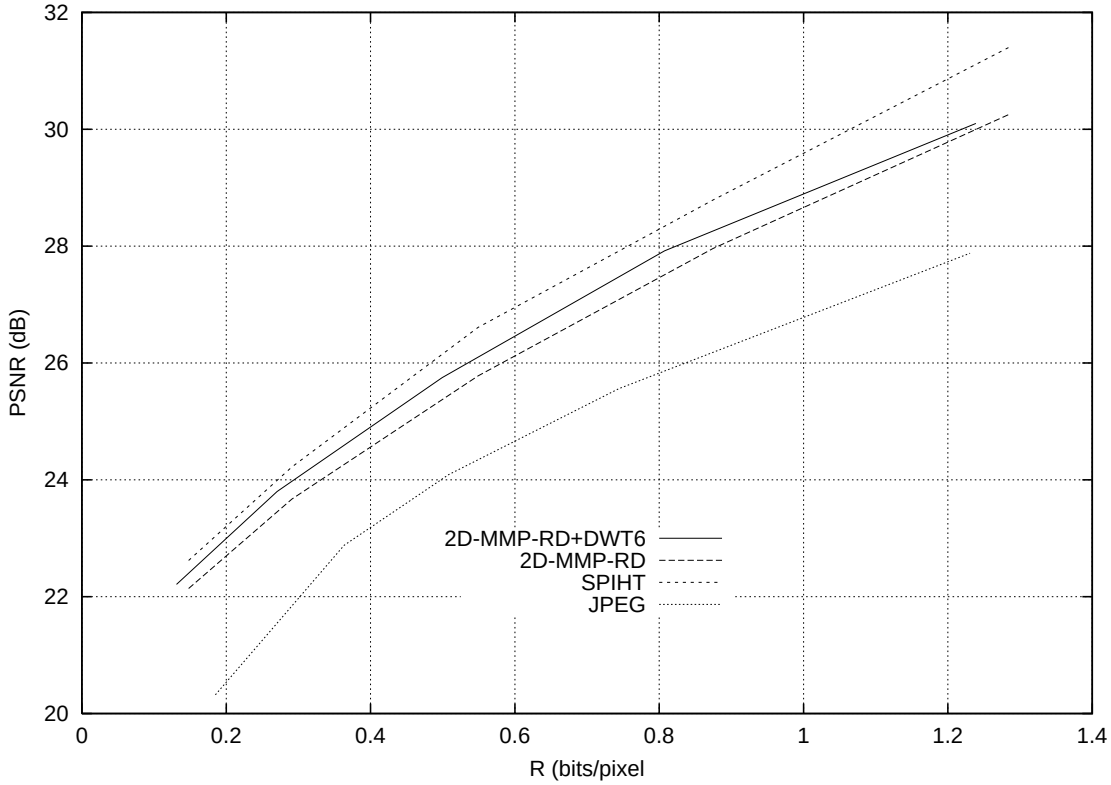


Figura H.6: R-D performance with BABOON 512×512.

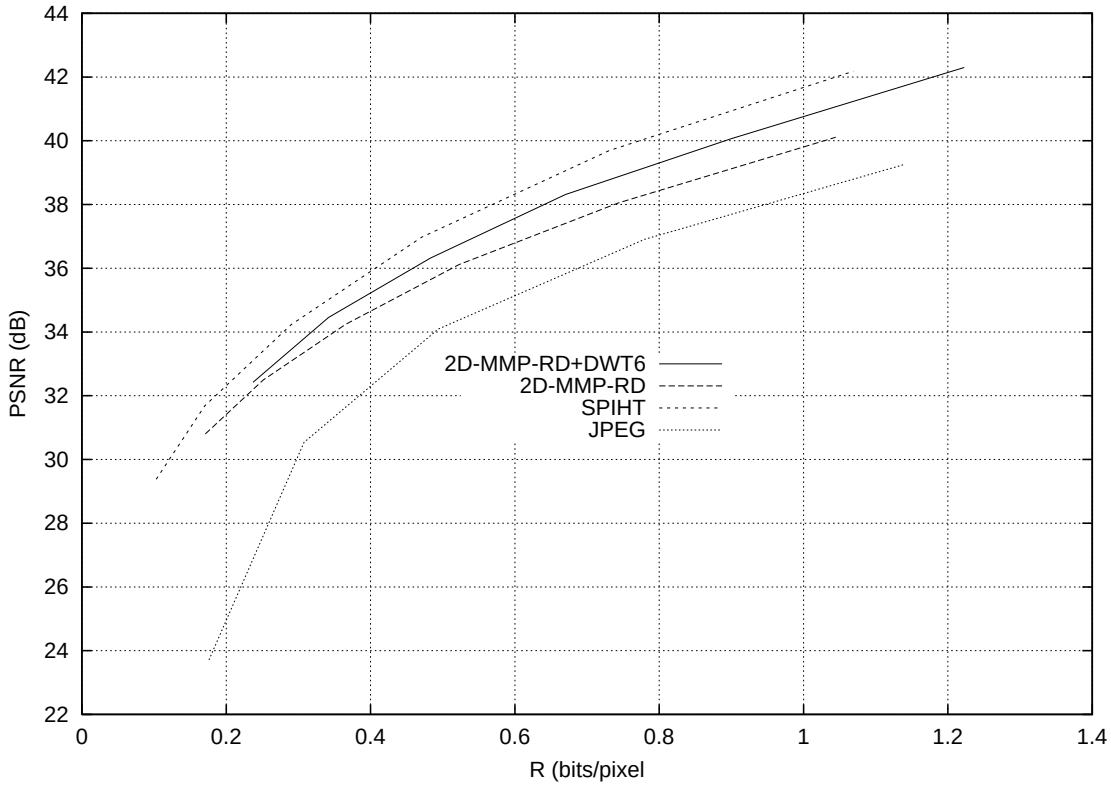


Figura H.7: R-D performance with F16 512×512.

