COMPRESSÃO DE IMAGENS POR APROXIMAÇÕES SUCESSIVAS EM ESPAÇOS DE HILBERT

Alexandre Gomes Ciancio

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

Aprovada por:

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

Prof. Gelson Vieira Mendonça, Ph.D.

Prof. Marcos Craizer, D.Sc.

RIO DE JANEIRO, RJ - BRASIL

JUNHO DE 2001

CIANCIO, ALEXANDRE GOMES

Compressão de Imagens por Aproximações Sucessivas em Espaços de Hilbert[Rio de Janeiro] 2001

XI,72 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia Elétrica, 2001)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1.Processamento de Imagens 2.Aproximações Sucessivas 3.Compressão de Imagens

I.COPPE/UFRJ II.Título (série)

Agradecimentos

Aos meus pais Sergio e Valéria, minha avó Vilma, e minha irmã Alessandra, por terem me dado condições de ter chegado até aqui.

A Cristine, minha (agora) esposa, por estar sempre lá, do seu jeito, muito antes de saber de tudo que viria pela frente.

Ao Eduardo, meu orientador desde a Iniciação Científica, pela força de sempre, pelos grandes desafios e pelas ótimas discussões.

Aos grandes amigos Neto e Francine, Alan e Renata, Junior e Aline, Thiago, pelo eterno apoio e pelos incontáveis "churrasquinhos", campings, caminhadas e outras aventuras, que sempre vieram em boa hora.

Aos amigos do LPS Ailton D. Santana Jr., André C. Vliese, Augusto H. Dantas, Cassio B. Ribeiro, Cassio G. G. Duarte, Charles B. do Prado, Cristiano N. dos Satos, Décio Fonini Júnior, Fábio Pacheco Freeland, Felipe R. Aquino, Maria Heveline V. Duarte, Mauro F. de Carvalho, Lara Cristina R. L. Feio (e Miguelito), Lisandro Livisolo, Luiz Wagner, Manoel Gomes de Pinho, Ranniery da S. Maia e Rogério Caetano (será que esqueci de alguém?) pelas sempre oportunas boas idéias.

Aos professores, pelos excelentes cursos, por estarem sempre abertos a novas idéias e por terem ajudado de tal maneira não só na minha formação profissional, mas também moral.

Aos que, por esquecimento, não foram mencionados, não se preocupem, vocês terão prioridade na Tese de Doutorado.

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

COMPRESSÃO DE IMAGENS POR APROXIMAÇÕES SUCESSIVAS EM ESPAÇOS DE HILBERT

Alexandre Gomes Ciancio

Junho/2001

Orientador: Eduardo Antônio Barros da Silva

Programa: Engenharia Elétrica

Algoritmos de aproximação sucessiva são amplamente utilizados como forma de representação e compressão de sinais digitais. Ao mesmo tempo, transformadas com grande concentração de energia também são usadas para representar sinais e imagens de forma mais eficiente.

Nesse trabalho, é proposto um novo método para compressão de imagens, que trata decomposições em espaços de Hilbert usando aproximações sucessivas, fundindo os passos de transformação e quantização.

Por se tratar de um novo método, o algoritmo inicial teve que sofrer várias modificações e adaptações, descritas ao longo deste trabalho.

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

SUCESSIVE APPROXIMATION IMAGE COMPRESSION IN HILBERT SPACES

Alexandre Gomes Ciancio

June/2001

Advisor: Eduardo Antônio Barros da Silva

Department: Electrical Engineering

Successive approximation algorithms are widely used as a way to represent and compress digital data. One can also use transforms with high energy concentration features as a very efficient way of representing images and signals.

In this work, a novel image compression scheme is proposed. This method uses successive approximations in Hilbert Spaces, merging the transformation and quantization steps.

Since it is a novel method, the initial algoritm had to go through a number of adjustments and modifications, which are described throughout the work.

Sumário

Lista de Figuras

Lista de Tabelas

Introdução

Algoritmos baseados em transformadas wavelet fazem parte do estado-da-arte em compressão de imagens. Uma característica que contribui para isso é que as transformadas wavelet apresentam similaridades entre os coeficientes de mesma orientação em diferentes resoluções. A grande vantagem, porém, é que as wavelets apresentam uma grande concentração de energia e grandes blocos com coeficientes de pequeno valor, que recebem o valor zero quando quantizados. A localização eficiente dos zeros foi explorada [1, 2] levando a excelentes resultados na compressão de imagens.

Com relação à quantização dos coeficientes da transformada, os melhores resultados foram obtidos através de algoritmos de aproximação successiva [1, 2, 3], que garantem a mesma distorção para todas as bandas.

Porém, os algoritmos usuais para aproximações sucessivas são muito dependentes do parâmetro responsável pelo passo de aproximação. Em [3] foi proposto um novo algoritmo que apresentava uma grande independência desse parâmetro, pertimitindo uma maior facilidade na utilização do método.

Percebeu-se mais tarde que o algoritmo proposto em [3] possui um paralelo com decomposições de sinais em sistemas de bases em espaços de Hilbert. Isto é, ele se baseia num conceito mais genérico, que abrange transformadas. Isso permitiria que a representação de um sinal ou imagem através de uma decomposição (por exemplo, uma transformada wavelet) e posterior quantização fosse realizada em um simples passo, através de aproximações sucessivas.

Durante esse trabalho, então, será proposto um novo método para compressão de imagens, baseado em aproximações sucessivas em espaços de Hilbert. Por se tratar de um algoritmo novo, várias modificações em relação à proposta inicial tiveram que ser implementadas para que eventuais problemas, só descobertos durante o de-

senvolvimento do projeto, fossem resolvidos.

Esse texto está organizado da seguinte forma:

- No capítulo 1 foi feita uma introdução aos Espações de Hilbert, passando por espaços vetoriais;
- No capítulo 2 são vistos os métodos usuais para aproximações sucessivas de escalares e vetores, e o algoritmo modificado proposto em [3];
- No capítulo 3 é feita a ponte entre decomposições nos espaços de Hilbert e o algoritmo modificado de aproximações sucessivas;
- No capítulo 4 são feitas considerações a respeito da implementação do algoritmo proposto;
- No capítulo 5 são apresentadas as modificações feitas ao algoritmo proposto inicialmente, devido à problemas encontrados durante o desenvolvimento do projeto;
- No capítulo 6 é mostrado como foi codificada a sequência de saída, de forma melhorar a taxa obtida;
- No capítulo 7 foram apresentados os resultados iniciais, que motivaram as modificações propostas no capítulo 5, e os resultados após a implementação dessas modificações;
- Finalmente, no capítulo 8 são apresentadas as conclusões do trabalho. São feitas também propostas para que os resultados obtidos possam ser melhorados, dando continuidade ao projeto.

Capítulo 1

Espaços de Hilbert

1.1 Introdução

Sejam $\mathbb{R}$ e $\mathbb{C}$ os conjuntos dos números reais e complexos, respectivamente. Os espaços de vetores de dimensão finita n sobre $\mathbb{R}$ ou $\mathbb{C}$ são representados por $\mathbb{R}^n$ e $\mathbb{C}^n$.

Quando consideramos um conjunto de vetores $\{v_k\}$ pertencente a tais espaços, algumas questões importantes devem ser consideradas:

- O conjunto é completo, ou seja, $\{v_k\}$ se expande em $\mathbb{R}^n$ ou $\mathbb{C}^n$? Em outras palavras, qualquer vetor pertencente à $\mathbb{R}^n$ ou $\mathbb{C}^n$ pode ser escrito como uma combinação linear de vetores pertencentes à $\{v_k\}$?
- Os vetores de $\{v_k\}$ são linearmente independentes?
- Como podemos encontrar um conjunto de vetores que se expanda em um espaço?

Além disso, duas definições importantes no estudo desses espaços são a norma de um vetor e o produto interno entre dois vetores pertencentes a um espaço. A norma[1], ou comprimento de um vetor $x \in \mathbb{R}^n$ é representada por

$$\|x\| = \left(\sum_{i=0}^{n} x_i^2 \right)^{1/2} \tag{1.1}$$

[1] Ao longo desse texto trataremos somente da norma quadrada, a não ser quando especificado.

O produto escalar entre dois vetores x e y é determinado por

$$\langle x, y \rangle = \sum_{i=0}^{n} x_i y_i \tag{1.2}$$

Quando $\langle x, y \rangle = 0$ dizemos que x e y são ortogonais.

Os conceitos de norma e ortogonalidade podem ser extendidos a funções através do produto interno entre funções. Nesse caso, as funções são vistas como vetores.

Além disso, a idéia de espaços vetoriais pode ser extendida para o caso de dimensões infinitas. Entretanto, é importante, nesse caso, restringir a norma de cada vetor a um valor finito. Se um subconjunto do conjunto completo $\{v_k\}$ é usado, estamos interessados na melhor aproximação possível de um elemento genérico do espaço. No caso particular de um subconjunto ortonormal e de uma aproximação por mínimos quadrados (que minimiza a norma da diferença), a resposta a esse problema possui uma solução relativamente simples. Uma vez que a geometria dos espaços de Hilbert é similar à Euclidiana, essa solução consiste na projeção ortogonal no subespaço de aproximação, que minimiza a distância e, por consequência, o erro de aproximação..

Espaços vetoriais, espaços de funções, espaços de Hilbert, bases ortonormais e expansões de subespaços serão vistos mais formalmente nas próximas seções.

1.2 Espaços Vetoriais

Um espaço vetorial sobre o conjunto de números complexos ou reais, $\mathbb{C}$ ou $\mathbb{R}$, é um conjunto de vetores, E, o qual para $x, y \in E$ e $\alpha, \beta \in \mathbb{C}$ ou $\mathbb{R}$ satisfaz, além de adição e multiplicação escalar:

- $x + y = y + x$
- $(x + y) + z = x + (y + z)$, $(\alpha\beta)x = \alpha(\beta x)$
- $\alpha(x + y) = \alpha x + \alpha y$, $(\alpha + \beta)x = \alpha x + \beta x$
- existe $0 \in E$ tal que $x + 0 = x$ para todo $x \in E$
- para todo $x \in E$, existe $(-x)$ tal que $x + (-x) = 0$

- $1 \cdot x = x$ para todo $x \in E$

Um subconjunto M de E é um *subespaço* de E se:

(a) para todo x e y pertencentes a M, $x + y$ pertence a M

(b) para todo $x \in M$ e $\alpha \in \mathbb{C}$ ou $\mathbb{R}$, $\alpha x \in M$

Dado $S \subset E$, a *expansão* de S é o subespaço de E constituído por todas as combinações lineares dos vetores em S. Para o caso de dimensões finitas temos:

$$\text{expansão}(S) = \left\{ \sum_{i=1}^{n} \alpha_i x_i | \alpha_i \in \mathbb{C} \text{ ou } \mathbb{R}, x_i \in S \right\} \tag{1.3}$$

Um conjunto de vetores $x_1, \ldots, x_n$ é dito *linearmente independente* se $\sum_{i=1}^{n} \alpha_i x_i = 0$ somente se $\alpha_i = 0$ para todo i. Do contrário, o conjunto é dito *linearmente dependente*.

Um subconjunto $\{x_1, \ldots, x_n\}$ de um espaço vetorial E é uma *base* de E se $E = \text{expansão}(x_1, \ldots, x_n)$ e $x_1, \ldots, x_n$ são linearmente independentes. Nesse caso, dizemos que E tem dimensão n. E terá *dimensão infinita* se contiver um conjunto de vetores linearmente independentes infinito. Um exemplo de espaço vetorial infinito é o espaço das sequências infinitas, que é a expansão do conjunto infinito $\{\delta[n-k]\}_{k \in \mathbb{Z}}$, onde

$$\delta[n-k] = \begin{cases} 1 & \text{se } n = k, \\ 0 & \text{se } n \neq k. \end{cases} \tag{1.4}$$

Como esses vetores são linearmente independentes, o espaço possui dimensão infinita.

Um *produto interno* num espaço vetorial E em $\mathbb{C}$ (ou $\mathbb{R}$) é uma função $\langle \cdot, \cdot \rangle$, definida em $E \times E$ com as seguintes propriedades:

1. $\langle x + y, z \rangle = \langle x, z \rangle + \langle y, z \rangle$
2. $\langle x, \alpha y \rangle = \alpha \langle x, y \rangle$
3. $\langle x, y \rangle^* = \langle y, x \rangle$
4. $\langle x, x \rangle \geq 0$, com $\langle x, x \rangle = 0$ se e somente se $x \equiv 0$

Note que as propriedades 1 e 2 fazem do produto interno uma operação linear. O produto interno de funções complexas em $\mathbb{R}$ e sequências em $\mathbb{Z}$ será dado por

$$\langle f, g\rangle = \int_{-\infty}^{\infty} f^*(t)g(t)dt, \tag{1.5}$$

e

$$\langle x, y\rangle = \sum_{-\infty}^{\infty} x^*[n]y[n], \tag{1.6}$$

respectivamente. A norma de um vetor pode ser definida a partir do produto interno como

$$\|x\| = \sqrt{\langle x, x\rangle}, \tag{1.7}$$

e a distância entre dois vetores x e y é simplesmente a norma da sua diferença $\|x - y\|$.

Além disso, o produto interno pode ser usado para definir ortogonalidade entre vetores. Dois vetores x e y serão ortogonais ($x \perp y$) se e somente se

$$\langle x, y\rangle = 0$$

Um vetor x é dito ortogonal a um conjunto de vetores $S = y_i$ se $\langle x, y_i\rangle = 0$ para todo i. Dois subespaços S_1 e S_2 são ortogonais ($S_1 \perp S_2$), se todos os vetores de S_1 forem ortogonais a todos os vetores de S_2. Um conjunto de vetores $x_1, \ldots, x_n$ é dito ortogonal se $x_i \perp x_j$ para todo $i \neq j$. Se os vetores são normalizados para ter a norma unitária, teremos um *sistema ortonormal*, que satisfaz $\langle x_i, x_j\rangle = \delta[i - j]$. Um sistema ortonormal $x_1, \ldots, x_n$ num espaço vetorial E é uma *base ortonormal* se expansão$(x_1, \ldots, x_n) = E$.

1.3 Espaços de Produto Interno Completo

Um espaço vetorial no qual são realizadas operações envolvendo produtos internos é chamado um *espaço de produto interno*. Considere uma sequência de vetores $\{x_n\}$ em E. Diz-se que essa sequência *converge para* x em E se $\|x_n - x\| \to 0$ quando $n \to \infty$. Uma sequência de vetores $\{x_n\}$ é chamada uma sequência *Cauchy* se $\|x_n - x_m\| \to 0$ quando n, $m \to \infty$. Se toda sequência Cauchy em E converge para um vetor em E, então E é dito *completo*.

Definição

Um espaço de produtos internos completo é um *Espaço de Hilbert.*

Um espaço de Hilbert é dito *separável* se ele contém uma base ortonormal contável.

1.3.1 O Espaço $L_2(\mathbb{R})$

O espaço $L_2(\mathbb{R})$ é um espaço de Hilbert de dimensão infinita, no qual as funções, elevadas ao quadrado, são integráveis. Uma função $f(t)$ definida em $\mathbb{R}$ pertence a $L_2(\mathbb{R})$ se $|f(t)|^2$ for integrável, ou seja, se

$$\int_{t\in\mathbb{R}} |f(t)|^2 dt < \infty \tag{1.8}$$

O produto interno em $L_2(\mathbb{R})$ é dado por

$$\langle f, g\rangle = \int_{t\in\mathbb{R}} f(t)^* g(t) dt, \tag{1.9}$$

e a norma por

$$\|f\| = \sqrt{\langle f, f\rangle} = \sqrt{\int_{t\in\mathbb{R}} |f(t)|^2 dt} \tag{1.10}$$

1.4 Bases Ortonormais

Dentre todas as bases possíveis num espaço de Hilbert, as bases ortonormais possuem um papel muito importante. Veremos a seguir um processo de ortogonalização de uma base genérica.

1.4.1 Processo de Ortogonalização de Gram-Schmidt

Dado um conjunto de vetores linearmente independentes $\{x_i\}$ pertencente a E, podemos chegar a um conjunto ortonormal $\{y_i\}$, com a mesma expansão de$\{x_i\}$ da seguinte maneira:

Comece fazendo

$$y_1 = \frac{x_1}{\|x_1\|}.$$

Em seguida, recursivamente faça

$$y_k = \frac{x_k - v_k}{\|x_k - v_k\|}, \qquad k = 2, 3, \ldots$$

onde

$$v_k = \sum_{i=1}^{k-1} \langle y_i, x_k \rangle y_i.$$

1.4.2 Bases Ortonormais

Para sabermos se um conjunto de vetores $S = \{x_i\}$ constitui uma base ortonormal, primeiro devemos verificar se o conjunto S é ortonormal, e então se ele é completo, ou seja, se cada vetor do espaço a ser representado pode ser escrito como uma combinação linear dos vetores de S. Em outras palavras, um sistema ortonormal $\{x_i\}$ é uma base ortonormal em E se para todo y pertencente a E,

$$y = \sum_k \alpha_k x_k. \tag{1.11}$$

Os coeficientes α_k da expansão são dados por

$$\alpha_k = \langle x_k, y \rangle \tag{1.12}$$

1.4.3 Projeções Ortogonais e Aproximações por Mínimos Quadrados

Muito frequentemente, um vetor num espaço de Hilbert E tem que ser aproximado por vetores pertencentes a um subespaço S. Supondo que S possui uma base ortonormal $\{x_1, x_2, \ldots\}$, a projeção do vetor $y \in E$ em S é dada por

$$\hat{y} = \sum_i \langle x_i, y \rangle x_i$$

Além disso, a diferença $d = y - \hat{y}$ satisfaz $d \perp S$.

A figura 1.1 ilustra a projeção ortogonal de um vetor em um subespaço. Nesse exemplo, $y \in \mathbb{R}^3$ e $\hat{y}$ é a sua projeção na expansão de $\{x_1, x_2\}$. Note que $y - \hat{y}$ é ortogonal à expansão de $\{x_1, x_2\}$.

Uma importante propriedade de tal decomposição é que ela é ótima no sentido dos mínimos quadrados, ou seja, $\min \|y - x\|$ é conseguido para $x = \sum_i \alpha_i x_i$ com $\alpha_i = \langle x_i, y \rangle$. É importante mencionar que essa propriedade é uma caraterística apenas das bases ortogonais.

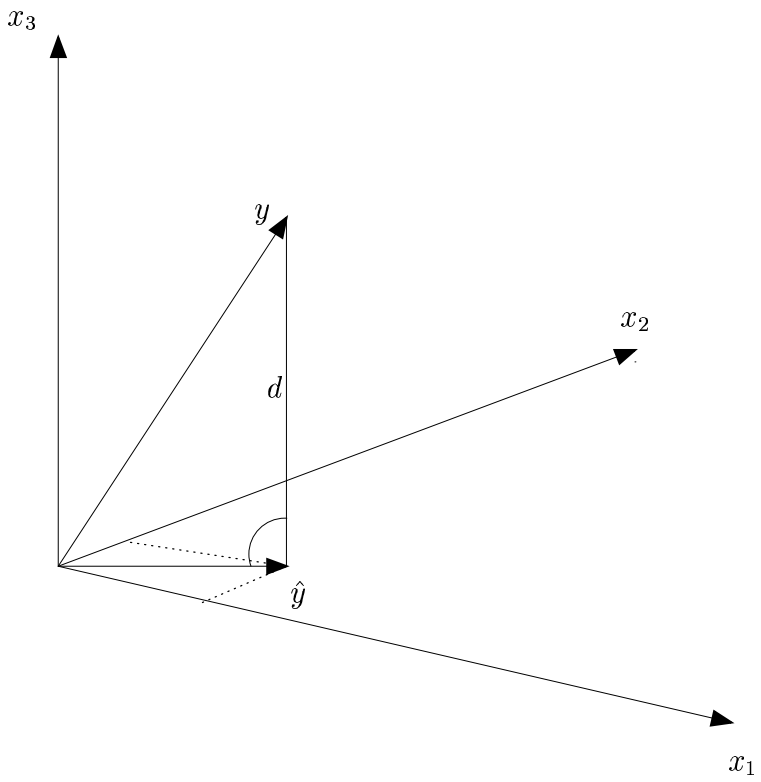

Figura 1.1: Projeção Ortogonal em um Subespaço

Capítulo 2

Método de Aproximações Sucessivas

2.1 Aproximação Sucessiva de Escalares

A aproximação sucessiva de escalares [4] visa representar um escalar, ou comprimento L, por uma soma de valores progressivamente menores. O número de passos, ou seja, o número de valores utilizados na aproximação, está diretamente relacionado ao nível de erro da aproximação.

A figura 2.1 ilustra o processo. Inicialmente, um escalar arbitrário l é escolhido, desde que $l > L$. Pode-se perceber que, a cada passo, a magnitude do erro é limitada pelo valor adicionado, que também se torna menor a cada passo. Sendo assim, aumentando o número de passos, fazemos com que o erro na representação de L diminua progressivamente até um valor arbitrariamente pequeno. Seguindo o algoritmo ilustrado na figura 2.1, podemos expressar o comprimento L como:

$$L = +l - \frac{l}{2} + \frac{l}{4} - \frac{l}{8} + \frac{l}{16} + \frac{l}{32} \ldots \tag{2.1}$$

Sendo assim, um comprimento L pode ser representado por uma sequência de símbolos "+" e "−". A cada símbolo acresentado, a precisão da aproximação é aumentada.

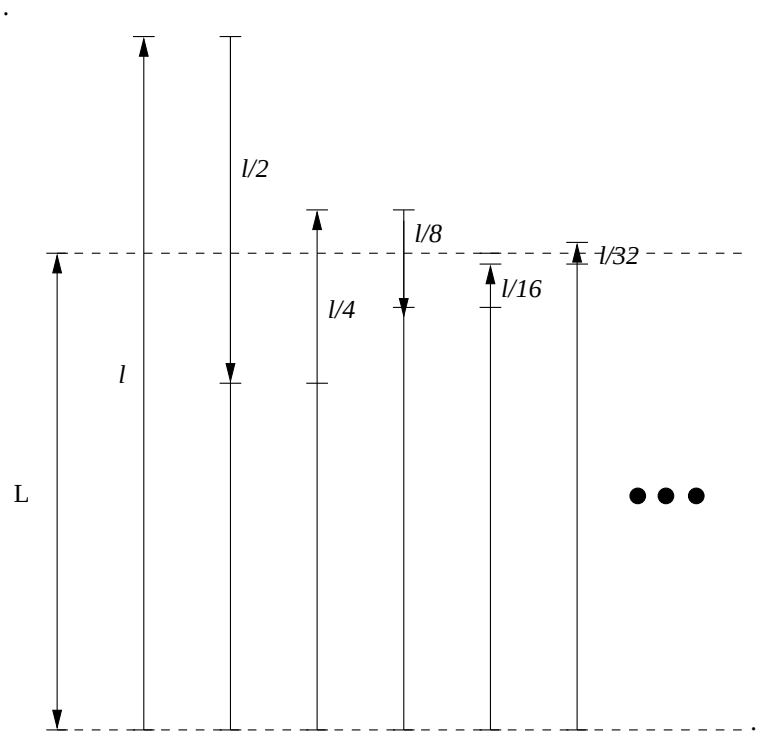

Figura 2.1: Método das Aproximações Sucessivas para o Caso Escalar

2.1.1 Convergência do Algoritmo

No algoritmo de aproximações sucessivas, o valor de l não necessariamente precisa ser reduzido à metada a cada passo. No caso geral, podemos ter um coeficiente α, $\alpha < 1$, que é multiplicado à l. Entretanto, a convergência do algoritmo é garantida para quaisquer valores de α?

Para um valor inicial l, após N passos, podemos expressar L como:

$$L = \sum_{n=0}^{N} (-1)^{\beta_n} l\alpha^n, \qquad \beta_n \in \{0, 1\} \tag{2.2}$$

A convergência do algoritmo implica que um escalar $L = 0$ possa ser aproximado para um número suficiente de passos. Ou seja:

$$\begin{aligned} \exists \beta_n, n = 0, \ldots, N \text{ tal que } \lim_{N\to\infty} \sum_{n=0}^{N} (-1)^{\beta_n} l\alpha^n = 0 \Rightarrow \\ S = \sum_{n=0}^{\infty} (-1)^{\beta_n} l\alpha^n = 0 \end{aligned} \tag{2.3}$$

Definindo

$$B = \sum_{n=0}^{\infty} \alpha^n = \frac{1}{1-\alpha} \tag{2.4}$$

temos, de (2.3) e (2.4), que:

$$S + B = \sum_{n=0}^{\infty} [1 + (-1)^{\beta_n}] \alpha^n = \frac{1}{1-\alpha} \tag{2.5}$$

Uma vez que $\beta_n \in \{0, 1\}$, então $d_n = [1 + (-1)^{\beta_n}] \in \{0, 2\}$, e a equação (2.5) pode ser escrita como

$$S + B = \sum_{n=0}^{\infty} d_n \alpha^n = \frac{1}{1-\alpha}, \qquad d_n \in \{0, 2\} \tag{2.6}$$

Assim, para que o algoritmo de aproximações sucessivas convirja para um dado valor de α, é necessário que existam valores de $d_n \in \{0, 2\}, n = 0, \ldots, \infty$ que garantam essa convergência.

A condição expressa em (2.6) pode ser dividida em dois casos, $d_0 = 0$ e $d_0 = 2$.

(1) $d_0 = 0$

De (2.6), uma vez que $d_0 = 0$,

$$\sum_{n=1}^{\infty} d_n \alpha^n = \frac{1}{1-\alpha} \tag{2.7}$$

Como $d_n \in \{0, 2\}$, o valor de $\sum_{n=1}^{\infty} d_n \alpha^n$ está entre 0 (quando $d_n = 0, \forall n \geq 1$) e $\frac{2\alpha}{1-\alpha}$ (quando $d_n = 2, \forall n \geq 1$).Assim,

$$0 \leq \frac{1}{1-\alpha} \leq \frac{2\alpha}{1-\alpha} \tag{2.8}$$

O que implica que para $d_0 = 0$, a convergência é garantida para valores de α tais que:

$$\frac{1}{2} \leq \alpha < 1 \tag{2.9}$$

(2) $d_0 = 2$

De (2.6), uma vez que $d_0 = 2$,

$$\sum_{n=1}^{\infty} d_n \alpha^n = \frac{1}{1-\alpha} - 2 = \frac{2\alpha - 1}{1-\alpha} \tag{2.10}$$

Da mesma forma que em (1), se $d_n \in \{0, 2\}$,

$$0 \leq \sum_{n=1}^{\infty} d_n \alpha^n \leq \frac{2}{1-\alpha} \tag{2.11}$$

De (2.10) vem que:

$$0 \leq \frac{2\alpha - 1}{1 - \alpha} \leq \frac{2}{1 - \alpha} \tag{2.12}$$

O que implica que para $d_0 = 2$, a convergência também é garantida para valores de α tais que:

$$\frac{1}{2} \leq \alpha < 1 \tag{2.13}$$

Assim, de (1) e (2), concluimos [4] que o algoritmo de aproximações sucessivas converge para valores de α no intervalo $[0.5, 1)$.

Uma vez que a magnitude do erro de aproximação é limitada por $\alpha^n l$, podemos concluir que, quanto menor o valor de α, menor será o erro a cada passo, e menor será o número de passos necessários para se atingir um mesmo nível de distorção. Assim, devemos procurar utilizar o menor valor possível para α, ou $\alpha = 0.5$.

2.2 Quantização Vetorial

Na quantização vetorial [5, 4] representamos vetores, pertencentes a um espaço de dimensão N, com um número determinado e limitado de vetores, que chamaremos K, neste espaço.

Seja $Q : \mathbb{R}^N \rightarrow C$ uma função de mapeamento de $\mathbb{R}^N$ em C, onde C é um conjunto finito. Chamaremos C de dicionário, e faremos: $C = \{\mathbf{v_1}, \mathbf{v_2}, ..., \mathbf{v_K}\}$ e $\mathbf{v_i} \in \mathbb{R}^\mathbf{N}$ para todo $i \in I = \{1, 2, ..., K\}$.

Podemos dividir o espaço $\mathbb{R}^N$ em K regiões R_i, $i \in I$. Teremos K regiões ou células, e cada região será dada por:

$$R_i = \{\mathbf{x} \in \mathbb{R}^\mathbf{N} : \mathbf{Q(X)} = \mathbf{v_i}\} \tag{2.14}$$

onde $Q(X)$ é chamada de função de quantização.

Um vetor genérico $\mathbf{w} \in \mathbb{R}^N$ será representado na quantização vetorial pela região R_i à qual o mesmo pertence. Assim,

$$Q(\mathbf{w}) = \mathbf{v}_i, \qquad \mathbf{w} \in R_i, \tag{2.15}$$

Entretanto, uma vez que conhecemos o conjunto dicionário, podemos representar $\mathbf{v}_i$ ou R_i simplesmente pelo índice i que será o índice correspondente à codificação de $\mathbf{w}$.

Podemos então imaginar que, ao invés de uma função de quantização Q como na equação (2.15), teremos uma uma função de quantização $Q'(\mathbf{w}) : \mathbb{R}^N \rightarrow I$ onde $I = \{1, 2, ..., K\}$. Porém, devemos observar que ao fazermos a conversão, ou seja, o mapeamento inverso $Q'^{-1}(i) : I \rightarrow \mathbb{R}^N$, teremos para os possíveis valores de $Q'^{-1}(i)$ os vetores $\mathbf{v}_k$ onde $k \in I$ e $\mathbf{v}_k \in C$. Como cada vetor $\mathbf{v}_k$ representa uma região R_i, e essa região é a região à qual pertencia o vetor $\mathbf{w}$, o mapeamento inverso acarreta num erro de aproximação devido à quantização, a não ser no caso onde $\mathbf{w} = \mathbf{v}_i$. Este erro será devido à diferença entre o vetor $\mathbf{w}$ e o vetor $\mathbf{v}_k$ escolhido para representá-lo.

A função de quantização $Q(\cdot)$ deve ser tal que considere o vetor de entrada $\mathbf{w}$ e os vetores $\mathbf{v}_k \in C$ de forma a determinar o índice k correspondente ao vetor $\mathbf{w}$. Assim, dada uma função de distorção $d(\mathbf{w}, \mathbf{v}_k)$, podemos expressar a função de quantização $Q(\cdot)$ como:

$$Q(\mathbf{w}) = \mathbf{v}_k \text{ tal que: } d(\mathbf{w}, \mathbf{v}_k) \leq d(\mathbf{w}, \mathbf{v}_i),\ \forall k \neq i \tag{2.16}$$

A condição de menor ou igual na equação (2.16) é importante no caso de termos um vetor de entrada $\mathbf{w}$ que apresente uma mesma distorção[1] para dois $\mathbf{v}_k$ distintos pertencentes a C. Observamos que esta é uma extensão ao caso unidimensional de quantização escalar [5, 4], porém para direções pré-estabelecidas do espaço $\mathbb{R}^N$. Uma vez que um vetor de dimensão N é a composição (soma) de N segmentos, sendo cada segmento pertencente a uma das direções do espaço $\mathbb{R}^N$, a quantização vetorial nada mais é do que uma extensão da quantização escalar [5] realizada em cada uma das N dimensões.

A figura 2.2 ilustra um esquema muito utilizado para quantização vetorial em um espaço bidimensional. No caso, o espaço $\mathbb{R}^2$ foi particionado em células hexagonais. Neste caso, qualquer vetor $\mathbf{U}$ que corresponde ao ponto p, pertencente à região j - chamado de *célula* j; será representado pelo vetor $\mathbf{V}_j$ ou pelo centróide [5] da célula j, c_j. Podemos ainda dizer que o vetor $\mathbf{U}$ será representado pelo índice j, caso possamos fazer a relação $j \rightarrow c_j$.

[1] Utilizamos a palavra distorção ao invés de erro para manter a generalidade.

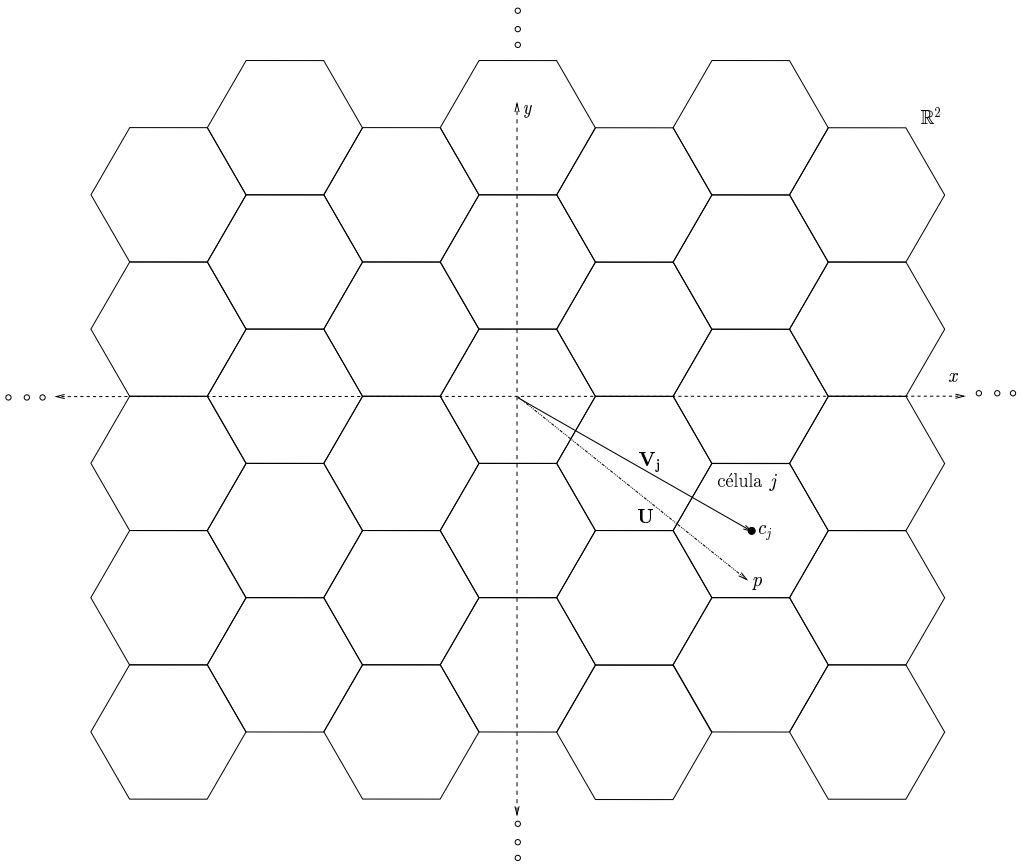

Figura 2.2: Quantização Vetorial

2.3 Quantização Vetorial por Aproximações Sucessivas

Na quantização vetorial por aproximações sucessivas (QV-AS) [6, 3], um vetor $\mathbf{x}$ é aproximado por um vetor $\mathbf{x}_n$ conforme a equação abaixo:

$$\mathbf{x}_n = \sum_{j=1}^{n} \alpha^j \mathbf{v}_{i_j} \tag{2.17}$$

onde $\mathbf{v}_{i_j} \in C_N = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_K\}$, e $0 < \alpha < 1$.

C_N é um dicionário composto de vetores com norma unitária, K é a cardinalidade do dicionário (número de elementos) e n é o número de passos utilizados na quantização-aproximação.

Este tipo de aproximação de um vetor pode ser visto na figura 2.3. Nesta figura $\mathbf{x}_n$ é o vetor aproximado após o passo n; $\mathbf{r}_n$ é o resíduo após o passo n - a

diferença na aproximação após o passo n, que ainda necessita ser codificado; e θ_i o ângulo entre o vetor que está sendo aproximado no passo n e o vetor $\mathbf{v}_{i_n}$, pertencente ao dicionário escolhido para a aproximação neste passo, n.

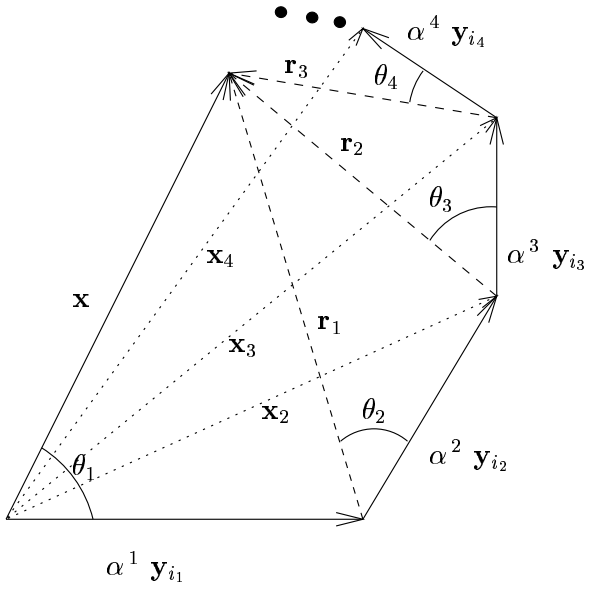

Figura 2.3: QV-AS

Um algoritmo que implementa a quantização vetorial por aproximaçaões sucessivas (QV-AS) pode ser visto a seguir. Neste algoritmo, T é um limiar de erro pré-estabelecido e N o número máximo de passos permitidos.

ALGORITMO 2.1 - QV-AS

1. Fazer $\mathbf{w} = \mathbf{x}$, onde $\mathbf{x}$ é o vetor a ser codificado, e $n = 1$
2. Achar $\mathbf{v}_{i_n} \in C$ tal que:
$$\mathbf{w} \cdot \mathbf{v}_{i_n} = \max_{\mathbf{v}_j \in C} \mathbf{w} \cdot \mathbf{v}_j$$
3. Fazer $\mathbf{w} = \mathbf{w} - \alpha^n \mathbf{v}_{i_n}$
4. Incrementar n de 1.
5. Se $\| \mathbf{w} \| < T$ ou $n = N$, pare; senão volte para o segundo passo.