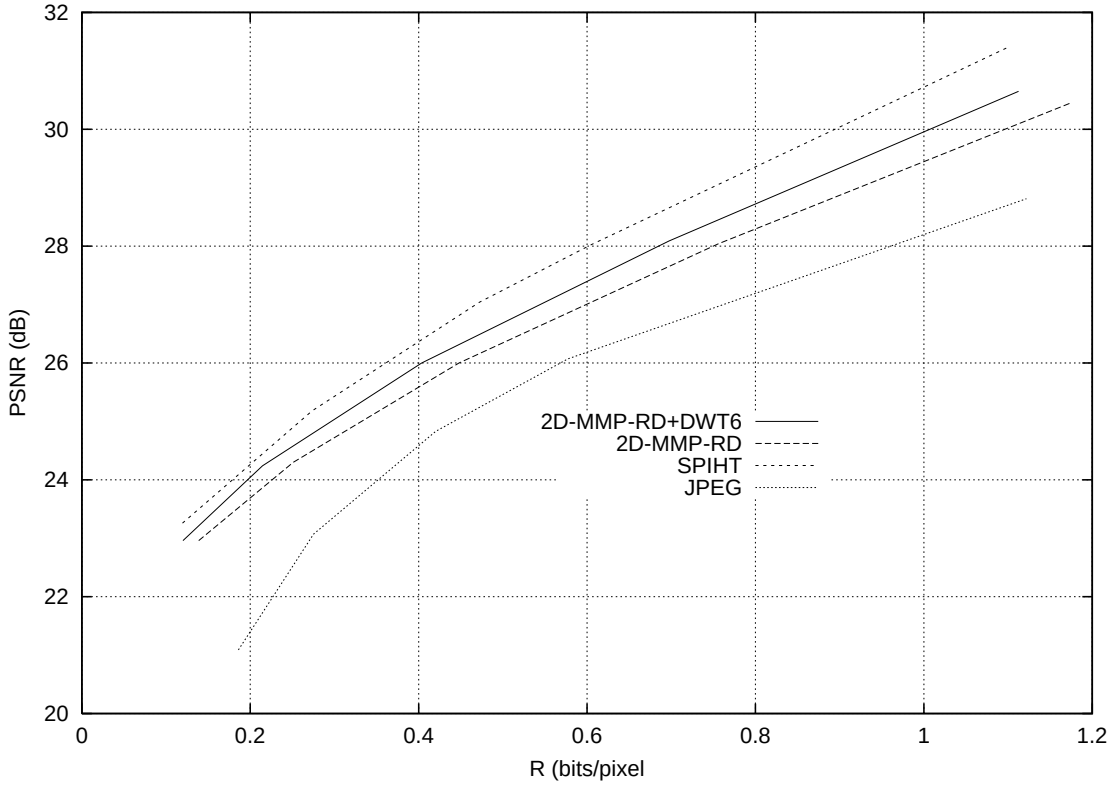


Figura H.8: R-D performance with BRIDGE 512×512.