Uma representação deste tipo será útil somente se, ao aumentarmos o número de passos N, e, portanto, o número de vetores utilizados para aproximar $\mathbf{x}$, o erro

de aproximação, $e_n = \|\mathbf{x} - \mathbf{x}_n\|$, tender a zero. Em [3] foi demonstrado que uma condição suficiente para isto é:

$$\begin{cases} \alpha \geq \frac{1}{2\cos\Theta(C_N)} , & \text{para } \Theta(C_N) \leq \frac{\pi}{4} \\ \alpha \geq \sin\Theta(C_N) , & \text{para } \Theta(C_N) \geq \frac{\pi}{4} \end{cases} \tag{2.18}$$

onde $\Theta(C_N)$ é o ângulo máximo entre qualquer vetor $\mathbf{x} \in \mathbb{R}^N$ e o seu vizinho mais próximo do dicionário C_N, isto é:

$$\Theta(C_N) = \max_{\|\mathbf{x}\|=1} \{ \min_{\mathbf{v}_n \in C_N} [\arccos(\mathbf{x} \cdot \mathbf{v}_n)] \} \tag{2.19}$$

2.3.1 Considerações sobre o Algoritmo QV-AS

A equação (2.17) nos mostra que no algoritmo QV-AS um vetor pode ser expresso como uma sequência de inteiros $i_1, i_2, \ldots, i_n$ correspondentes aos vetores $\mathbf{v}_{i_1}, \mathbf{v}_{i_2}, \ldots, \mathbf{v}_{i_n}$ escolhidos do dicionário.

Assim como no algoritmo *Matching Pursuit* proposto por Mallat [7], o algoritmo QV-AS minimiza o valor do erro de aproximação a cada passo k, uma vez que o método busca o vetor $\mathbf{v}_{i_k}$ do dicionário mais próximo ao resíduo r_{i_k}. Entretanto, esse procedimento não garante o erro mínimo para o algoritmo como um todo. Ou seja, dada uma sequência de vetores $\mathbf{v}_{i_1}, \mathbf{v}_{i_2}, \ldots, \mathbf{v}_{i_n}$, que representa uma aproximação $\mathbf{x}_n$ para um vetor $\mathbf{x}$, não há garantia de que não haja uma outra sequência $\mathbf{v}_{j_1}, \mathbf{v}_{j_2}, \ldots, \mathbf{v}_{j_n}$, representando uma aproximação $\mathbf{x}'_n$ tal que $\|\mathbf{x} - \mathbf{x}'_n\| < \|\mathbf{x} - \mathbf{x}_n\|$.

Uma outra característica do algoritmo QV-AS é a sua dependência do parâmetro α. Conforme pode ser visto em [3], o algoritmo QV-AS possui curvas taxa $\times$ distorção que, para diferentes funções de entrada, não apresentam uma boa estabilidade em função de α. Assim, cada imagem a ser codificada possui um valor de α ótimo diferente, para o qual melhores taxas são obtidas para um mesmo nível de distorção.

As condições estabelecidas em (2.18) mostram que o parâmetro α deve ser escolhido de acordo com o valor de $\Theta(C_N)$. Além disso, foi visto que o erro a cada passo é proporcional ao valor de α. Sendo assim, os valores de α devem ser escolhidos de forma a serem os menores possíveis, pois assim estaremos minimizando o erro de aproximação, e fazendo com que menos passos sejam necessários para que se atinja um determinado nível de distorção. Entretando, vale salientar que, dados

dois dicionários com dois conjuntos diferentes de vetores, aquele que apresentar um valor de $\Theta(C_N)$ inferior será, a princípio, mais adequado à codificação, pois esse conjunto permitirá a escolha de um valor de α menor. Logo, um conjunto de N vetores será otimizado como um dicionário se possuir, para aquele número de vetores, o menor $\Theta(C_N)$ possível. Isso será conseguido simplesmente escolhendo um conjunto de vetores o mais uniformemente distribuído possível.

Entretanto, os limites inferiores para o valor de α nas condições (2.18) também supõem que a cada passo, o ângulo entre o vetor mais próximo escolhido e o resíduo correspondente é sempre igual a $\Theta(C_N)$, o que dificilmente occorre na prática. Sendo assim, verifica-se que melhores resultados são obtidos para valores de α inferiores aos limites estabelecidos nas equações (2.18).

2.4 Algoritmo QV-AS Modificado

Nesta seção será apresentado o algoritmo QV-AS modificado, proposto em [3]. A principal característica deste algoritmo é que ele converge para uma maior faixa de valores de α. Além disso, as curvas taxa $\times$ distorção também se mostram muito mais independentes de α.

Para que o algoritmo convirja para uma maior faixa de valores de α, ele deve ser modificado para garantir que $\alpha^{n+1}X_{max} \leq \|r_n\| \leq \alpha^n X_{max}$ a cada passo, onde X_{max} é a norma do vetor a ser aproximado.

Na prática, essa modificação implica que agora, ao invés de apenas um vetor relacionado a cada expoente de α, como na equação (2.17), podemos ter vários. Agora teremos um conjunto de vetores associado a cada valor do expoente de α. O algoritmo QV-AS modificado pode ser escrito como:

$$\mathbf{x}_{n,l} = \sum_{j=1}^{n} \alpha^j X_{max} \sum_{k=1}^{l} \mathbf{v}_{i_{j,k}} \tag{2.20}$$

onde $\mathbf{v}_{i_{j,k}} \in C_N$, o conjunto dicionário. O vetor a ser aproximado $\mathbf{x}$ será expresso por uma sequência $(i_{j,k})$, $0 \leq (i_{j,k}) \leq q$, onde q é o número de vetores em C_N.

Um algoritmo para a obtenção da sequência $(i_{n,l})$ pode ser visto a seguir:

ALGORITMO 2.2 - QV-AS MODIFICADO

1. Fazer $\mathbf{w} = \mathbf{x}$, onde $\mathbf{x}$ é o vetor a ser codificado, $n = 1$, e $l = 1$.

2. Se $\|\mathbf{w}\| \leq \alpha^n X_{max}$, faça $i_{n,l} = 0$.

3. Se $\|\mathbf{w}\| > \alpha^n X_{max}$, achar $i_{n,l} \in \{1, 2, \ldots, q\}$ tal que:

$$\mathbf{w} \cdot \mathbf{v}_{i_{n,l}} = \max_{1 \leq k \leq q} \{\mathbf{w} \cdot \mathbf{v}_k\}$$

4. Fazer $\mathbf{w} = \mathbf{w} - \alpha^n X_{max} \mathbf{v}_{i_{n,l}}$.

5. Se $\|\mathbf{w}\| > \alpha^n X_{max}$, faça $l = l + 1$ e volte ao passo 3.

6. Incrementar n de 1 e faça $l = 1$.

7. Se $\| \mathbf{w} \| < T$, onde T é um limiar pré-determinado, pare; senão volte para o passo 1.

Conforme foi visto na seção 2.3, o algoritomo QV-AS possui convergência garantida, qualquer que seja o conjunto dicionário, desde que as condições estabelecidas nas equações (2.18) sejam cumpridas. Entretanto, o algoritmo 2.2 apresentado garante a convergência para quaquer $0 < \alpha < 1$, desde que o conjunto dicionário apresente $\Theta(C_N) < \pi/3$, uma condição que terá implicações importantes nos capítulos seguintes. A prova matemática para esta condição pode ser encontrada em [3].

Capítulo 3

Transformadas Vistas Como Aproximações Sucessivas

3.1 Expansão de Sinais em Séries

Seja um sinal x pertencente a um espaço S, onde S pode ter dimensão finita (como $\mathbb{R}^n$ ou $\mathbb{C}^n$) ou infinita (como $L_2(\mathbb{R})$). Queremos encontrar um conjunto de sinais elementares $\{\psi_i\}_{i\in\mathbb{Z}}$ naquele espaço, de forma que possamos escrever x como um combinação linear

$$x = \sum_i \alpha_i \psi_i. \tag{3.1}$$

Conforme mencionado na seção 1.3, o conjunto $\{\psi_i\}$ será completo em S se todos os sinais $x \in S$ puderem ser expressos como em (3.1). Nesse caso, existirá um conjunto dual $\{\tilde{\psi}_i\}$ tal que os coeficientes α_i em (3.1) possam ser calculados da forma

$$\alpha_i = \sum_n \tilde{\psi}_i[n]x[n], \tag{3.2}$$

se x e $\tilde{\psi}_i$ são sequência reais e discretas, e

$$\alpha_i = \int \tilde{\psi}_i(t)x(t)dt, \tag{3.3}$$

se são funções reais contínuas no tempo. Conforme visto na seção 1.2, as equações acima nada mais são do que os produtos internos entre os $\tilde{\psi}_i$'s e x, denotado por $\langle\tilde{\psi}_i, x\rangle$.

Quando o conjunto $\{\psi_i\}$ é *ortonormal e completo*, teremos uma *base ortonormal* de S e a base e sua dual serão iguais, ou seja, $\psi_i = \tilde{\psi}_i$. Se o conjunto é completo, e os vetores ψ_i são linearmente independentes, mas não são ortonormais, então teremos uma *base biortogonal*. Finalmente, se o conjunto $\{\psi_i\}$ é completo, mas seus vetores são redundantes, ou seja, não são linearmente independentes, teremos uma representação chamada de *frame*.

3.1.1 Transformadas Genéricas

Uma vez que o conjunto $\{\psi_i\}$ é conhecido, podemos então representar o sinal x apenas pelos coeficientes α_i. Em outral palavras, podemos dizer que agora, x está sendo *representado em outra base*, no caso, $\{\psi_i\}$, e α_i são os *coefientes da transformação*.

Mais formalmente, seja $f(t)$ um sinal contínuo definido em $\mathbb{R}$ ou $\mathbb{C}$. Seja $\{\psi_n\}$ o conjunto base da transformada, um subespaço de $\mathbb{R}$ ou $\mathbb{C}$. Podemos expressar $f(t)$ como

$$f(t) = \sum_{n=-\infty}^{\infty} c_n \psi_n(t), \tag{3.4}$$

onde c_n, os coeficientes da transformada, ou seja, produto interno entre cada base e o sinal $f(t)$, é dado por

$$c_n = \int \tilde{\psi}_n(t) f(t) dt. \tag{3.5}$$

As equações (3.4) e (3.5) definem a transformada correspondente ao conjunto de bases $\{\psi_n\}$.

3.2 Transformadas Vistas como Aproximações Sucessivas

Utilizando uma representação binária de c_n teríamos:

$$c_n = s_n \sum_{k=0}^{\infty} 2^{-k} b_{k,n}, \qquad s_n = \{-1, 1\}, b_{k,n} = \{0, 1\}. \tag{3.6}$$

É importante notar que essa representação binária de c_n pode também ser vista como a quantização do coeficiente c_n. A precisão dessa quantização está ligada ao

número de passos utilizados no somatório em k na equação (3.6).

Assim, substituindo (3.6) em (3.4) temos

$$\begin{aligned} f(t) &= \sum_{n=-\infty}^{\infty} s_n \sum_{k=0}^{\infty} 2^{-k} b_{k,n} \psi_n(t), \\ &= \sum_{k=0}^{\infty} 2^{-k} \sum_{n=-\infty}^{\infty} s_n b_{k,n} \psi_n(t). \end{aligned} \tag{3.7}$$

Uma vez que $s_n = \{1, -1\}$, podemos definir o conjunto $\phi_n(t) = \{\psi_n(t) \cup -\psi_n(t)\}$, e a equação (3.7) ficaria como

$$f(t) = \sum_{k=0}^{\infty} 2^{-k} \sum_{n=-\infty}^{\infty} b_{k,n} \phi_n(t). \tag{3.8}$$

Devemos agora notar a correspondência entre as equações (3.8) e (2.20), aqui reproduzida para maior comodidade:

$$\mathbf{x}_{n,l} = \sum_{j=1}^{n} \alpha^j X_{max} \sum_{k=1}^{l} \mathbf{v}_{i_{j,k}}. \tag{3.9}$$

Se o sinal a ser aproximado possui norma unitária, então $X_{max} = 1$. Assim, a equação (3.8) pode ser vista como o cálculo de uma transformada e posterior quantização de seus coeficientes, e pode ser computada em uma só etapa através do algoritmo de aproximações sucessivas definido pela equação (3.9), onde $\alpha = \frac{1}{2}$ e o conjunto $\{\mathbf{v}_k\} = \{\phi_n(t)\}$.

No caso de imagens ou sinais digitais, o conjunto base da transformada $\{\phi_n\}$ é constituído por um conjunto de vetores discretos, $\phi_n(n)$.

Capítulo 4

Implementação do Algoritmo

4.1 Introdução

Conforme visto no capítulo anterior, uma transformada quantizada pode ser implementada através do algoritmo 2.2, visto na seção 2.4. É importante notar que a base da transformada em questão deve agora satisfazer as condições de convergência do algoritmo, mais especificamente $\Theta(\phi_n(t)) < \frac{\pi}{3}$.

A idéia por trás da representação de uma transformada e posterior quantização através do algoritmo de aproximações sucesivas, conforme explicado no capítulo anterior, é que uma imagem f passaria a ser representada apenas por uma sequência de índices, relacionados ao conjunto de vetores-base (dicionário) da transformada em questão.

Assim, considere a imagem f. Seguindo o algoritmo 2.2, f poderá ser expressa por

$$\begin{array}{rcll} f & = & \alpha^1 \vec{v}_{1,1} & +R_1 \\ & = & \alpha^1(\vec{v}_{1,1} + \vec{v}_{1,2}) & +R_2 \\ & = & \alpha^1(\vec{v}_{1,1} + \vec{v}_{1,2} + \ldots + \vec{v}_{1,l_1}) & +R_{l_1} \\ \vdots & = & \vdots & \vdots \\ f & = & \alpha^1(\vec{v}_{1,1} + \ldots + \vec{v}_{1,l_1}) + \alpha^2(\vec{v}_{2,1} + \ldots + \vec{v}_{1,l_2}) & +R_{l_1+l_2} \\ \vdots & & \vdots & \vdots \\ \Rightarrow \quad f & = & \displaystyle\sum_{i=1}^{\infty} \alpha^i \sum_{k=0}^{l_i} \vec{v}_{i,k} & +R_\infty \end{array} \tag{4.1}$$

onde R_n é o resíduo num determinado passo n. Obviamente, quando $n \to \infty$,

$R_n \to 0$, e

$$f = \sum_{i=1}^{\infty} \alpha^i \sum_{k=0}^{l_i} \vec{v}_{i,k}. \tag{4.2}$$

Percebe-se que para cada valor do exponte i de α existe um conjunto de l_i vetores relativos àquele expoente.

Se, no entanto, utilizarmos apenas Q passos, teremos então uma imagem aproximada

$$f_Q = \sum_{i=1}^{q} \alpha^i \sum_{k=0}^{l_i} \vec{v}_{i,k} + R_Q, \tag{4.3}$$

onde, à rigor,

$$Q = \sum_{i=1}^{q} l_i,$$

e q é o expoente máximo atingido por α.

Na prática, para calcular a sequência de vetores que expressa f, devemos, a cada passo, procurar o vetor $\vec{v}_{i,k}$ que minimiza o resíduo

$$R_Q = f - \sum_{i=1}^{q} \alpha^i \sum_{k=0}^{l_i} \vec{v}_{i,k}, \tag{4.4}$$

sabendo que todos os vetores anteriores já são conhecidos.

Entretanto, como pode ser observado, para que o vetor que minimiza o resíduo R_Q seja encontrado, é necessário que sejam considerados todos os vetores calculados anteriormente no somatório da equação (4.4). Esse somatório envolve matrizes com as dimensões da imagem f, e o resíduo mínimo será aquele que apresentar a menor norma para a expressão em (4.4). Todos esses cáculos se traduzem num esforço computacional que inviabiliza o algoritmo.

Uma pequena modificação que torna a implementação do algoritmo viável é explicada na próxima seção.

4.2 Minimização do Esforço Computacional

Seja $\vec{v}$ o vetor a ser encontrado no passo Q, e sejam $\{\vec{v}_{i,k}\}$ os vetores encontrados nos passos anteriores. Fazendo o produto interno nos dois lados da equação (4.4), temos:

$$\langle R_Q; \vec{v} \rangle = \langle f; \vec{v} \rangle - \sum_{i=1}^{q} \alpha^i \sum_{k=0}^{l_i} \langle \vec{v}_{i,k}; \vec{v} \rangle. \tag{4.5}$$

Devemos agora buscar o vetor $\vec{v}$ que minimiza $\langle R_Q; \vec{v} \rangle$. Assim, uma vez que o conjunto dicionário $\{v_k\}$ já é conhecido antes de se iniciar o algoritmo, podemos fazer algumas observações:

1. $\langle f; \vec{v} \rangle$ é a projeção da imagem f no conjunto $\{v_k\}$, e corresponde à decomposição da imagem original nesse dicionário, que pode ser calculada antes do início do algoritmo;

2. $\langle \vec{v}_{i,k}; \vec{v} \rangle$ é o produto interno entre cada vetor do dicionário e todos os outros vetores; esses valores também podem ser calculados de antemão e guardados em uma tabela;

Sendo assim, a cada passo, o algoritmo terá apenas que fazer uma busca em tabelas de escalares pré-calculadas para chegar ao vetor mais próximo do resíduo naquele passo.

4.3 Escolha do Sistema de Bases

Inicialmente, o dicionário escolhido para a representação de imagens através do método proposto foi baseado nas transformadas *wavelet*. As transformadas wavelet são extensivamente utilizadas em diversos métodos para compressão e codificação de imagens, apresentando ótimos resultados. Assim, poderia ser feita uma comparação entre o algoritmo proposto e tais métodos. Entretanto, na prática utilizamos uma decomposição num sistema de bases que corresponde a uma transformada wavelet gerada utilizando-se a mesma estrutura dos bancos de filtros, porém sem subamostragem. Uma descrição mais detalhada das tranformadas wavelet e dos referidos bancos de filtro pode ser encontrada no Apêndice A.

4.4 Extensão Utilizada Durante a Decomposição

Conforme descrito no apêndice A, nas filtragens para o cálculo da transformada wavelet pode-se utilizar extensões periódicas, simétricas ou antisimétricas, dependendo do tipo dos filtros de entrada e se o número de coeficientes desses filtros é par ou ímpar. No caso das bases utilizadas, a mesma analogia pode ser feita, e

deveríamos levar em conta o número de coeficientes dos vetores na equação (4.5) na escolha da extensão mais adequada.

A escolha pela extensão periódica diminui em muito o esforço computacional no cálculo da tabela de produtos internos, uma vez que casos de borda passam a ser vistos como casos triviais. No caso da extensão simétrica ou antisimétrica, cada deslocamento no qual um ou mais coeficientes de um vetor do dicionário em questão atinja a borda da imagem deve ser considerado como uma caso de borda diferente, e cada um desses deslocamentos irá originar um novo vetor, com os coeficientes reescalados de modo que a energia do vetor original continue unitária. Isso aumentaria em muito o número de vetores que, conforme veremos na sessão a seguir, já não é pequeno.

4.5 Cálculo de $\langle f; \vec{v} \rangle$

Conforme dito anteriormente, $\langle f; \vec{v} \rangle$ corresponde à decomposição da imagem original na base escolhida. Sendo assim, uma vez que estaremos utilizando um dicionário semelhante ao da transformada wavelet, o cálculo de $\langle f; \vec{v} \rangle$ será feito conforme descrito no apêndice A, sem os passos de subamostragem. É importante ressaltar que o sistema de bases (dicionário) utilizado corresponde à uma transformada wavelet, sem subamostragem, e é calculada através da mesma estrutura de bancos de filtros da wavelet usual, sem os blocos de decimação e interpolação. Um outro ponto em comum é que os vetores bidimensionais gerados são também separáveis (ver apêndice A), o que significa que as decomposições bidimensionais serão feitas através de decomposições unidimensionais, primeiramente nas linhas e, a seguir, nas colunas da imagem.