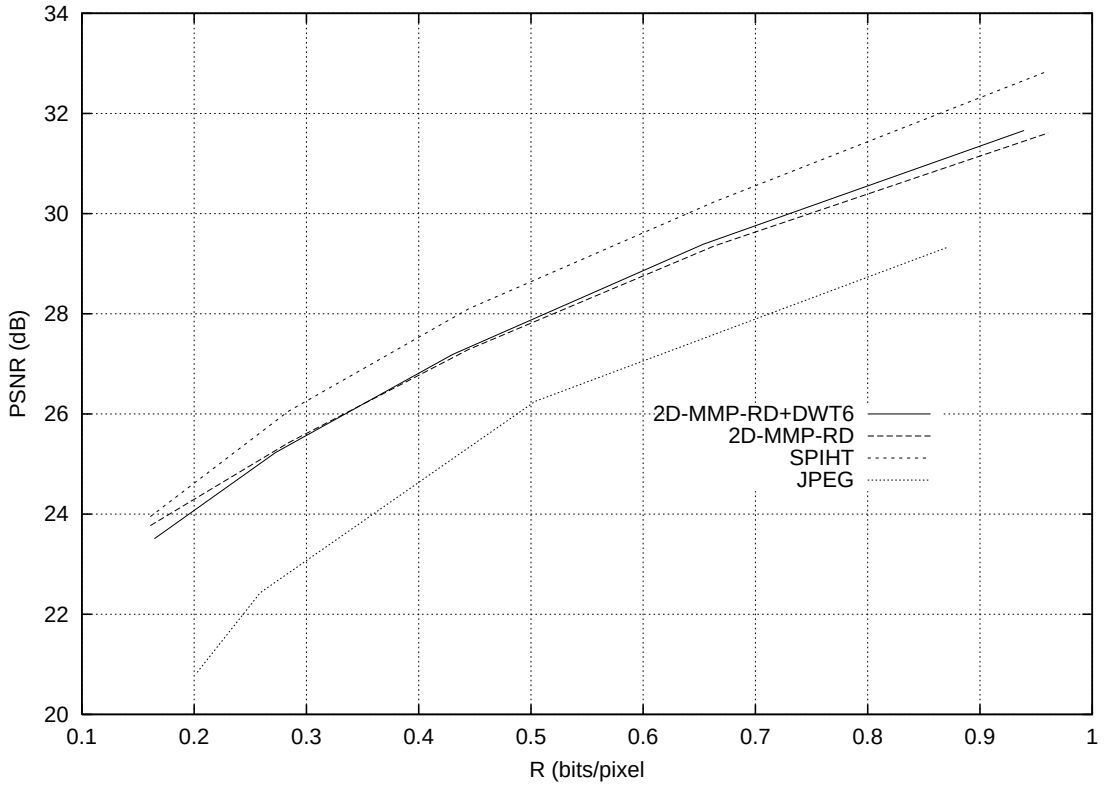


Figura H.9: R-D performance with AERIAL 512×512.

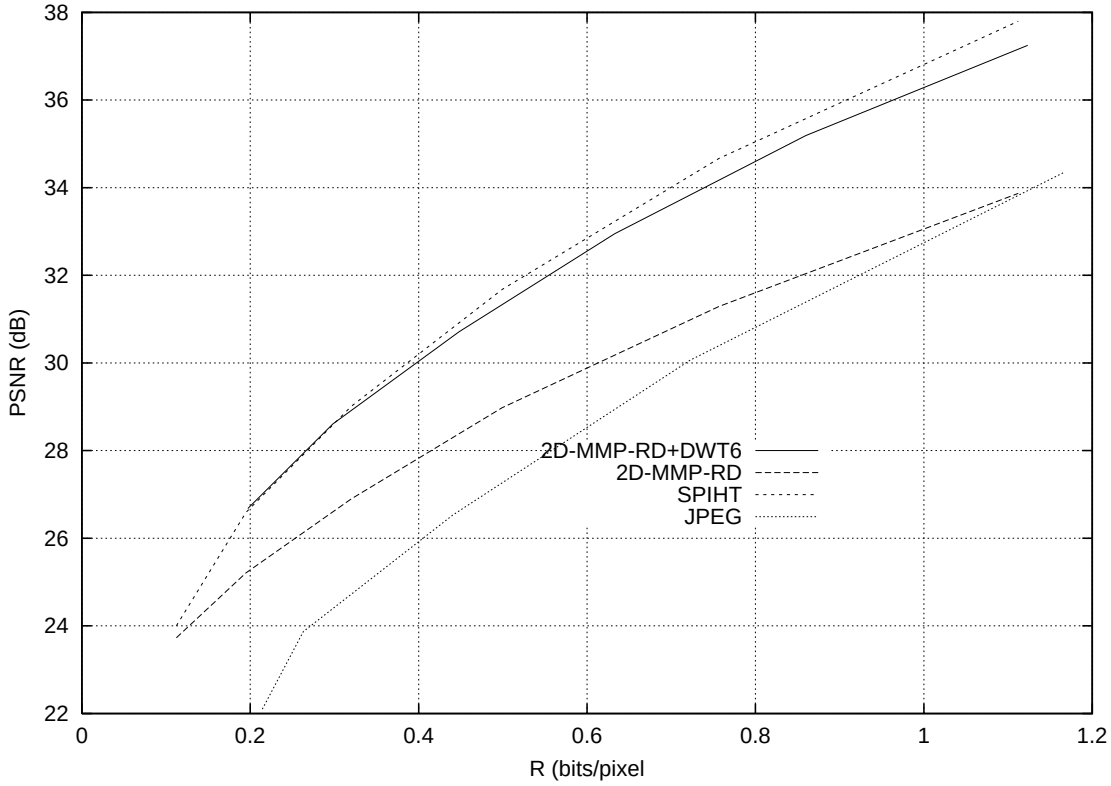


Figura H.10: R-D performance with BARBARA 512×512.