Na prática, podemos partir dos vetores (unidimensionais) características da decomposição $H_0(z)$ e $H_1(z)$, sendo que $H_0(z)$ é um passa-baixas e $H_1(z)$ um passa-altas. A cada estágio da decomposição são geradas quatro imagens de saída (através da convolução com os vetores), com as mesmas dimensões da imagem original, uma vez que a estrutura do banco de filtros está sendo usada sem subamostragem. Dessas quatro imagens, três são relativas às bandas de alta frequência, e uma será utilizada como imagem de entrada para o cáculo de quatro novas imagens no próximo

estágio, até o último estágio, quando todas as quatro constituirão imagens de saída. Entretanto, os novos vetores utilizados nas decomposições nos estágios subseqüentes (a partir do primeiro) serão calculados através da interpolação de zeros nos dois vetores originais. Assim, os vetores interpolados relativos ao estágio n da decomposição serão então expressos por $H_0(z^{2^n})$ e $H_1(z^{2^n})$. Para uma decomposição com N estágios, serão geradas $3N + 1$ imagens de saída. A figura 4.1 ilustra o processo.

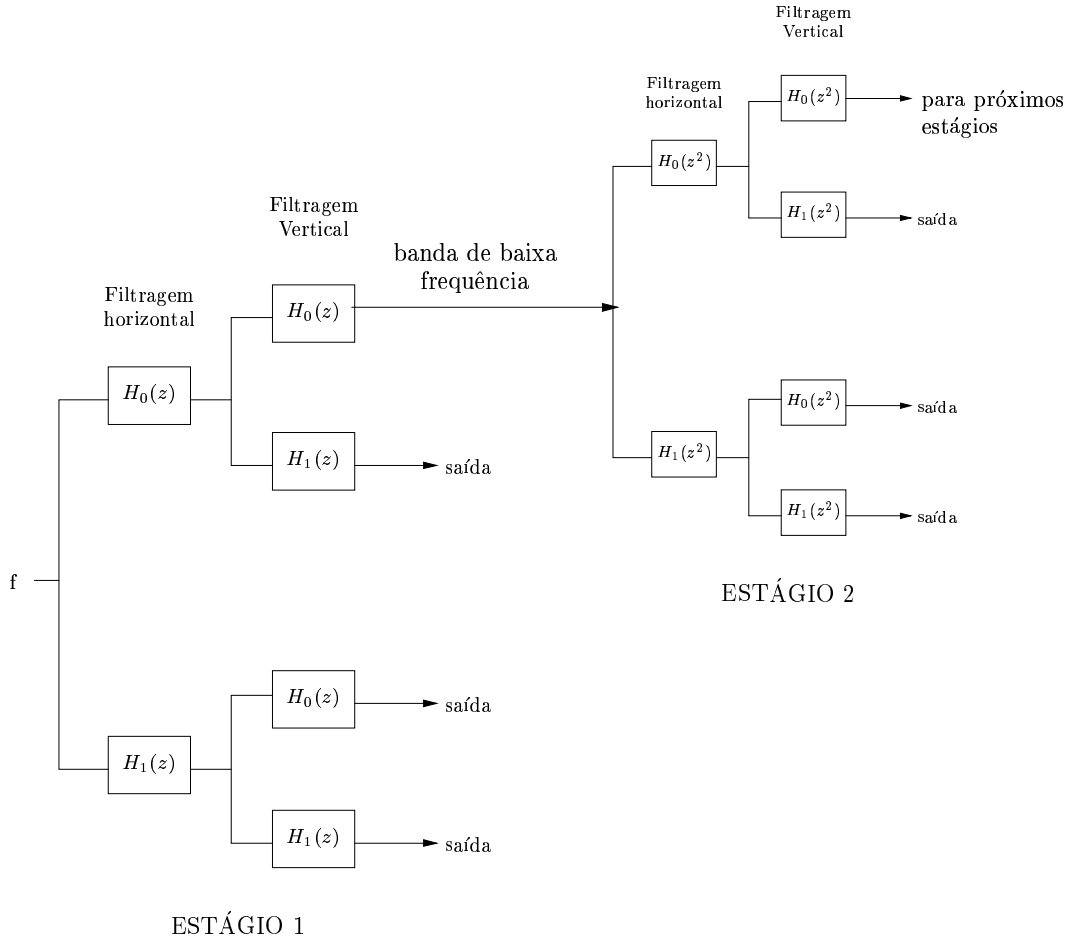

Figura 4.1: Processo de geração da tabela da decomposição.

Uma outra forma de se chegar à tabela da decomposição seria pré-calculando os vetores finais relativos à cada banda, e realizando apenas uma decomposição, ao invés de várias decomposições progressivas, como na figura 4.1. Esses vetores finais seriam calculados simplesmente através da convolução entre os vetores interpolados dos estágios anteriores.

4.6 Cálculo do Número de Estágios Utilizados na Decomposição

Durante o cálculo da decomposição, o tamanho dos vetores utilizados está limitado ao tamanho da imagem original. Além disso, estamos interessados em utilizar vetores com regiões de suporte o maior possível, pois dessa forma a redução na energia do resíduo será a máxima possível, e o algoritmo necessitará de menos passos para convergir. Dessa forma, devemos utilizar uma decomposição com o maior número de estágios possível, sem que o tamanho dos vetores finais ultrapasse as dimensões da imagem. Por exemplo, uma imagem de dimensões 256×256 com vetores $H_0(z)$ e $H_1(z)$ com 2 coeficientes (Base de Haar) necessitaria de uma decomposição com 8 estágios.

4.7 Cálculo da Tabela de Produtos Internos entre as Bases

4.7.1 Tamanho da tabela

Para uma imagem de dimensões $M \times M$ e uma transformada de N estágios, o dicionário apresentará $M \times M \times (3N+1)$ vetores, pois para cada uma das $3N+1$ escalas existem $M \times M$ posições (ou deslocamentos) possíveis, correspondentes a cada pixel da imagem.

Uma tabela que contivesse os produtos internos entre todos esses vetores teria, à rigor, $\frac{M^2(3N+1)\times(M^2(3N+1)+1)}{2}$ entradas, o que torna impraticável não só o cálculo, mas também o armazenamento e o acesso à tal tabela. Entretanto, na prática, várias simplificações podem ser feitas.

Uma vez que a decomposição é separável, os vetores bidimensionais, podem ser separados em duas componentes, horizontal e vertical. Dessa forma, o produto interno entre dois vetores g_i e g_j pode ser desmembrado em

$$\langle g_i, g_j \rangle = \langle g_{i_h}, g_{j_h} \rangle \cdot \langle g_{i_v}, g_{j_v} \rangle, \tag{4.6}$$

onde os índices h e v denotam as componentes horizontais e verticais, respectivamente.

Além disso, se observarmos a figura (4.1), perceberemos que as sete saídas da figura foram geradas através de convoluções que envolviam apenas quatro vetores unidimensionais diferentes ($H_0(z), H_1(z), H_0(z^2)$ e $H_1(z^2)$). Essa idéia pode ser generalizada e chega-se à conclusão de que para se chegar às $3N+1$ escalas bidimensionais diferentes, são necessários apenas $2N$ vetores unidimensionais, calculados a partir dos vetores $H_0(z)$ e $H_1(z)$. Sendo assim, ao invés de $3N+1$ escalas g_i diferentes, podemos ter uma tabela que considere apenas $2N$ vetores unidimensionais diferentes (g_{i_h} e g_{i_v}) como entrada (é claro que, para ambos os casos, os deslocamentos espaciais ainda deverão ser considerados). Essa tabela teria agora $MN(2MN+1)$ entradas, uma redução significativa.

Um outro ponto é que apenas os deslocamentos relativos entre os vetores fazem diferença no cálculo do produto interno, conforme pode ser visto na figura 4.2. Nessa figura, dois vetores com deslocamentos absolutos diferentes (na parte

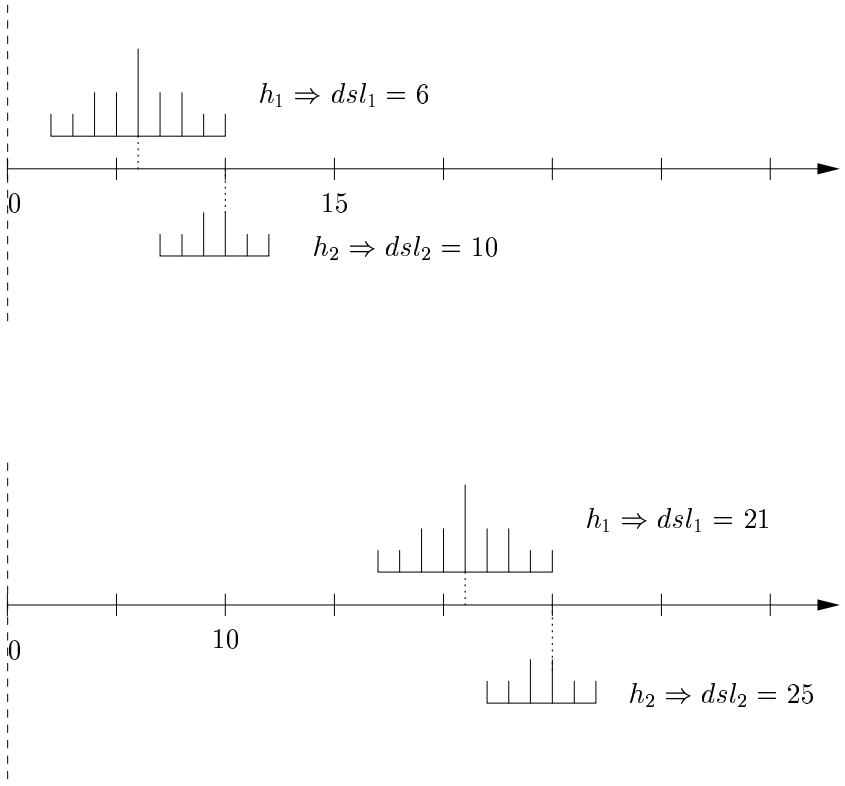

Figura 4.2: Produto interno entre vetores com mesmo deslocamento relativo.

superior $dsl_1 = 6$ e $dsl_2 = 10$, e na inferior $dsl_1 = 21$ e $dsl_2 = 25$) apresentam o mesmo deslocamento relativo ($dsl_2 - dsl_1 = 4$) e, portanto, o mesmo valor para o

produto interno entre eles.

Uma outra consideração é que os vetores $H_0(z^{2^n})$ e $H_1(z^{2^n})$, normalmente, apresentam um número reduzido de coeficientes, se comparado ao tamanho usual das imagens utilizadas. Assim, por exemplo, dois vetores com 10 coeficientes só teriam $10 + 10 - 1 = 19$ deslocamentos relativos tais que seu produto interno fosse diferente de zero, ou seja, que cada vetor apresentasse pelo menos um coeficiente em superposição em relação ao outro.

Para calcular o resultado do produto interno quando algum coeficiente de um vetor atinge a borda da imagem, deve-se levar em consideração o tipo de extensão utilizada. No caso das extensão periódica, o cálculo pode ser feito como em uma convolução circular com período igual ao tamanho da imagem. Dessa forma, casos de borda passam a ser tratados de forma trivial.

4.7.2 Organização da tabela

Uma vez que

$$\langle g_i, g_j \rangle = \langle g_j, g_i \rangle, \tag{4.7}$$

a tabela será organizada de forma que $i \leq j$. A tabela possuirá ainda um cabeçalho contendo informações sobre o número de escalas da transformada utilizada, o número de deslocamentos, e o endereço dos produtos internos entre os vetores do dicionário associados à cada escala dentro da própria tabela. Vale lembrar que uma transformada gerada usando N estágios apresentaré $3N + 1$ escalas bidimensionais diferentes que, conforme explicado na seção 4.7.1, podem ser representadas através de $2N$ vetores unidimensionais diferentes, além de seus deslocamentos espaciais. Um vetor auxiliar, $pos\_esc[\cdot]$, guarda a posição na tabela que indica a posição inicial dos produtos internos dos vetores em cada escala. A figura 4.3 ilustra a estrutura da tabela.

Cada bloco de uma determinada escala i guarda os produtos internos entre os vetores daquela escala e os vetores de todas as outras escalas j tais que $i \leq j$, organizados em duas partes: $dsl_1 \geq dsl_2$ e $dsl_1 < dsl_2$, conforme pode ser visto na figura 4.4. Cada bloco *posição escala i* possui duas entradas, cada uma com a posição de cada um desses dois subblocos.

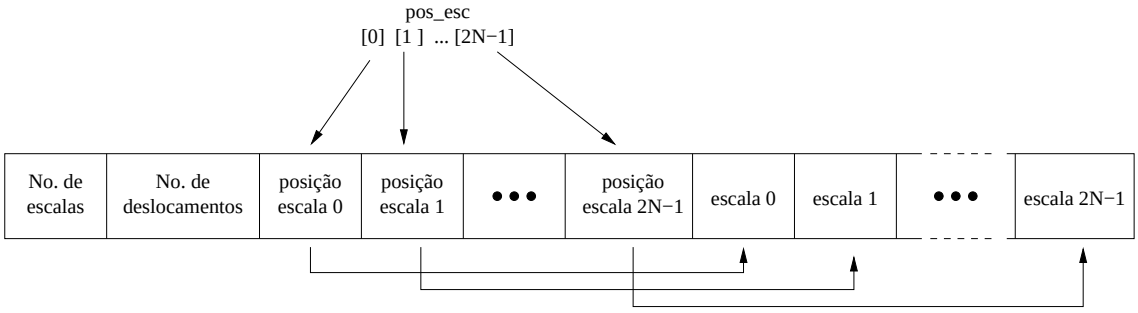

Figura 4.3: Estrutura da tabela de produtos internos entre os vetores do dicionário.

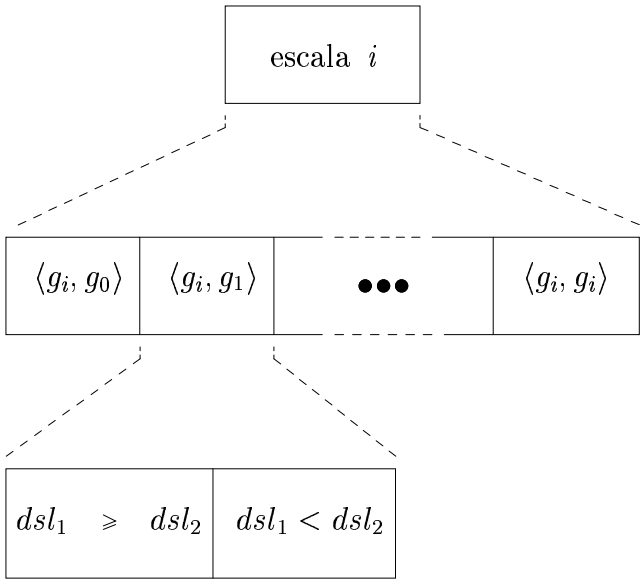

Figura 4.4: Estrutura interna dos blocos das escalas.

O vetor auxiliar $pos\_esc[\cdot]$ visa apenas facilitar a referência ao índice da tabela correspondente ao produto interno procurado.

Sendo assim, a busca do valor de um determinado produto interno pode ser feita quase que diretamente, com os índices relativos ao produto interno procurado facilmente encontrados.

Capítulo 5

Modificações no Algoritmo Proposto

5.1 Limitações no Algoritmo Inicial

Imagine que o maior dos vetores unidimensionais característicos da decomposição utilizada, $H_0(z)$ e $H_1(z)$, tenha M coeficientes, com M tipicamente pequeno (em torno de 10). Os vetores bidimensionais relativos ao estágio 0 na figura 4.1 teriam então uma região de suporte de, no máximo, $M \times M$. Entretanto, pode-se observar pela figura que o tamanho dos vetores $H_0(z^{2^n})$ nos estágios subseqüentes é dado por $M_n = M(2^n + 1) - 1$ (tamanho dos vetores interpolados), onde n é o número do estágio em questão. Assim, podemos dizer, a grosso modo, que o tamanho dos vetores será proporcional a $M2^n$. Assim, as regiões de suporte dos vetores bidimensionais seguiriam aproximadamente os padrões ilustrados na figura 5.1.

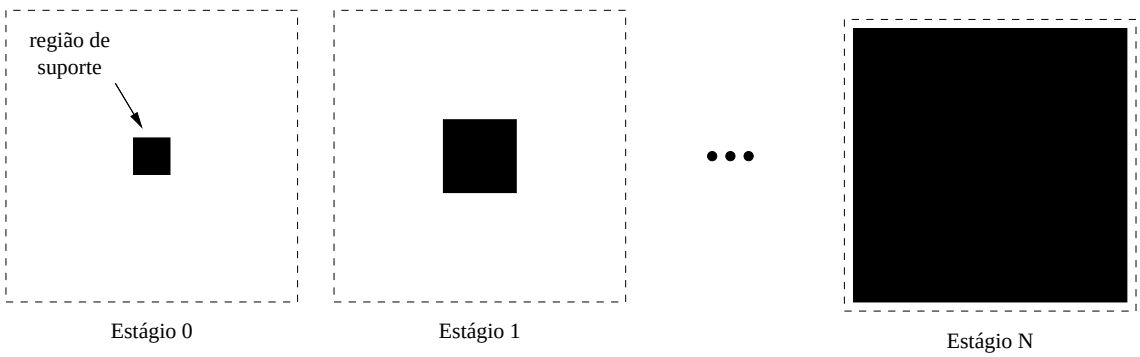

Figura 5.1: Padrões das regiões de suporte dos vetores bidimensionais.

Assim, concluimos que vetores geradas através de estruturas de divisões por oitavas possuem regiões de suporte bidimensionais aproximadamente proporcionais a $M2^n \times M2^n$.

Suponha agora que, durante a implementação do algoritmo proposto através da equação (4.5), um resíduo num passo n apresente um dos padrões da figura 5.2:

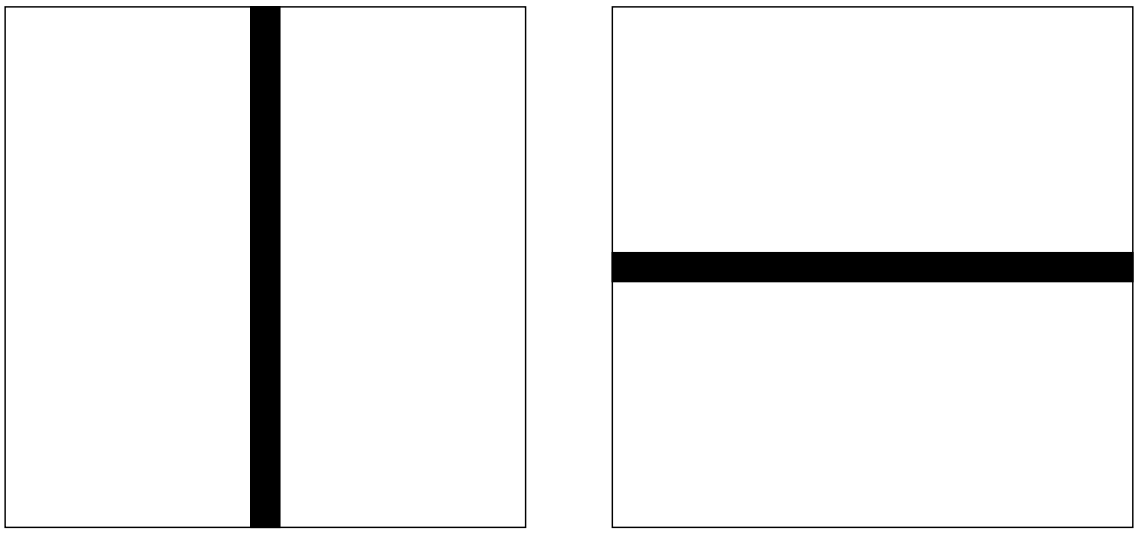

Figura 5.2: Exemplo de resíduos

Nesse caso, o algoritmo somente será capaz de compor o resíduo através da escolha de vários vetores com região de suporte pequena, como o da figura 5.3, deslocados a cada passo, pois não há vetores com regiões de suporte semelhantes aos resíduos apresentados.