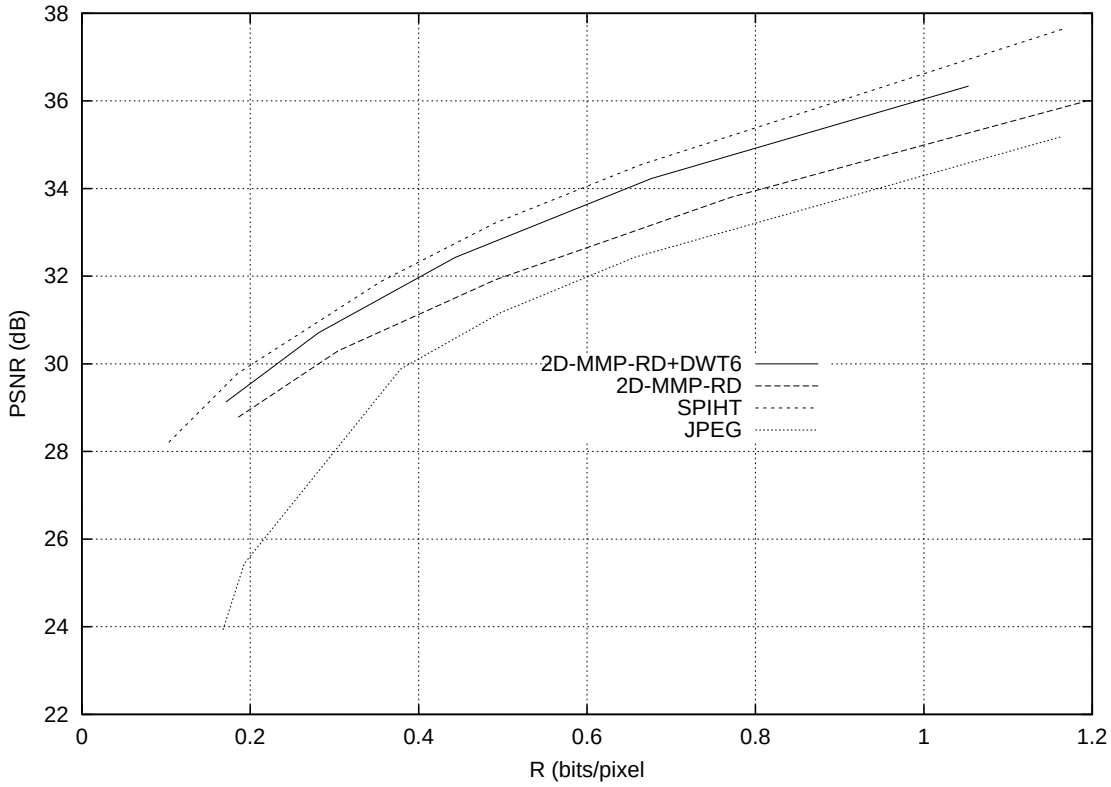


Figura H.11: R-D performance with GOLD 512×512.

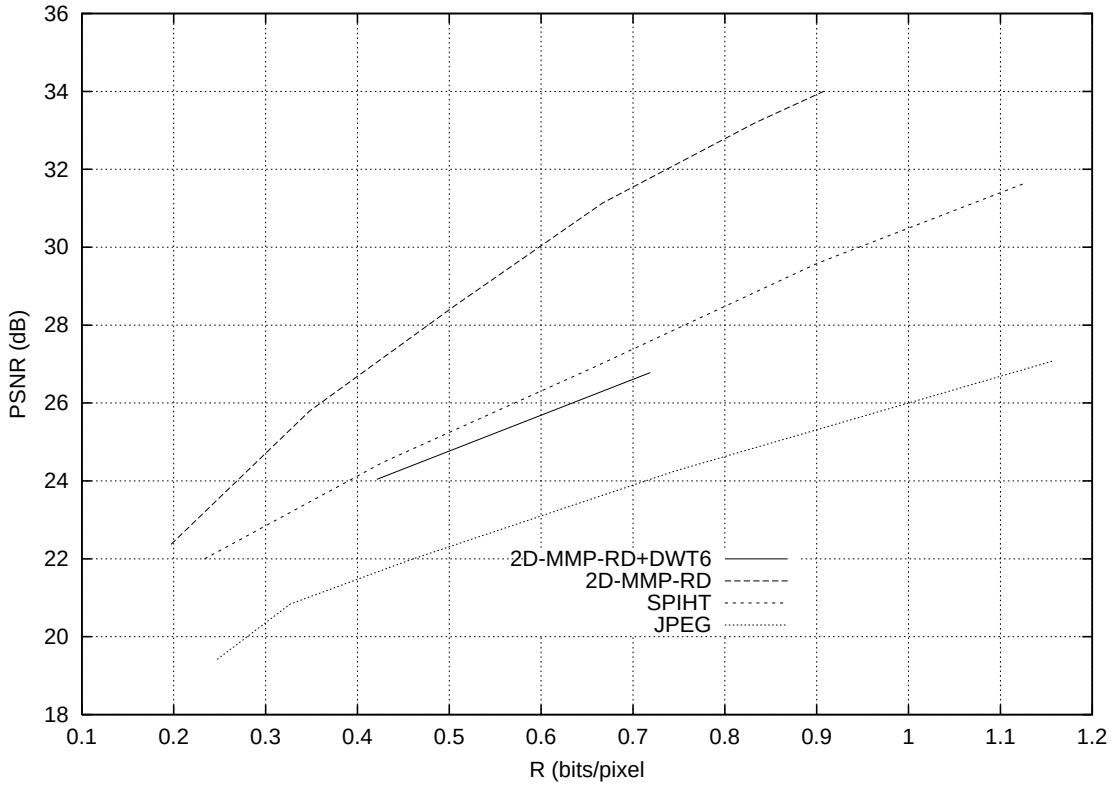


Figura H.12: R-D performance with PP1205 512×512.

are concentrated in a predictable way. We explored this particular structure when we chose to process sequentialy each band of the DWT, starting by the highest-frequency band. Since the higher-frequency bands have lower energy, the dictionaries grow slowly because we have many matches with large segments. The combination of small dictionaries with large matches improves the coding efficiency.

However, as can be seen from figures H.12 and H.13, the R-D performance for PP1205 and PP1209 got worse when we used the DWT. Since these images have a great amount of sharp edges almost uniformly spatially distributed, the energy of the coeficients is not clustered. Therefore the MMP cannot take advantage of the spatial distribuition of the energy. Moreover, the dynamic range still increases at the DWT domain, so the MMP has to deal with a source that appears to be more difficult to compress in the DWT domain.

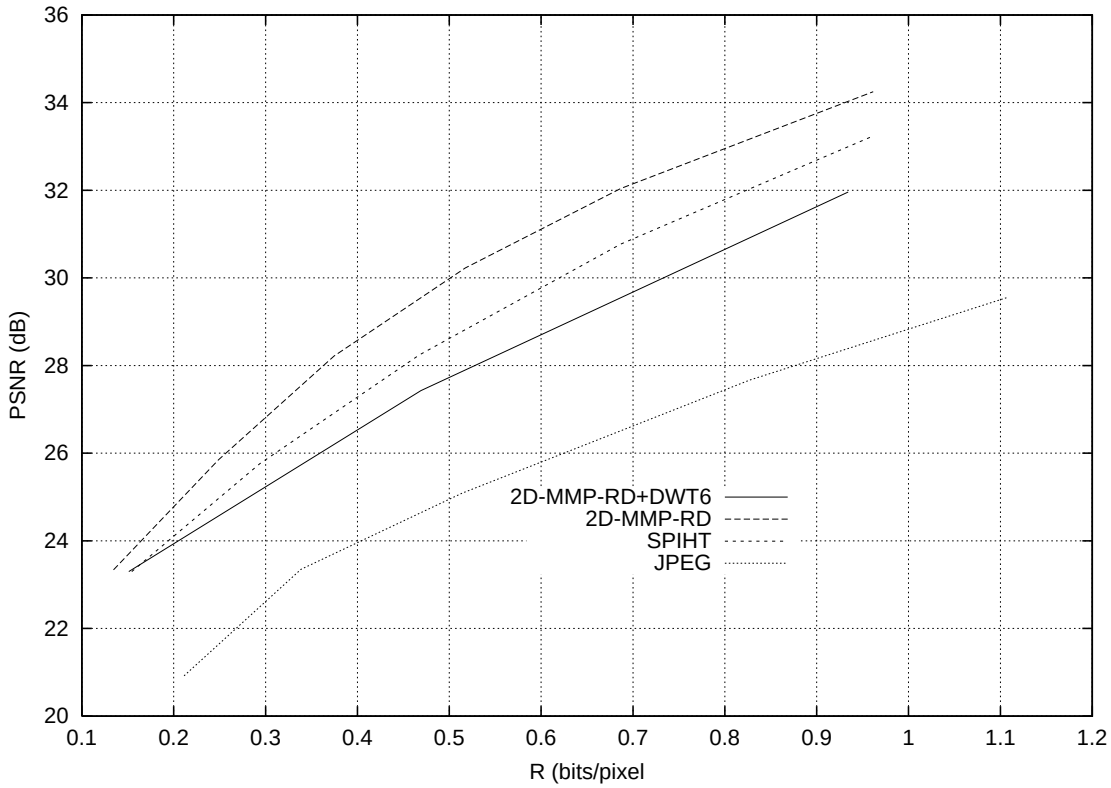


Figura H.13: R-D performance with PP1209 512×512.

Apêndice I

Conclusion

We have presented a new class of algorithms for lossy data compression based on *approximate matching of recurrent patterns using scales*. We have shown that, although not asymptotically optimal for large blocklengths, the matching with scales can outperform ordinary pattern matching for finite blocklengths at low rates.

Two types of algorithms were proposed: The MMP and the UMMP. Both algorithms perform a segmentation of the input vector and attempt to approximate each segment using vectors in a dictionary. The basic difference between them is that UMMP uses a linear combination of scaled vectors in a dictionary to represent an input block, while MMP uses only one vector at a time. In our simulations, the performance of the MMP was always superior but we suspect that the performance of UMMP can be improved if we consider some of its characteristics. For example, the vectors in the dictionaries at different scales tend to have different power, the smaller scale containing vectors of smaller power. Therefore the updating procedure should modify the power of a vector to be included in the dictionary at a given scale to allow a better fit to the statistics of that dictionary. Also, the UMMP algorithm performs a frame expansion. It is well known that the optimal frame for reconstruction is not the frame used for the analysis of the signal [17]. Therefore the UMMP decoder should be modified to reconstruct with the dual frame for optimal performance.

We have also developed a rate-distortion optimized version of MMP with improved performance. It uses dynamic programming concepts to perform the optimization of the segmentation tree of MMP for a given input vector and a control

parameter λ. It provides a significant improvement in performance when compared to the non-optimized MMP.