O ângulo θ entre o vetor escolhido $\vec{v}$ e o resíduo $\vec{r}$ em questão é dado por

$$cos\theta = \frac{\|\vec{r}.\vec{v}\|}{\|\vec{r}\|.\|\vec{v}\|} \tag{5.1}$$

Uma vez que os vetores do dicionário utilizado são normalizados, ou seja, $\|\vec{v}\| = 1$, uma pequena região de suporte faz com que o numerador da equação (5.1) seja pequeno, e o ângulo, por consequência, tenda a crescer. Assim, a condição de $\Theta(C_N) \leq 60^o$ estaria deixando de ser cumprida e o algoritmo não convergiria.

Durante testes práticos, casos como esse se mostraram muito incidentes, fazendo com que o algoritmo apresentasse problemas logo nas primeiras iterações.

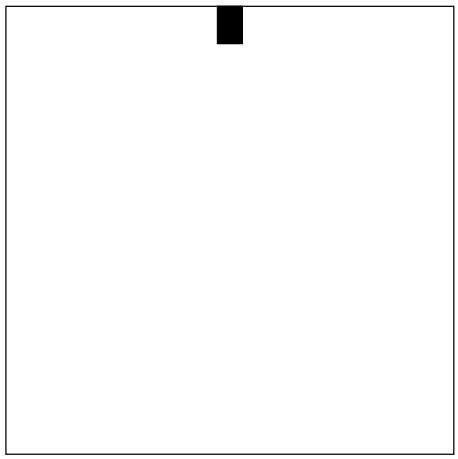

Figura 5.3: Vetor com região de suporte pequena

Algumas modificações teriam então que ser feitas para que esses problemas fossem superados.

5.2 Solução Inicial (e não suficiente)

A geração do dicionário, conforme explicado no apêndice A, é feita a partir dos vetores $H_0(z)$ e $H_1(z)$, através de divisão por oitavas, como ilustra a figura 5.4.

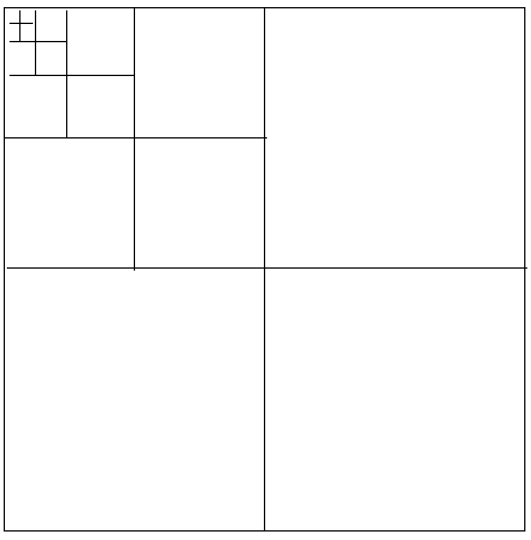

Figura 5.4: Divisão por oitavas

Se o conjunto dicionário passasse a ser gerado por divisão em múltiplas escalas

teríamos a composição das imagens na figura 5.5:

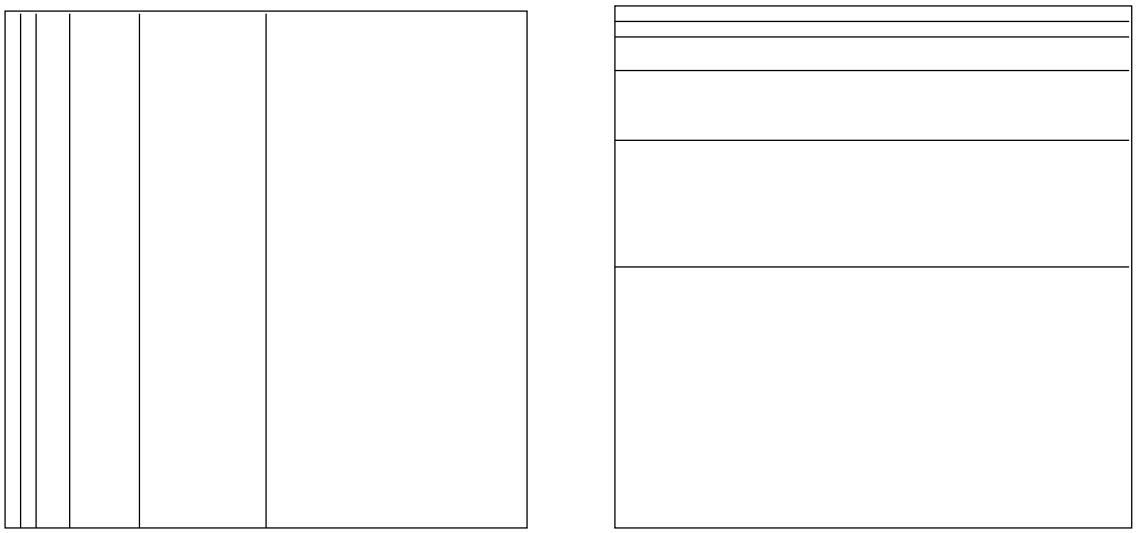

Figura 5.5: Divisão por múltiplas escalas

Agora teríamos vetores que resolveriam o problema da figura 5.2. Entretanto, essa solução seria ainda insuficiente para o caso geral. Casos como o da figura 5.6 ainda não seriam resolvidos por não serem separáveis e, portanto, não poderem ser compostos com vetores provenientes da divisão por múltiplas escalas.

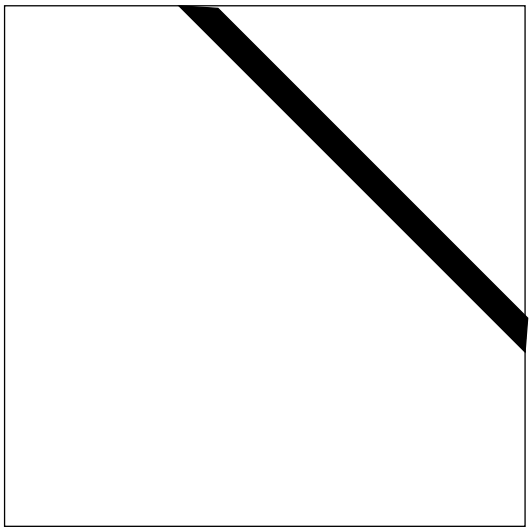

Figura 5.6: Resíduo diagonal

5.3 Solução

Percebeu-se que dicionários gerados a partir de divisões por oitavas, como a decomposição que está sendo utilizada, seriam sempre deficientes na representação de resíduos como os da figura 5.2.

Para que os problemas descritos nas seções anteriores sejam solucionados, deve-se garantir que, a cada passo, o ângulo entre o vetor encontrado e o resíduo não ultrapasse 60^o, mesmo nos casos em que $\Theta(C_N) > 60^o$. Para isso, uma vez que as normas dos vetores do dicionário são sempre unitárias, independente do tamanho de sua região de suporte, só nos resta tentar diminuir a norma do resíduo em questão. Uma forma de fazer com que esse valor diminua é fazer com que a região de suporte do resíduo diminua.

Assim, cada vez que o ângulo entre o vetor escolhido e o resíduo fosse maior que 60^o, a imagem seria dividida em quatro blocos e o algoritmo continuaria recursivamente, e cada bloco seria visto como uma nova imagem a ser codificada. Com isso, forçaríamos $\|\vec{r}\|$ a diminuir, aumentando assim o valor de $cos\theta$ e garantindo a convergência.

A decomposição dos resíduos correspondentes a cada um dos quatro blocos deverá ser calculada, e quatro algoritmos (um relativo à cada bloco) passarão a ser executados paralelamente. Assim, passaremos a ter, para cada passo, quatro vetores. No limite, o chegaremos a blocos de tamanho 2×2, que podem ser codificados sem distorção com relativa facilidade através de bases de Haar [7].

É importante observar que o algoritmo pode utilizar dois dicionários diferentes na codificação de uma mesma imagem. Pode-se, por exemplo, fazer com que um segundo dicionário seja utilizado quando o tamanho dos blocos for menor que um valor pré-estabelecido. Assim, vetores mais adequados a tamanhos diferentes podem ser utilizados, fazendo com que o desempenho do algoritmo melhore como um todo.

5.4 Modificações na Decomposição da Imagem

A tabela $\langle f, \vec{v} \rangle$, que corresponde à decomposição da imagem f no dicionário $\{v_k\}$, isto é, à projeção da imagem em cada um dos vetores desse dicionário, consiste, na prática, em uma sequência de $3N+1$ imagens, onde N corresponde ao número de

estágios utilizados na decomposição em questão. Cada imagem está associada a uma escala, com seu vetor característico e seus deslocamentos horizontais e verticais, e cada pixel de cada uma dessas imagens corresponde ao resultado do produto interno entre a imagem original e o vetor bidimensional com o deslocamento associado à posição do pixel em questão. A figura 5.7 ilustra a localização do produto interno entre o vetor g_0 com $dsl_h = x$ e $dsl_v = y$ e a imagem f na tabela $\langle f, \vec{v} \rangle$. O vetor g_0 estaria associada à escala 0, e, portanto, à primeira imagem. Sendo assim, o pixel (x, y) desta imagem indicaria o valor do produto interno procurado.

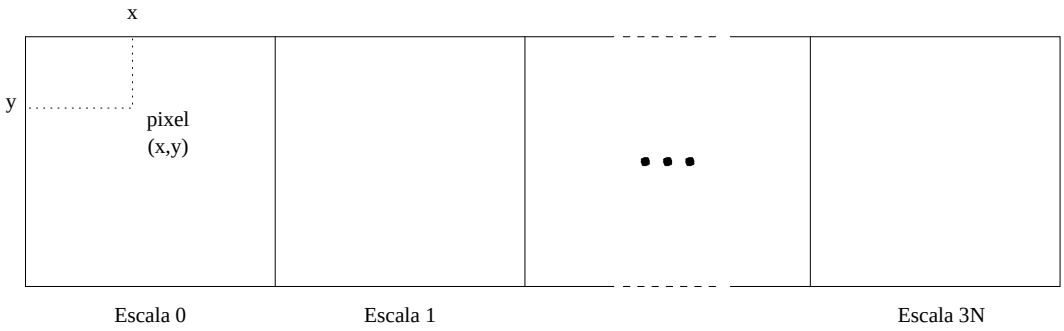

Figura 5.7: Localização do produto interno entre um vetor e a imagem original na tabela $\langle f, \vec{v} \rangle$

Conforme dito anteriormente, após a divisão da imagem resíduo em quatro novos blocos, a decomposição de cada um desses blocos no dicionário deverá ser calculada. Entretanto, uma vez que as dimensões dos novos blocos serão metade das do bloco que foi dividido, os vetores de tamanho superior à essas dimensões deixarão de ser utilizados no cálculo da decomposição dos sub blocos. Na prática, a decomposição utilizada para o cálculo dos produtos internos em questão deverá utilizar $N - 1$ estágios, onde N é o número de estágios utilizados pela transformada do bloco que foi dividido.

Isso implica que, em relação ao bloco original, as decomposições dos blocos menores possuirão 3 vetores a menos (e seus deslocamentos espaciais), ou seja, a nova tabela $\langle f, \vec{v}' \rangle$ terá três imagens a menos. A figura 5.8 mostra a estrutura da tabela $\langle f, \vec{v} \rangle$ após a primeira divisão na imagem.

Imagine agora que o bloco 1 na figura 5.8 deva ser dividido. Seguindo o mesmo raciocínio, a nova tabela ficaria como na figura 5.9.

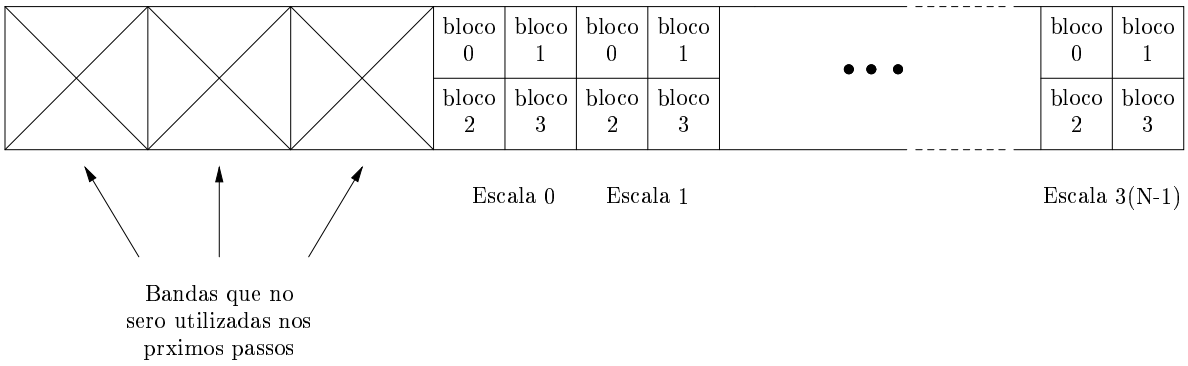

Figura 5.8: Estrutura da tabela $\langle f, \vec{v} \rangle$ após a primeira divisão na imagem.

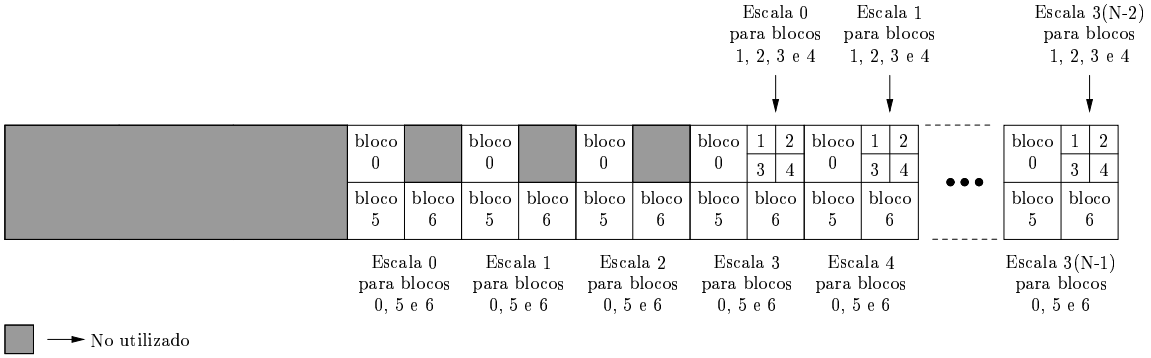

Figura 5.9: Estrutura da tabela $\langle f, \vec{v} \rangle$ após divisão do bloco 1.

5.5 Modificações na Tabela de Produtos Internos

Uma vez que os vetores utilizados após a divisão de um bloco são os mesmos de antes (exceto pelo fato de que talvez os de mais baixa frequência tenham deixado de ser utilizados), não há a necessidade de realizar nenhuma modificação na tabela de produtos internos. Sendo assim, a estrutura detalhada na seção 4.7 continua a ser utilizada da mesma forma.

5.6 Varredura dos Blocos

Uma importante questão a ser considerada agora é a definição da ordem de varredura dos blocos. Imagine que o algoritmo divida a imagem num determinado passo. Teríamos então quatro resíduos organizados conforme a figura 5.10.

Se não houvesse um critério de varredura definido, o algoritmo poderia prosseguir de forma que somente vetores relativos ao bloco 0 fossem transmitidos, e

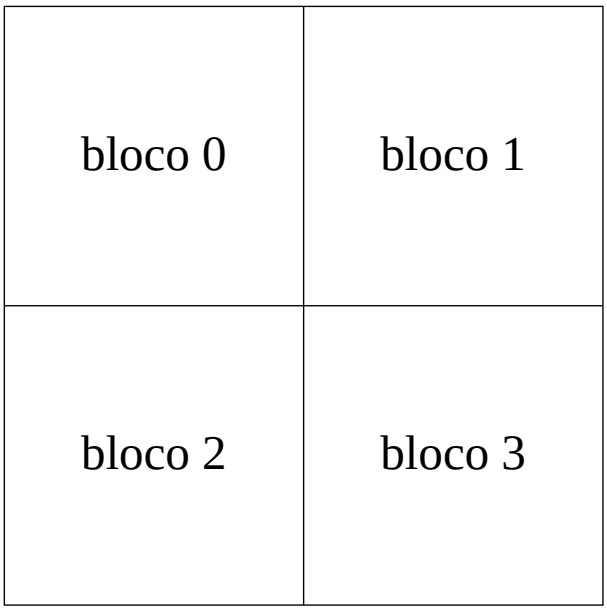

Figura 5.10: Imagem após a primeira divisão em blocos

diferentes regiões da imagem poderiam ser aproximadas com níveis de distorção diferentes. De fato, dado um determinado nível de distorção para a imagem aproximada, poderiam ser transmitidos somente vetores relativos à um bloco, resultando numa excelente distorção para uma região, enquanto outras regiões nem chegariam a ser aproximadas.

Sendo assim, é necessário que a transmissão de vetores obedeça a uma ordem pré-estabelecida, de forma que nenhum dos blocos seja priorizado em relação a outro. A ordem definida obedece ao princípio de numeração dos blocos como na figura 5.10. Dessa forma, se o bloco 1 nessa figura sofresse uma divisão, os blocos resultantes seriam ordenados conforme a figura 5.11. Essa divisão geraria a tabela $\langle f; \vec{v} \rangle$ com estrutura semelhante à da figura 5.9.

Assim, o algoritmo deve então transmitir um vetor relativo a cada bloco, seguindo essa ordenação, para só então voltar ao bloco 0. Dessa forma, a cada passagem, fazemos com que todos os blocos sejam codificados, sem que nenhum deles seja priorizado.

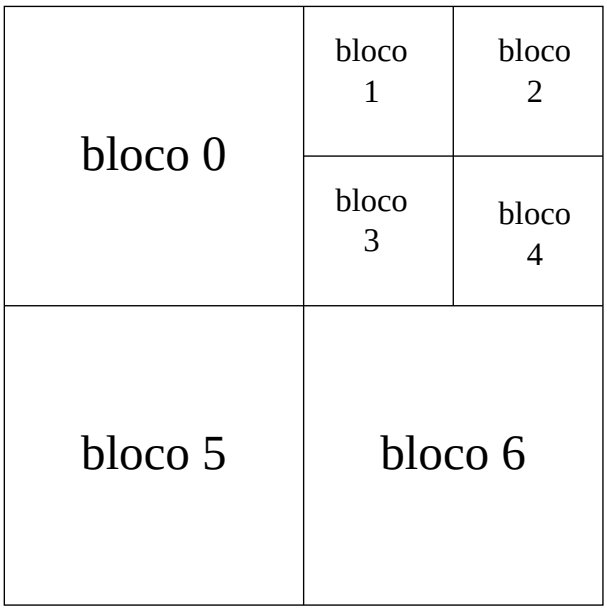

Figura 5.11: Imagem após a divisão do bloco 1

5.7 Critério de Decisão para Incremento do Expoente de α

Antes das modificações propostas neste capítulo, o algoritmo de aproximações sucessivas proposto utilizava como critério de decisão para o incremento do expoente de α a norma do resíduo no passo n, comparada à norma da imagem original (X_{max}). Isso pode ser visto nos passos 5 e 6 do algoritmo 2.2, aqui reproduzidos:

5. Se $\|\mathbf{w}\| > \alpha^n X_{max}$, faça $l = l + 1$ e volte ao passo 3.

6. Incrementar n de 1 e faça $l = 1$.

Entretanto, com a divisão do algoritmo em blocos independentes, parece bastante provável que os resíduos relativos a cada bloco reduzam suas normas de forma, a princípio, aleatória. Sendo assim, a existência de um expoente global, que estaria relacionado ao nível de distorção total, se tornaria impossível, pois cada bloco poderia estar em um nível de distorção (expoente) diferente.

Como desejamos garantir que a imagem como um todo apresente o mesmo nível de distorção, devemos garantir que todos os blocos estejam aproximadamente no mesmo nível de distorção, ou seja, no mesmo expoente. Assim, foi definido

um flag *hold* que diz ao codificador que um determinado bloco atingiu o nível de distorção para que seu expoente seja incrementado, e que ele deve então deixar de ser codificado até que todos os outros blocos atinjam o mesmo nível. Então, quando todos os blocos atingirem o mesmo nível de distorção, o expoente global será incrementado e todos os blocos continuarão a ser codificados utilizando esse novo expoente.

5.8 Nível de Distorção a Cada Passo

Mais tarde, após algumas simulações, pecebeu-se que garantir que todos os blocos estavam sendo codificados com o mesmo expoente não garante que todos estejam no mesmo nível de distorção. É fácil perceber isso quando se leva o tamanho dos blocos em consideração. Blocos menores possuem normas menores e a comparação com $\alpha^n X_{max}$ tenderia a fazer com que o expoente fosse incrementado em momentos errados, uma vez que o valor $\alpha^n X_{max}$ supõe a comparação com toda a imagem. Assim, deve-se realizar algum tipo de modificação que leve em conta as dimensões do bloco na decisão do incremento do expoente.

A modificação implementada consiste em permitir que blocos menores sejam codificados com expoentes maiores do que o expoente global atual. A decisão de quão maior serão esses expoentes leva em conta o tamanho relativo entre o bloco em questão e o maior bloco no momento. Seja n o expoente global atual, e sejam l_0 e l o tamanho do maior bloco no momento e o tamanho do bloco a ser codificado, respectivamente. Então o bloco poderá ser codificado com o expoente $n + m$, desde que m satisfaça a inequação

$$2^{2l}\alpha^{n+m} < 2^{2l_0}\alpha^n. \tag{5.2}$$

Teremos agora um vetor auxiliar que guarda o valor de m para cada bloco da imagem.

Capítulo 6

Organização e Decodificação da Sequência de Saída

6.1 Considerações Iniciais

Uma vez que a sequência de vetores $\{v_{k_i}\}$, $k_i = 0, 1, \ldots, Q-1$, onde Q é o número de vetores utilizados, foi gerada pelo codificador, a reconstrução da imagem de saída será feita simplesmente através da soma dos vetores dessa sequência, conforme a equação (6.1):

$$f = \sum_{k_i=0}^{Q-1} X_{max} \alpha^{n_{k_i}} v_{k_i}. \tag{6.1}$$

Cada vetor v_{k_i} do dicionário pode ser expresso pela sua escala correspondente com seus respectivos deslocamentos horizontal e vertical. Além disso, o expoente n_{k_i} em que o vetor foi codificado deve também ser informado. A figura 6.1 ilustra a composição da imagem a partir dos vetores escolhidos durante a codificação.

Assim, a imagem é reconstruída através de uma simples soma algébrica. Isso implica em que a ordem em que os vetores são somados não altera a imagem final. Entretanto, é importante notar que o algoritmo de codificação busca progressivamente o vetor mais próximo ao resíduo, fazendo com que a sequência de vetores gerada esteja ordenada do vetor mais significativo para o menos significativo.

Um outro fato percebido é que, como diferentes tamanhos de bloco estão sendo utilizados em níveis diferentes do algoritmo, um mesmo vetor poderá ter que ser somado de forma diferente. Uma vez que a extensão utilizada é periódica,

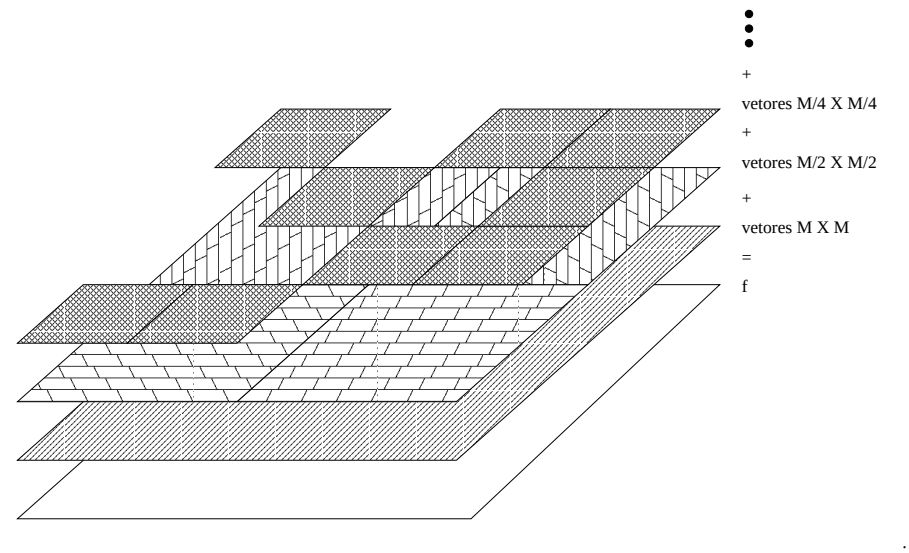

Figura 6.1: Reconstrução da imagem

dependendo do tamanho do bloco em questão, um determinado vetor pode ou não necessitar ser extendido. Na figura 6.2 o mesmo vetor com os mesmos deslocamentos horizontal e vertical é visto em situações com tamanhos de bloco diferentes.

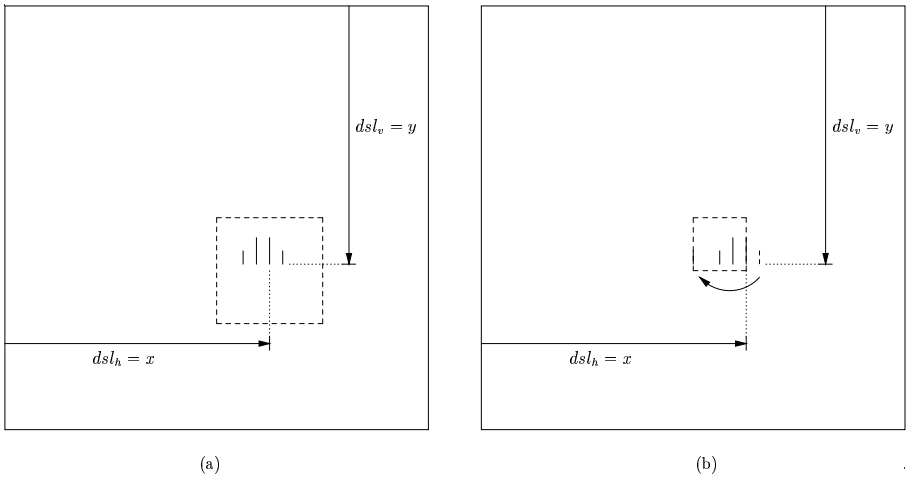

Figura 6.2: (a) A região de suporte do vetor se encontra dentro das dimensões do bloco: não é necessário fazer extensão; (b) A região de suporte está fora do bloco: o último coeficiente do vetor é extendido periodicamente para a posição à esquerda do bloco.

Sendo assim, além da escala, devemos informar ao decodificador também os deslocamentos horizontais e verticais, o expoente e o tamanho do bloco correspon-

dente à cada vetor. Isso leva a um número muito grande de informações a serem transmitidas, tornando os resultados práticos muito pouco eficientes. Nas próximas seções serão propostas formas de se organizar os vetores de saída de forma que os problemas aqui apresentados sejam eliminados ou reduzidos.

6.2 Localização os Vetores Através de Árvore Quaternária

Se considerarmos todas as escalas com todos os deslocamentos possíveis, teríamos um número grande demais de vetores a ser considerado (ver seção 4.7.1). A transmissão dos deslocamentos para os vetores, ou seja, da localização de cada vetor dentro da imagem, utilizaria da mesma forma um número muito grande de bits.

Uma forma de localizar todos os vetores utilizados na imagem é utilizando o conceito de árvore. O algoritmo para localização de pixels através de uma árvore é bastante conhecido [8]. A idéia consiste em dividir a imagem em quatro blocos e transmitir bits indicando se existem ou não vetores com a origem naquele bloco. Em seguida, cada bloco dentro dos quais existam vetores são novamente divididos e os bits relativos a cada sub bloco transmitidos. O algoritmo prossegue até que se chegue a blocos 1×1, que corresponde à localização exata dos vetores. Como exemplo, considere a imagem de dimensões 8×8 da figura 6.3 com os três pixels assinalados.

O processo para a geração da sequência de bits para a localização dos pixels na imagem da figura 6.3 está ilustrado na figura 6.4.

A partir da transmissão dos bits da árvore, poderia-se então iniciar a transmissão dos vetores, seguindo a ordem das posições dadas pelas folhas da árvore. Dessa forma o decodificador já teria a informação sobre a sua localização. A estrutura em árvore necessita de muito menos bits do que a transmissão do deslocamento de cada vetor, tornando a organização da sequência de saída mais eficiente.

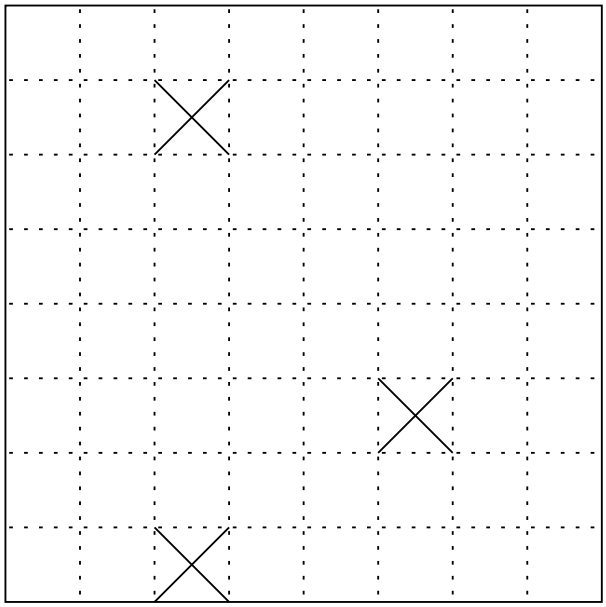

Figura 6.3: Imagem 8×8 com três pixels a serem localizados através da árvore.

6.3 Limitando o Número de Escalas e Deslocamentos por Bloco

Pelo explicado na seção 5.1, vetores com regiões de suporte relativamente menores do que o tamanho do bloco para o qual estão sendo codificadas são muito pouco prováveis de serem utilizados. Sendo assim, na prática, um número menor do que $3N + 1$ escalas tende a ser utilizado. A fim de otimizar a codificação, podemos então forçar o algoritmo de codificação a fazer a busca pela melhor escala num subconjunto do conjunto de escalas original. Isso diminuiria ainda mais o número de bits necessários à transmissão das escalas utilizadas.

Para vetores com grande região de suporte, deslocamentos pequenos teriam pouca influência no produto interno entre o vetor e o resíduo. Seguindo essa idéia, pode-se tentar também reduzir o número de deslocamentos possíveis, o que tornaria possível a otimização do algoritmo da árvore. Essa modificação é explicada na seção a seguir.

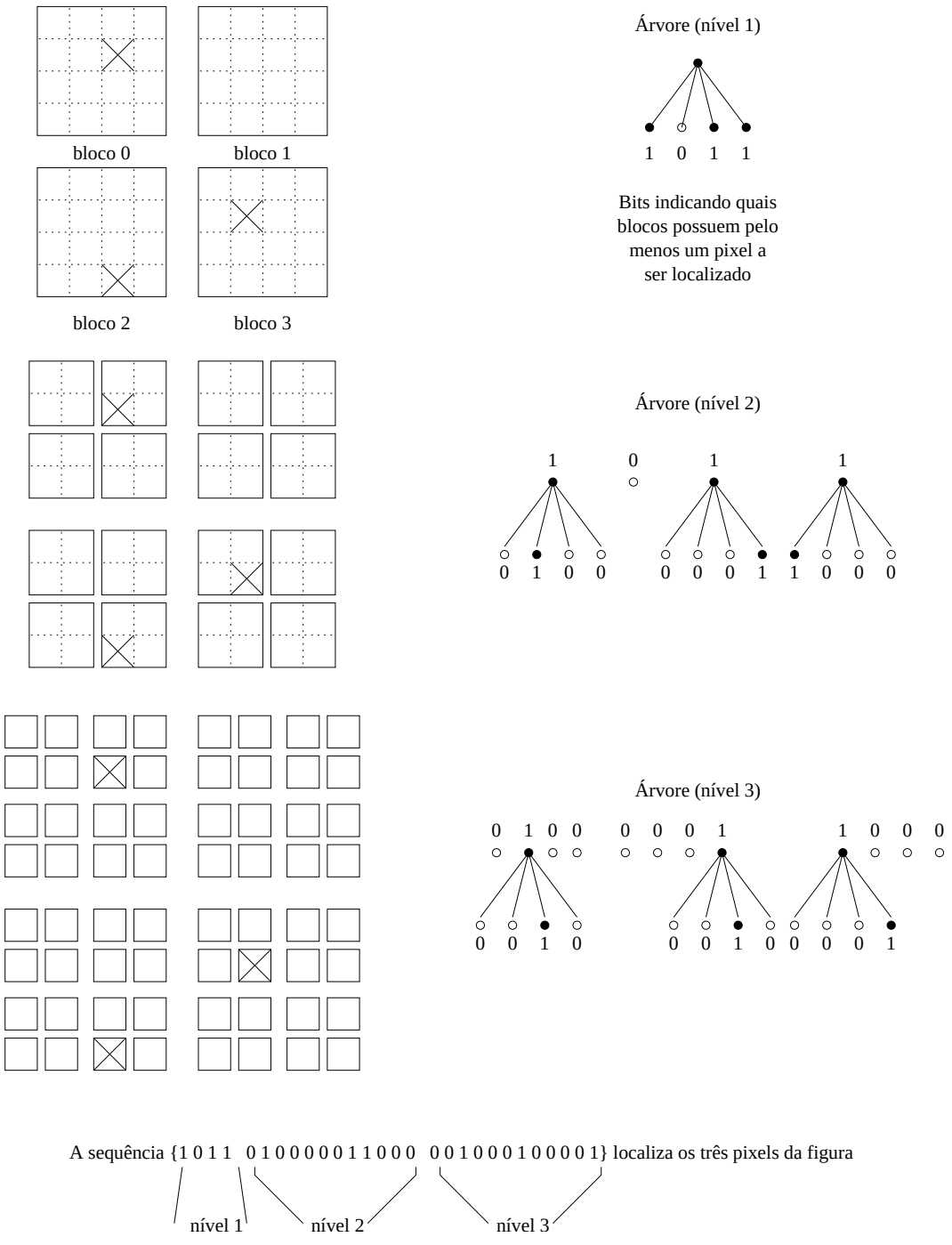

Figura 6.4: Localização dos pixels da figura 6.3 através da árvore.

6.4 Modificações na Geração da Sequência de Saída

Conforme pode ser visto na figura 6.4, cada nível da árvore está associado a um tamanho de bloco. Pode-se usar essa informação para evitar a necessidade da transmissão do tamanho das bases codificadas. Isso pode ser feito transmitindo-se os vetores relativos à cada tamanho de bloco logo após a transmissão dos bits do nível da árvore correspondente, o que ainda acarreta na diminuição do tamanho da árvore, pois os vetores já transmitidos deixam de ser considerados nos próximos níveis da árvore. Um flag *proximo_bloco* indicaria que vetores do próximo bloco passariam a ser transmitidos.

Se, além disso, houver um número reduzido de deslocamentos possíveis, essa sequência de vetores relativos à um tamanho de bloco pode ser transmitida seguindo a ordem de deslocamentos, sendo separados por outro flag, *proximo_deslocamento*. Um exemplo de blocos com quatro deslocamentos possíveis pode ser visto na figura 6.5. Nesse exemplo, seriam transmitidas inicialmente os vetores cujo deslocamento coincidice com a posição "0" na figura, em seguida com a posição "1", e assim sucessivamente.

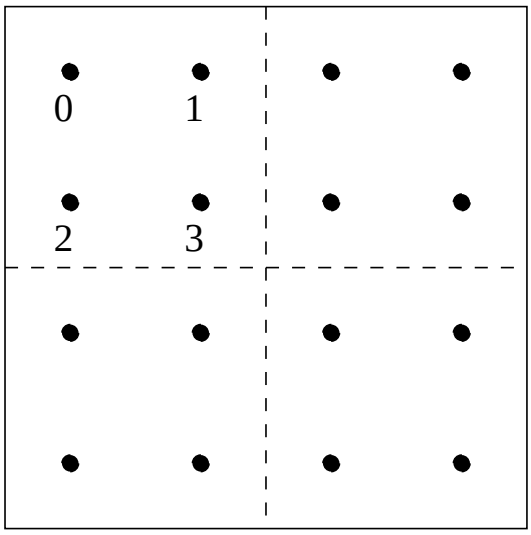

Figura 6.5: Bloco com número de deslocamentos reduzido.

Para reduzir ainda mais o número de escalas possíveis, foi criado o flag *sinal*. Na seção 3.2 vimos que o conjunto dicionário utilizado na codificação foi definido como $\phi_n(t) = \{\psi_n(t) \cup -\psi_n(t)\}$, o que dobra o número de vetores da base original

$\psi_n(t)$. Assim, os vetores transmitidos seriam organizados em dois blocos, separados pelo flag *sinal*, correspondentes aos vetores positivos e negativos.

Um último flag, *expoente*, indica que o expoente dos vetores transmitidos, organizados também em ordem crescente de expoente, foi incrementado.

A figura 6.6 mostra a organização da sequência de vetores transmitida pelo codificador.