As many block-based coders, MMP can produce artifacts in the reconstructed signals referred to as blocking effects. In some applications, such as image coding, this kind of distortion can be very annoying. We developed a method for reduction of the blocking-effects on MMP, based on an adaptive post-filtering technique. The subjective quality of the compressed images was clearly improved by our procedure. The objective performance, measured by the PSNR versus the rate, was also improved when we used a particular filter with a Gaussian shape for all pure-grayscale test images at all rates. This suggests that the performance of MMP with this kind of source can be improved if we add some vectors in the dictionary, meeting some smothness criteria. In order to do this, we believe that MMP must be generalized to work with overlapped vectors at the encoder side.

We also applied the MMP algorithm to compress image data in the DWT domain. The algorithm was slightly modified to allow the MMP to learn new symbols as needed. This way we could keep the initial dictionary very small, regardless of the very high dynamic range of the test images in the transformed domain. Some interesting topics yet to be addressed are:

1. The finite-blocklength rate-distortion behavior of FD-MMP with non-Gaussian sources.

2. The performance of FD-MMP using other fidelity measures.

3. The asymptotic FD-MMP R-D performance for long blocklengths.

4. Better flag encoding on UMMP and MMP. These algorithms should perform closer to the random-VQ limit given in section C.

5. Learning rate analysis. It is interesting to find bounds on the size of the dictionary in function of the input sequence size.

6. The standard UMMP and MMP decoders do not use all the information available to reconstruct the input signal. In fact when the encoder chooses a dictionary index to output, it implicitly defines a series of constraints that places the input vector of size N inside a region of the N-dimensional space, instead

of simply specifying a point in that space. This information can be possibly used to make a consistent estimate of the input vector that also meets some smoothness criterion, for example controlling the blocking effect.

7. The UMMP algorithms can be viewed as making an expansion of the input signal in a frame. In a frame expansion, the best frame to reconstruct the input signal, the *dual frame* is usually different from the frame used to calculate the expansion coefficients. The UMMP decoder could be modified to reconstruct the input signal using *inverse frame computations* as in [28].

8. As pointed out in section C, the matching with scales performs better at lower rates. Some kind of compromise relating the extent of the use of scales to the target distortion, or the actual $\mathsf{R}(\mathsf{D})$ behavior, could improve the performance of UMMP and MMP. For example, one could allow the use of a 2:1 scale ratio when performing a match but not 4:1 or higher ratios, depending on the choice of d^*. Other possibility is to dynamically adjust the allowed scaling ratios based on some kind of local performance measure.

9. Generalization of the MMP encoder to work with overlapping vectors.

10. The learning rule of UMMP and MMP attempts to build a dictionary with vectors resembling vectors of samples of the input data. The learning rule could be modified to build instead a dictionary that is optimum, in an $\mathsf{R}(\mathsf{D})$ sense, to code the input data and at the same time is better structured, allowing faster implementations of the algorithm. Another possibility is to use a superset of vectors with some desired approximation properties (for example, composed of vectors that meet some smoothness requirement) as a *reference dictionary* $\mathcal{D}_{\mathtt{ref}}$. The initial dictionary $\mathcal{D}_0$ and the current dictionary $\mathcal{D}$ would always be a subset of this reference dictionary. The dictionary $\mathcal{D}$ could be updated by choosing in the reference dictionary the vector that is closer to the concatenation of the previously encoded vectors, instead of using the concatenation itself.

Apêndice J

Original Images

In this appendix we show the original test images LENA, BABOON, F-16, BRIDGE, AERIAL, BARBARA and GOLD. They are of size 512×512 and can be found at location http://www.sipi.usc.edu. We also show the images PP1205 and PP1209. These images were scanned respectively from pages 1205 and 1209 of the IEEE Transactions on Image Processing, Volume 9, number 7, July 2000.

Figura J.1: Original LENA 512×512 .