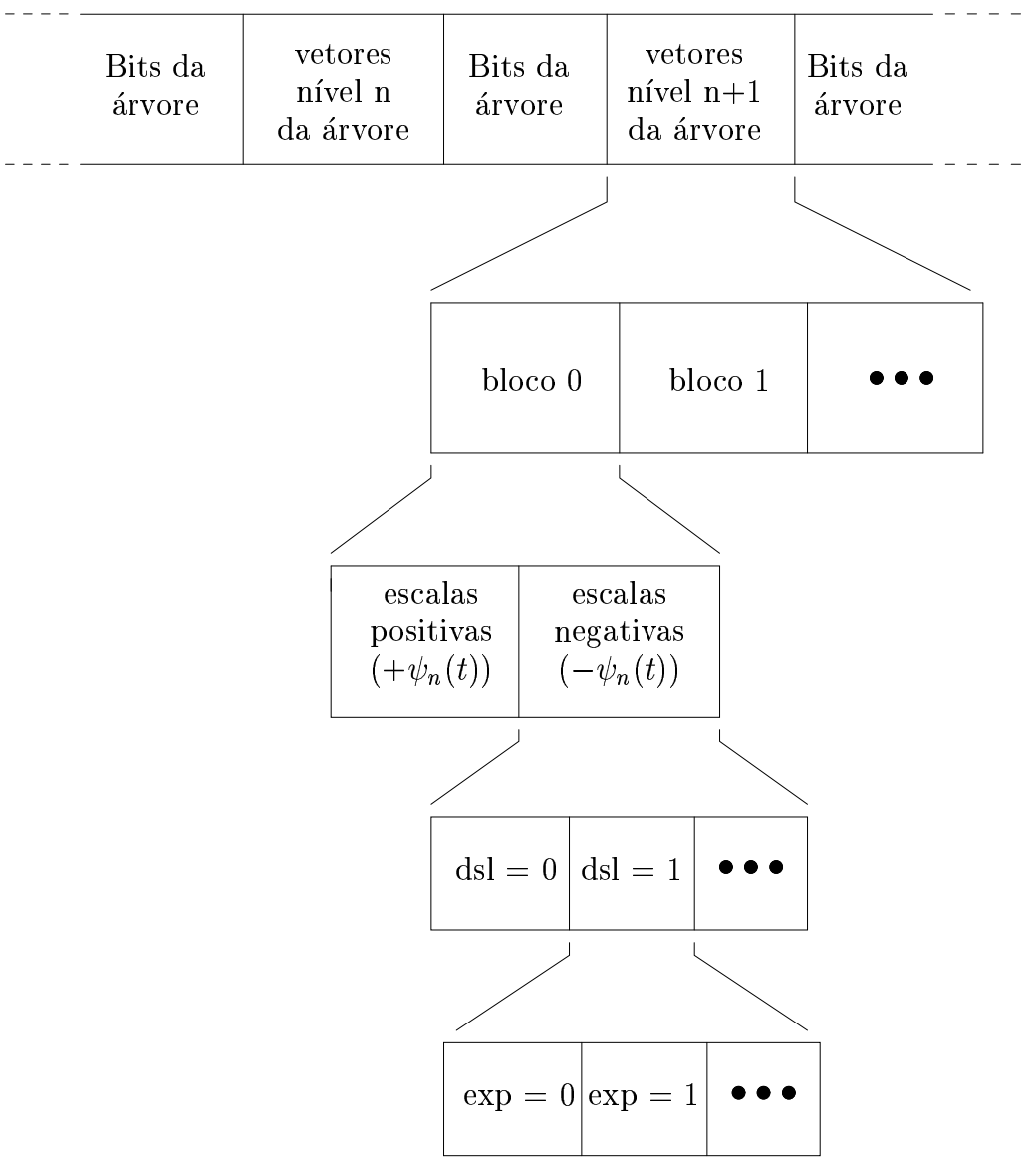

Figura 6.6: Organização da sequência de saída.

6.5 Codificação Aritmética

Para aumentar a compressão da sequência de saída, foi utilizado um codificador aritmético [9]. Inicialmente, foi usado um codificador com somente um modelo, atribuindo um símbolo para cada escala, flag e símbolo da árvore (0's e 1's). Em seguida, para melhorar ainda mais o resultado, utilizou-se um codificador aritmético com dois modelos, um para as escalas e flags e outro binário para os bits da árvore. Uma comparação quantitativa entre os dois codificadores será feita no capítulo 7.

Capítulo 7

Resultados

Os vetores da decomposição utilizados nas simulações a seguir foram baseados nas bases de *Haar*. Elas possuem $H_0(z) = \{1, 1\}$ e $H_1(z) = \{-1, 1\}$ (ver apêndice A). Essa decomposição certamente não é a mais adequada para a representação de sinais, sendo pouco utilizada na literatura para este fim. Entretanto, decomposições com bases maiores limitariam o tamanho mínimo dos bloco às dimensões dessas bases. Uma base da wavelet 9/7 (cujas bases posuem 9 e 7 elementos), por exemplo, só poderia ser usada para blocos de tamanho maior ou igual a 16, e outro sistema de bases (como a Haar) teria que ser usado para os blocos menores. Assim, escolheu-se as bases de Haar pelo fato delas cobrirem todos os tamanhos de blocos até 2×2, o que garante a convergência do algoritmo. Outras bases, que além de cobrirem os casos de blocos 2×2, apresentem um maior número de diferentes escalas por bloco, o que pode indicar um menor valor para $\Theta(C_N)$, poderiam ser utilizadas, o que tenderia a melhorar os resultados.

Os valores da distorção entre as imagens originais e as codificadas serão feitas utilizando a medida *PSNR* (ver apêndice B).

7.1 Organização dos Vetores de Saída por Significância

Devido às modificações propostas nas seções 5.7 e 5.8, a sequência de vetores de saída deixou de ser organizada por expoentes crescentes. Uma vez que expoentes menores estão associados à vetores de maior energia, é natural presumir que se uma

sequência S_1 está organizada na forma crescente de expoentes e uma outra sequência S_2, com o mesmo número de vetores de S_1, não está, então a decodificação de S_1 levará a uma menor distorção que S_2.

Sendo assim, imagine que o algoritmo gere uma saída composta de uma sequência S com Q vetores. Agora imagine que desejamos compor duas sequências S_1 e S_2, a partir de S, com Q' vetores, $Q' < Q$. A sequência S_1 é gerada simplesmente pegando os Q' primeiros vetores de S. S_2 é gerada fazendo uma busca em S primeiramente por todos os seus vetores com expoente 1, em seguida por todos com expoente 2, e assim sucessivamente, até que tenham sido encontrados Q' vetores.

Estamos interessados aqui em mostrar que podemos diminuir a distorção para um mesmo número de vetores utilizados na decodificação gerando uma sequência maior do que a desejada, reorganizando-a por ordem crescente de expoente, e utilizando apenas o número de vetores inicialmente desejado. Ainda não serão feitas comparações levando-se em conta o número de bits/pixel atingido, mas apenas o número de vetores utilizados. Taxas expressas em bits/pixel serão mais coerentes somente após a utilização da árvore para localização dos vetores, e da utilização de codificadores aritméticos.

Sendo assim, a imagem de teste *lena* de dimensões 512×512 foi codificada de forma a gerar um sequência de saída com aproximadamente 100.000 vetores. Foram geradas imagens a partir da decodificação das sequências S_1 (não organizadas por expoente) e S_2 (organizadas por expoente). A curva da figura 7.1 mostra que a simples reorganização das sequências de saída pode levar a ganhos de até 1dB.

A título de ilustração, as figuras 7.2 e 7.3 mostram as imagens lena decodificadas para 30.000 vetores com as sequências S_1 e S_2.

7.2 Redução no Número de Deslocamentos e Escalas Possíveis

Nesta seção serão mostrados os resultados que motivaram a redução no número de escalas, conforme proposto na seção 6.3.

A tabela 7.1 mostra a distribuição das escalas na sequência de vetores na codificação da imagem lena 512×512, utilizando cerca de 100.000 vetores e 9 estágios

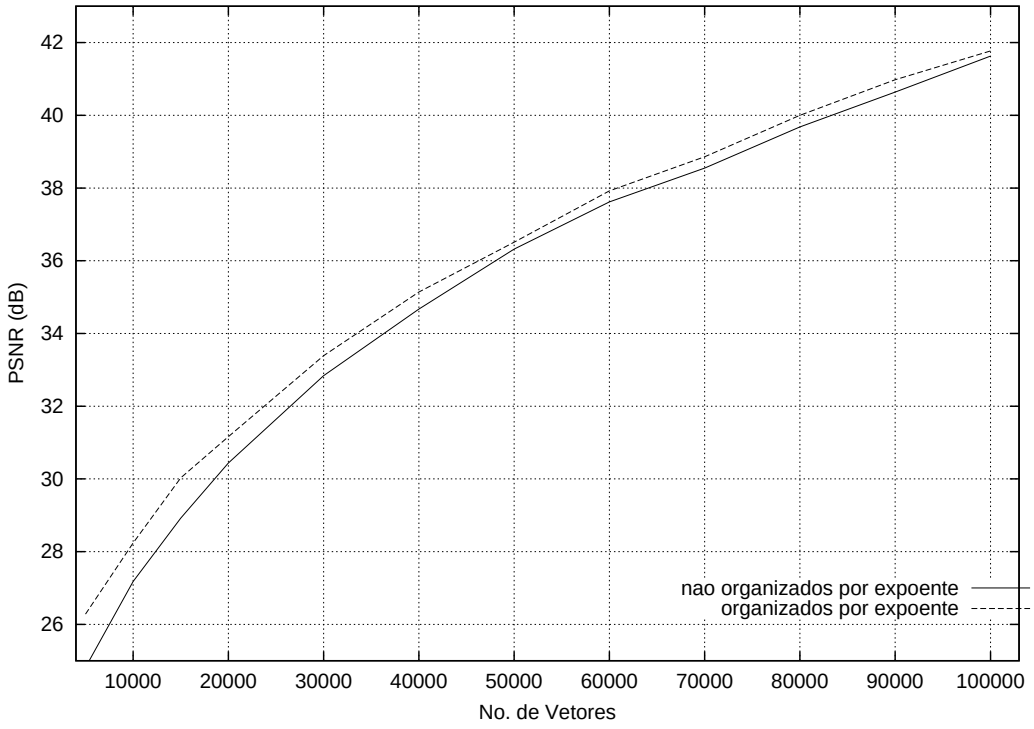

Figura 7.1: Comparação entre a decodificação de sequências organizadas por expoente e sequências não organizadas por expoente para a imagem Lena 512×512.

na decomposição.

Assim, percebe-se que a busca dos vetores durante a codificação pode ser simplificada limitando-se o número de escalas consideradas, uma vez que a grande maioria delas não é utilizada. No caso de uma redução para um número de vetores inferior ao que foi efetivamente utilizado, o algoritmo deverá eventualmente substituir um vetor anteriormente escolhido, mas que agora não faz mais parte do dicionário. Isso, a princípio, pode reduzir a eficiência do algoritmo, pois o vetor escolhido não é mais aquele que apresenta o menor ângulo com o resíduo. Entretanto, a redução no número de bits gastos por escala melhora a taxa, resultando num melhor valor de distorção em função da taxa.

Porém, o algoritmo de aproximações sucessivas é um algoritmo *greedy*. Isto significa que, apesar de garantir que o menor erro é obtido a cada passo, não necessariamente o resultado final será o melhor [10]. Sendo assim, é possível que mesmo com a redução do número de escalas, ainda tenhamos resultados melhores em termos absolutos.

Figura 7.2: Lena 512×512 decodificada com 30.000 vetores da sequência S_1 (não organizada por expoente).

Figura 7.3: Lena 512×512 decodificada com 30.000 vetores da sequência S_2 (organizada por expoente).

Tabela 7.1: Distribuição de escalas na sequência de vetores utilizados para codificação da imagem Lena 512×512

escala	n^o de ocorrências	% do total
0	22898	21.38%
1	16978	15.85%
2	12562	11.72%
3	9082	8.48%
4	22804	21.28%
5	15820	14.76%
6	6783	6.34%
7	119	0.12%
8	68	0.07%
9	4	$\sim 0\%$
10	6	~ 0 %
11	0	0 %
12	0	0 %
13	0	0 %
14	0	0 %
15	0	0 %
16	0	0 %
17	0	0 %
18	0	0 %
19	0	0 %
20	0	0 %
21	0	0 %
22	0	0 %
23	0	0 %
24	0	0 %
25	0	0 %
26	0	0 %
27	0	0 %
total	107124	100 %

Na figura 7.4 é feita uma comparação entre a codificação da imagem lena fazendo uma busca em todas as escalas possíveis, em somente 8, e em 4, considerando ainda somente o número de vetores utilizados. Note que, como possível resultado da característica *greedy* do algoritmo, a codificação utilizando apenas 4 bases possíveis mostrou melhores resultados que a codificação com um maior número de escalas. Foram considerados somente 4 deslocamentos possíveis para cada bloco, já com o objetivo de preparar a sequência de saída de forma a otimizar o algoritmo da árvore, conforme será visto na seção 7.3.

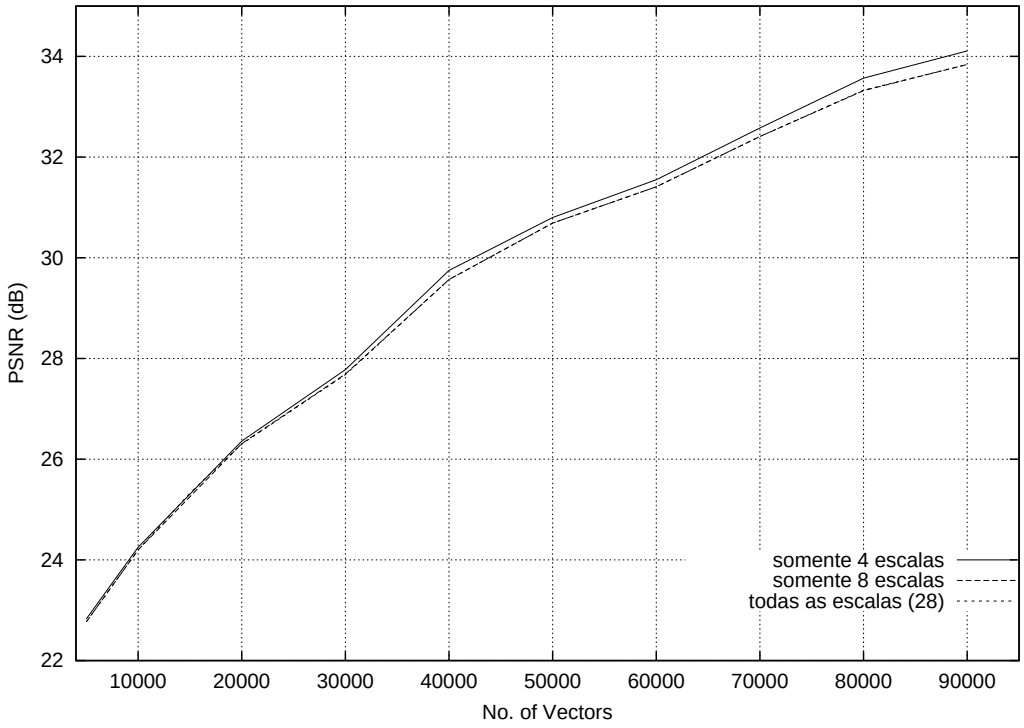

Figura 7.4: Efeito da redução no número de escalas consideradas na codificação da imagem Lena 512×512.

Vale lembrar que os resultados da figura 7.4 ainda consideram somente o número de vetores utilizados. A redução no número de escalas deverá tornar os resultados ainda melhores devido à redução no número de bits utilizados para representar cada vetor.

Conforme foi visto na tabela 7.1, a codificação utilizando todas as escalas possíveis apresenta uma grande ocorrência de vetores de tamanhos 4×4 e 2×2, com um número menor, mas ainda razoável, de vetores de outros tamanhos. Outra

possível explicação para o melhor resultado absoluto apresentado pela codificação com somente 4 escalas é que, nesse caso, praticamente todos os vetores escolhidos estavam associados à blocos 2×2. Isso tende a diminuir os efeitos de bloco, o que pode ter implicação no aumento da PSNR.

Apesar disso, a grande concentração de vetores associados a blocos pequenos tende a fazer com que mais bits sejam gastos pela árvore para sua localização. Em contrapartida, o número de vetores encontrados por bloco tende a aumentar, fazendo com que a geração da sequência de saída tenda a utilizar um menor número de flags.

7.3 Localização dos Vetores de Saída Através da Árvore

A estrutura em árvore apresentada na seção 6.2 diminui consideravelmente o número de bits necessários à localização das escalas. Para a imagem lena em questão, cada deslocamento (horizontal e vertical) precisaria de $\log_2(512) = 9$ bits para ser codificado. O algoritmo proposto foi utilizado para localização dos vetores encontrados na codificação da imagem lena. A tabela 7.2 mostra a redução no número de bits gastos pela estrutura em árvore para a localização dos vetores.

Entretanto, a árvore enxerga um vetor a ser localizado como um pixel, representado pelas coordenadas dsl_h e dsl_v (ver seção 6.2 e figuras 6.3 e 6.4). Todos os vetores, não importando sua escala ou tamanho de bloco associado, são localizados somente após o último nível da árvore. Porém, a redução no número de deslocamentos possíveis torna essa representação ainda mais eficiente. Agora, conforme discutido na seção 6.2, após cada nível de bits da árvore, os vetores relativos àquele tamanho de bloco já podem ser transmitidos e passam a ser desconsiderados nos próximos níveis da árvore. Com isso, o número de bits gastos diminui consideravelmente. A tabela 7.3 mostra o número de bits gastos com a árvore após essa modificação, utilizando apenas quatro posições possíveis por bloco. Note a grande redução no número de bits por vetor em comparação à tabela 7.2.

Tabela 7.2: Bits gastos para localização de vetores através da estrutura em árvore.

nº de vetores utilizados	tamanho da árvore (em bits)	bits de localização /vetor
5000	36608	7.3
10000	59140	5.9
20000	90480	4.5
30000	121228	4.0
40000	142796	3.6
50000	164608	3.3
60000	187064	3.1
70000	213488	3.0
80000	232636	2.9
90000	245536	2.7
100000	257984	2.6

Tabela 7.3: Bits gastos para localização de vetores através da estrutura em árvore após modificação.

nº de vetores utilizados	tamanho da árvore (em bits)	bits de localização /vetor
5000	11068	2.2
10000	19580	1.9
20000	28672	1.4
30000	39524	1.3
40000	45112	1.1
50000	52884	1.1
60000	59944	1.0
70000	73860	∼1.0
80000	81512	∼1.0
90000	82288	0.9

7.4 Codificação Aritmética

A utilização de um codificador aritmético visa melhorar a taxa de compressão da sequência de saída. Conforme explicado na seção 6.5, inicialmente utilizou-se um codificador aritmético com somente um modelo, atribuido um símbolo para cada escala, flag e bit da árvore. Em seguida, o codificador foi adaptado para a utilização de dois modelos, um para as escalas e flags e outro para a árvore, melhorando expressivamente os resultados. A figura 7.5 mostra a comparação entre os dois codificadores. Foram geradas imagens utilizando 4 e 8 escalas e 4 e 16 posições possíveis por bloco. Na figura, "d" indica o número de posições, "e" o de escalas, e "m" o de modelos do codificador aritmético utilizado.