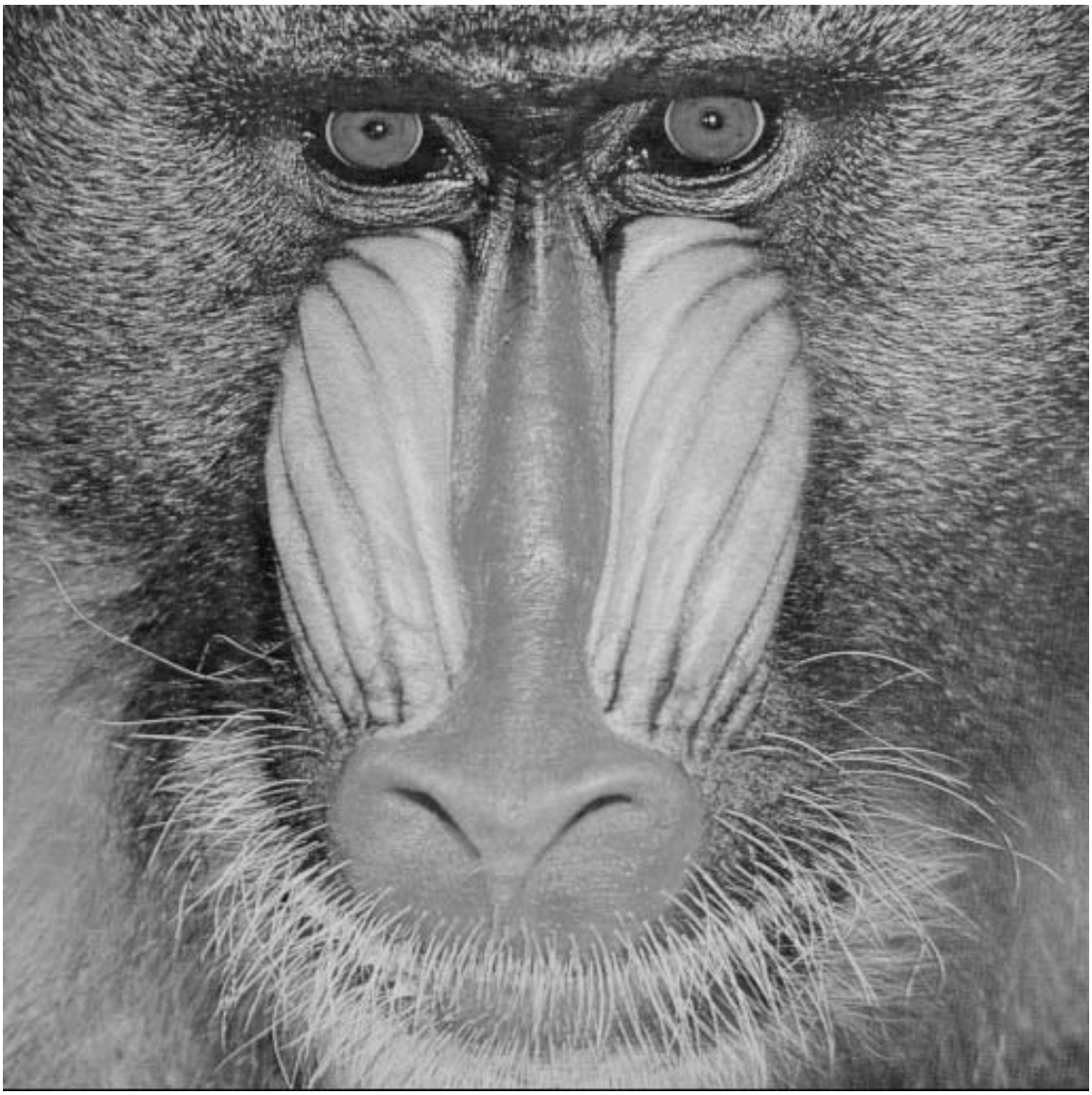


Figura J.2: Original BABOON 512×512 .

Figura J.3: Original F-16 512×512 .

Figura J.4: Original BRIDGE 512×512 .

Figura J.5: Original AERIAL 512×512 .

Figura J.6: Original BARBARA 512×512 .

Figura J.7: Original GOLD 512×512 .

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

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

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

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

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

A (quantized) version of $\hat{\alpha}_\epsilon^{\text{cod}}$, $\hat{\alpha}_r^{\text{cod}}$ and $\hat{\beta}^{\text{cod}}$ is received by the decoder, and denoted, respectively, by m_ϵ^{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

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

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

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

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 α_ϵ, α_r, β by (see Appendix II-A)

$\hat{\alpha}_\epsilon$, $\hat{\alpha}_r$, $\hat{\beta}$

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

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

Algorithm 3

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

$$\frac{1}{\alpha_\epsilon^k} = \mu_\epsilon \frac{1}{m_\epsilon^{cod}} + (1-\mu_\epsilon)\frac{1}{\alpha_\epsilon^{k,dec}} \quad (34)$$

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

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

where

$$\mu_\epsilon = \frac{2l\left(m_\epsilon^{cod}\right)}{2l\left(m_\epsilon^{cod}\right)+p} \quad (37)$$

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

Figura J.8: Original PP1205 512×512 .

TABLE II
PSNR OBTAINED BY ESTIMATING THE PARAMETERS AT THE CODER

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

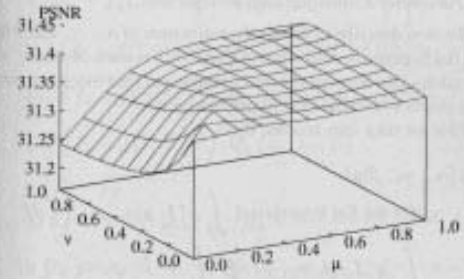


Fig. 6. PSNR for different values of μ and ν on the *Lena* image compressed at 0.29 bpp.

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

The parameters obtained at the coder and the decoder were then combined. The same normalized confidence parameters μ_c and μ_r, defined in (37) and (38), were used for α_c and α_r. The values used in the experiments were $\mu_c = \mu_r = \mu \in \{0.0, 0.1, \cdots, 1\}$. The normalized confidence parameter ν, defined in (39), belongs to the same range. The 3-D plot in Fig. 6 shows the PSNR as a function of μ and ν for the *Lena* highly compressed image. The center part of the compressed image and the best reconstruction, corresponding to the parameter values $m_c^{cod} = \alpha_c^{cod} = 30.82^{-1}$, $m_r^{cod} = \alpha_r^{cod} = 5.36^{-1}$ and $\eta^{cod} = \beta^{cod} = 36.36^{-1}$ 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 μ

(a)

(b)

Figura J.9: Original PP1209 512×512 .

Referências Bibliográficas

[1] C.E. Shannon, "A mathematical theory of communication," Bell Syst. Tech. Journal, vol. 27, pp. 379-423, 1948.

[2] R. E. Blahut, "Principles and Practice of Information Theory" Addison-Wesley publishing Company, 1988.

[3] C.E. Shannon, "Coding theorems for a discrete source with a fidelity criterion," in IRE National Convention Record, Part 4, pp. 142-163, 1959.

[4] A. Papoulis, "Probability, Random Variables and Stochastic Processes,", McGRAW-HILL BOOK COMPANY, 1984.

[5] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Transactions on Information Theory*, vol. it-24, No. 5, pp. 530-536, September 1978.

[6] Y. Linde, A. Buzzo and R. M. Gray "An algorithm for vector quantizer design," IEEE Transactions on Communications, VOL. COM-28, No. 1, January, 1980.

[7] P. A. Chou, T. Lookabaugh and R. M. Gray, "Entropy-constrained vector quantization," IEEE Transactions on Acoustics Speech and Signal Processing, VOL. 37, N0. 1, January 1989.

[8] M. B. de Carvalho and W. A. Finamore, "Lossy Lempel-Ziv on subband coding of images," *IEEE International Symposium of Information Theory*, June 27 - July 1, 1994, Trondheim, Noruega.

[9] T. Łuczak and W. Szpankowski, " A suboptimal lossy date compression based on aproximate pattern matching," IEEE Transactions on Information Theory, Vol. 43, No. 5, september 1997.

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

[11] ISO/IEC JTC1/SC29/WG11, "JPEG2000 Verification Model 5.3 (technical description)", November 1999.

[12] J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 41, pp. 3445-3462, December 1993.

[13] A.Said and W.A. Pearlman, "A new, fast and efficient image codec based on set partitioning in hierarchical trees," *IEEE transactions on circuits and systems for video technology*, vol. 6, pp. 243–250, June 1996.

[14] T.-H. Lan and A. H. Tewfik, "MultiGrid Embedding (MGE) Image Coding" in 1999 IEEE International Conference on Image Processing, Kobe, October 1999.

[15] J. Andrew, "A Simple and Efficient Hierarchical Image Coder" in 1997 IEEE International Conference on Image Processing, Santa Barbara, October 1997.

[16] A. Ortega and K. Ramchandran, "Rate-distortion methods for image and video compression," *IEEE Signal Processing Magazine*, November 1998.

[17] M.Vetterli, J. Kovačević, "Wavellets and Subband Coding," Prentice-Hall, 1995.

[18] A. K. Jain, "Fundamentals of Digital Image Processing," Prentice-Hall, 1989.

[19] H. S. Malvar, D. H. Staelin, "The LOT: trnsform coding without blocking effects," *IEEE Transactions on Acoustics, Speech and Signal Processing*, Vol. 37, No. 4, pp. 553-559, April 1989.

[20] P. P. Vaidyanathan, "Multirate Systems and Filter Banks," Prentice-Hall Inc., 1993.

[21] G. Davis, "Adaptive nonlinear approximations," Ph.D. dissertation, Mathematics Departament, NYU, September 1994.

[22] S. G. Mallat and Z. Zhang, "Matching pursuits with time-frequency dictionaries," *IEEE Transactions on Signal Processing*, vol. 41, No. 12, pp. 3397-3415, December 1993.

[23] M. J. Sabin and R. M. Gray, "Product code vector quantizers for waveform and voice coding," *IEEE Transactions on Acoustics, Speech and Signal Processing*, Vol. ASSP-32, pp. 474-488, June 1984.

[24] M. J. Lighthill, "Introduction to Fourier Analysis and Generalized Functions," New York, Cambridge University Press, 1960.

[25] R. A. DeVore, B. Jawerth and B. J. Lucier, "Image compression through wavelet transform coding," *IEEE Transactions on information Theory*, Vol. 38, No 2, pp 719-746, March 1992.

[26] J. C. Kieffer, T. H. Park, Y. Xu, S. J. Yakowitz, "Progressive lossless image coding via self-referential partitions", *1998 IEEE International Conference on Image Processing*, October 1998, Chigago, Illinois.

[27] R. R. Coifman, Y. Meyer and M .V. Wickerhauser, "Entropy based algorithms for best basis selection". IEEE Transactions on Information Theory, Vol. 32, pp. 712-718, March 1992.

[28] S. Mallat, "A Wavelet Tour of Signal Processing," Academic Press, 1998.

[29] K. H. Tan, M. Ghangari, "Layered image coding using the DCT pyramid," *IEEE Transactions on Image Processing*, Vol. 4, No 4, pp. 512-516, April 1995.

[30] M. Antonini, M. Barlaud, P. Mathieu, and J. Daubechies, "Image coding using wavelet transform," *IEEE Transactions on Image Processing*, Vol. 1, pp.205-221, April 1992.