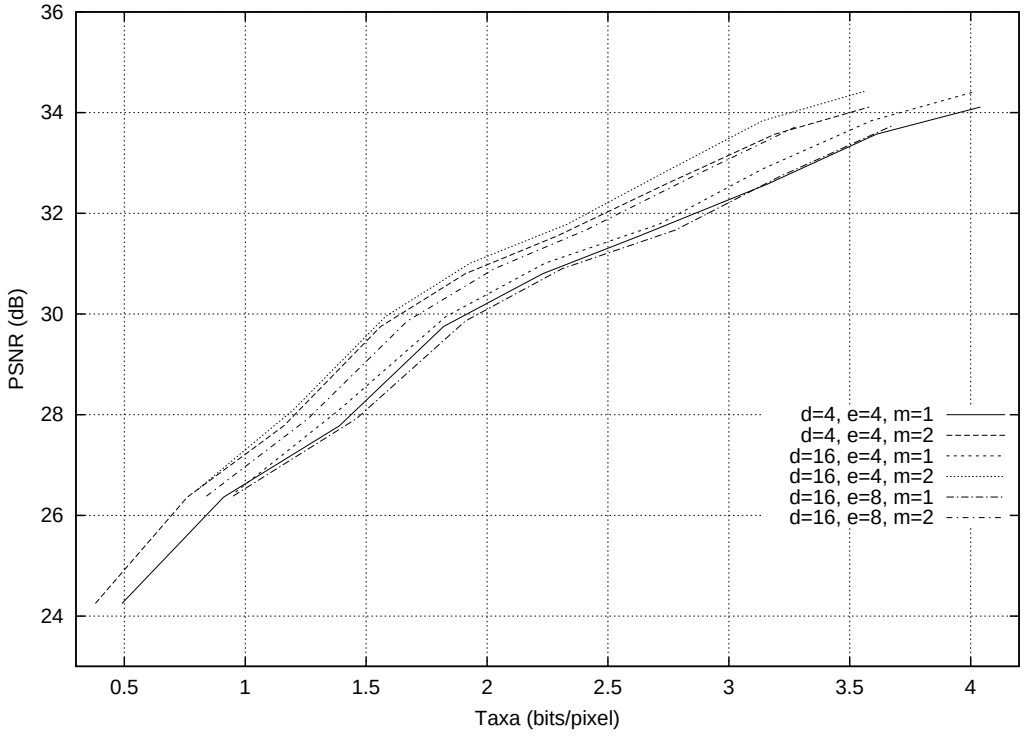

Figura 7.5: Resultados após uso de codificadores aritméticos com um e dois modelos para imagem Lena 512×512.

Capítulo 8

Conclusões e Continuação do Trabalho

Os resultados apresentados no capítulo 7 mostram um desempenho que ainda requer melhoras. As curvas finais de taxa $\times$ distorção da seção 7.4 se encontram cerca de 7 a 9dB abaixo de algoritmos reconhecidamente eficientes, como SPIHT [2], EZW [1] e MGE [8], uma diferença bastante expressiva.

Tais resultados refletem a necessidade de uma pesquisa mais profunda, de forma a se buscar melhores soluções aos problemas encontrados durante o projeto. As soluções e modificações propostas nos capítulos 4 e 5 talvez possam ser reconsideradas e soluções mais eficientes podem ser propostas. Entretanto, alguns pontos possuem direta influência nos resultados obtidos, e merecem destaque. Esses pontos, comentados a seguir, podem e devem ser revistos, como forma de continuação do trabalho.

1. Pelos motivos explicados na seção 4.4, optou-se pela utilização de extensões periódicas durante a decomposição do sinal no dicionário. Entretanto, o motivo principal para esta escolha, a redução significativa no número de vetores, perde importância quando passamos a limitar o número de escalas e de deslocamentos por bloco. Agora, o número de vetores, mesmo considerando-se o caso de extensões simétricas, deixa de crescer de forma proibitiva. Além disso, extensões periódicas contribuem muito para efeitos de blocagem, vistos nas imagens decodificadas apresentadas no capítulo 7. A utilização de extensões simétricas, sem dúvida, causaria uma direta melhoria nos níveis de distorção

dos resultados, ocasionada pelo melhor tratamento das bordas.

2. É possível que as bases de Haar sejam pouco adequadas para a codificação de imagens através do método proposto. Verificou-se que grande parte do vetores codificados (cerca de 90%) estavam associados a blocos de tamanho 4×4 ou 2×2. Isso mostra o insucesso do sistema de bases em representar blocos maiores, de maior energia, o que implica na redução da eficiência do método. Assim, fica evidente que a busca por um dicionário mais adequado à codificação, que apresente um valor para $\Theta(C_N)$ o menor possível, levaria a resultados melhores.

3. Não há indicações fortes de que a organização da sequência de saída da forma apresentada no capítulo 6 seja a mais adequada. Os vetores de saída podem ser organizados de inúmeras formas, podendo levar a melhores ou piores resultados. Um estudo mais detalhado, levando em conta possíveis características particulares da própria sequência poderia permitir uma melhor organização dessa sequência, de forma que menos flags de controle, ou menos bits de localização fossem utilizados.

4. Por fim, uma melhor organização da sequência de saída talvez permita o desenvolvimento de modelos mais adequados para serem utilizados durante a codificação aritmética, implicando numa ainda maior economia de bits para a representação do sinal.

Dentre os pontos anteriores, o caso das extensões deve ser ressaltado. Pode-se tentar modificar o algoritmo proposto para que ele utilize uma decomposição de forma que as dimensões dos vetores utilizados possa ultrapassar as dimensões do bloco a ser codificado. Uma vez que a aproximação em um bloco é independente dos outros, a extensão de um vetor para fora desse bloco pode prejudicar a convergência do bloco vizinho, pois as modificações que seriam causadas em um bloco devido à aproximação de outro são, a princípio, aleatórias. Entretanto, podemos fazer com que um determinado vetor só seja escolhido se ele obrigatoriamente reduzir a norma do resíduo em questão[1], e portanto, contribuir para a redução da norma dos

[1]Em algoritmos *greedy* é possível que num passo intermediário a norma de um resíduo sofra um pequeno aumento.

blocos afetados por ele devido aos coeficientes que se encontram fora do bloco. Após todos os blocos atigirem um mesmo nível de distorção, poderia ser feita uma nova varredura na imagem, para aquele mesmo nível de distorção, de forma a compensar as alterações causadas em um bloco pela aproximação de seus vizinhos. Assim, cada nível de refinamento representado por um expoente poderia ser codificado utilizando as funções que apresentam melhores ganhos de codificação (como as wavelets 9/7) mesmo para os blocos de tamanho 2×2, além de resolver o problema do efeito de blocos.

De qualquer forma, apesar de resultados ainda deficientes, o método proposto apresenta grandes possibilidades de melhoria. Entretanto, por se tratar de um algoritmo novo, poucas são as fontes de referência que poderiam ajudar no tratamento dos problemas da forma mais adequada, e, ao menos em parte dos casos, um estudo particular deverá ser feito.

Apêndice A

Transformadas Wavelet

A.1 Transformada Wavelet Discreta (DWT)

A transformada wavelet de um sinal consiste na decomposição do mesmo em um conjunto de funções-base composto por contrações, expansões e translações de uma função-mãe $\psi(t)$, chamada wavelet.

Qualquer sinal $x(t) \in L^2(\mathbb{R})$ - o espaço de funções que, elevadas ao quadrado, possuem integral finita - pode ser expressa como:

$$x(t) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} \breve{x}_{m,n}\, 2^{-m/2}\, \bar{\psi}(2^{-m}t - n), \tag{A.1}$$

$$\breve{x}_{m,n} = \int_{-\infty}^{\infty} 2^{-m/2}\, \psi(2^{-m}t - n)\, x(t)\, dt, \tag{A.2}$$

onde $\breve{x}_{m,n}$ são os coficientes da transformada wavelet biortogonal discreta de $x(t)$ e as funções $\psi(t)$ e $\bar{\psi}(t)$ são chamadas wavelets de análise e de síntese, respectivamente. Na implementação da transformada wavelet de sinais discretos, também devem ser consideradas as *scaling functions* $\phi(t)$ e $\bar{\phi}(t)$, de análise e de síntese, respectivamente. Estas funções são definidas para evitar os somatórios infinitos da equação (A.1). A *scaling function*, que possui características de filtro passa-baixas, produz coeficientes que representam as informações de baixa resolução no espaço. Os coeficientes das wavelets produzem o refinamento, em diversas etapas (resoluções) da imagem de baixa resolução obtida a partir da *scaling function*. O conjunto formado por estas funções deve ser bi-ortogonal, ou seja, as funções devem satisfazer às seguintes propriedades ($m \in \mathbb{Z}$):

$$\langle\, \phi(t),\, \bar{\phi}(t - m)\, \rangle\, = \delta_m, \tag{A.3}$$

$$\langle\, \psi(t),\, \bar{\psi}(t-m)\,\rangle\, = \delta_m, \tag{A.4}$$

$$\langle\, \phi(t),\, \bar{\psi}(t-m)\,\rangle\, = 0, \tag{A.5}$$

$$\langle\, \psi(t),\, \bar{\phi}(t-m)\,\rangle\, = 0, \tag{A.6}$$

onde

$$\delta_m = \begin{cases} 1, & m = 0, \\ 0, & \text{caso contrário.} \end{cases}$$

Além disto, as seguintes equações devem ser satisfeitas:

$$\phi(t) \;= \sum_n h_{0n}\sqrt{2}\,\phi(2t+n), \tag{A.7}$$

$$\bar{\phi}(t) \;= \sum_n g_{0n}\sqrt{2}\,\bar{\phi}(2t-n), \tag{A.8}$$

$$\psi(t) \;= \sum_n h_{1n}\sqrt{2}\,\phi(2t+n), \tag{A.9}$$

$$\bar{\psi}(t) \;= \sum_n g_{1n}\sqrt{2}\,\bar{\phi}(2t-n). \tag{A.10}$$

Considerando que as funções $\phi(t)$, $\bar{\phi}(t)$, $\psi(t)$ e $\bar{\psi}(t)$ possuem região de suporte finita, as transformadas-z das seqüências h_{0n}, g_{0n}, h_{1n} e g_{1n}, respectivamente $H_0(z)$, $G_0(z)$, $H_1(z)$ e $G_1(z)$, devem satisfazer às seguintes equações:

$$H_0(z)G_0(z) + H_0(-z)G_0(-z) = 2 \tag{A.11}$$

$$G_0(z) = z^{2m-1}\,H_1(-z) \tag{A.12}$$

$$G_1(z) = -z^{2m-1}\,H_0(-z) \tag{A.13}$$

Para que as funções bi-ortogonais sejam suaves, $H_0(z)$ e $G_0(z)$ devem ser filtros passa-baixas e $H_1(z)$ e $G_1(z)$ filtros passa-altas.

No cálculo da transformada wavelet discreta bi-ortogonal de um sinal $x(n)$, considera-se que a seqüência $x_{0,n}$, sua representação digital, equivale à projeção do sinal $x(n)$ sobre o espaço gerado pelas funções $\bar{\phi}(t-n)$, ou seja,

$$x_{0,n} = \int_{-\infty}^{\infty} x(t)\,\phi(t-n)\,dt. \tag{A.14}$$

A transformada wavelet, $\breve{x}_{m,n}$ pode, então, ser calculada através da seguinte recursão:

$$x_{m,n} = \sum_k h_{0k}x_{m-1,2m-k} \tag{A.15}$$

$$\breve{x}_{m,n} = \sum_k h_{1k} x_{m-1,2m-k} \tag{A.16}$$

E, no cálculo da transformada inversa, emprega-se outra recursão:

$$x_{m-1,n} = \sum_k x_{m,k}\, g_{0,n-2k} + \sum_k \breve{x}_{m,k}\, g_{1,n-2k} \tag{A.17}$$

A partir das equações (A.15) e (A.16), observa-se que os coeficientes da transformada wavelet podem ser obtidos filtrando-se a seqüência $x_{0,n}$ com $H_0(z)$ e $H_1(z)$ e sub-amostrando as saídas de ambos os filtros por 2. Repetindo o procedimento com a saída sub-amostrada de $H_0(z)$ obtém-se a decomposição em sub-bandas do sinal por oitavas. Da mesma forma, a equação (A.17) mostra que o sinal discreto original pode ser reconstruído a partir dos coeficientes da transformada wavelet através de um processo de síntese de sub-bandas. A Fig. A.1 ilustra o procedimento apresentado.

A transformada wavelet bi-dimensional pode ser de dois tipos: separável ou não-separável. Nas transformadas separáveis, o cálculo dos coeficientes é feito aplicando-se as recursões definidas nas equações (A.15) e (A.16), primeiramente nas linhas e posteriormente nas colunas da imagem. Além disso, durante o cálculo das convoluções das equações (A.15) e (A.16), deve ser considerado o tipo adequado de extensão a ser feito. Essa extensão pode ser periódica, simétrica ou antisimétrica, e depende do fato dos filtros de análise e síntese possuírem um número par ou ímpar de coeficientes. A extensão correta dos sinais torna a transformada wavelet mais eficiente no sentido de concentrar ainda mais a energia em poucos coeficientes. Quando a extensão é feita de maneira errada as transições das bordas da imagem são erradamente interpretadas como informação de alta freqüência.

A.2 Banco de Filtros Não-Subamostrados

Conforme explicado no capítulo 4, o sistema de bases utilizado para a codificação de imagens foi baseado na estrutura de bancos de filtros apresentada na figura A.1, porém sem os blocos de subamostragem. Assim, todas as bases foram geradas a partir das duas bases características da decomposição, $H_0(z)$ e $H_1(z)$, bem como as imagens de saída apresentadas na figura 4.1. A figura A.5 mostra a estrutura de bancos de filtros que gera as imagens utilizadas na tabela $\langle f; \vec{v} \rangle$.

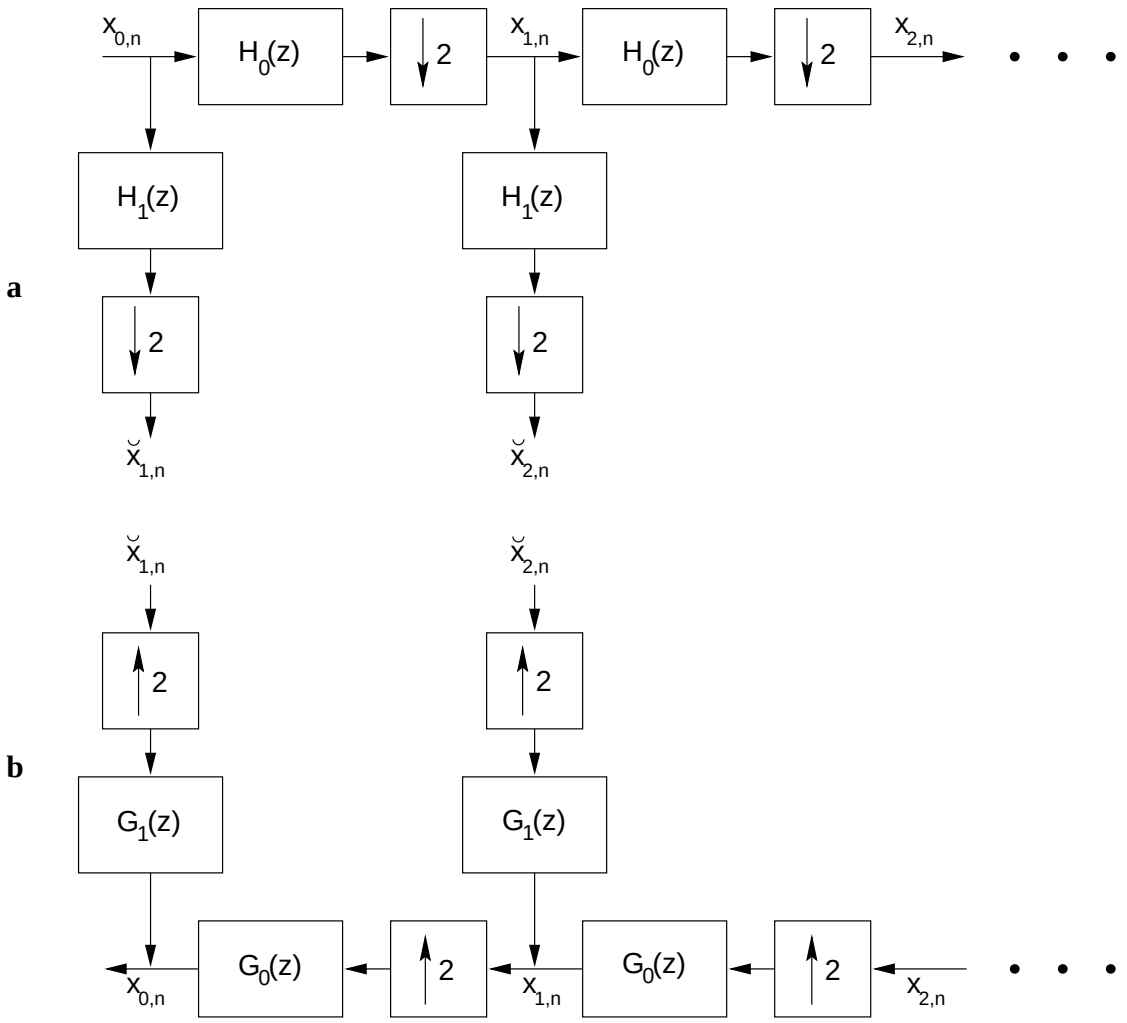

Figura A.1: (a) Cálculo da transformada wavelet discreta; (b) Reconstrução do sinal a partir dos coeficientes da wavelet.

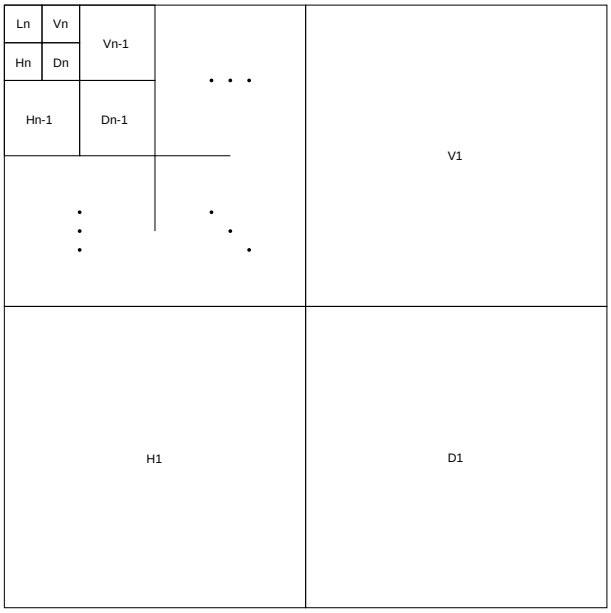

Figura A.2: Transformada wavelet bi-dimensional com n estágios.

Figura A.3: Transformada wavelet bi-dimensional com 3 estágios da imagem Lena.

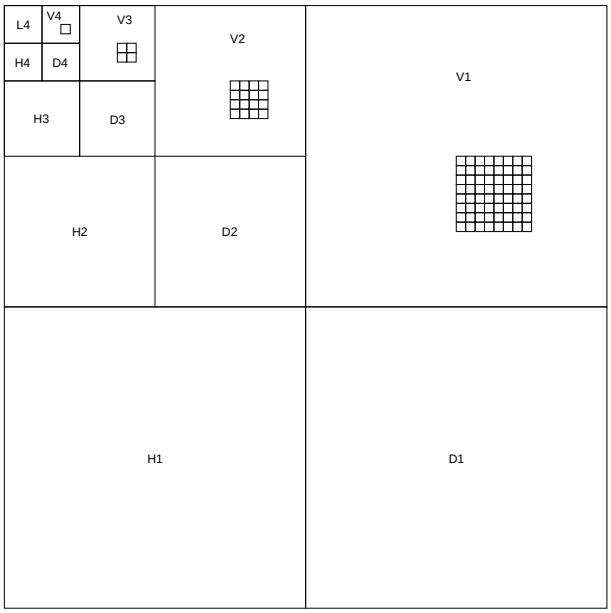

Figura A.4: Coeficientes correspondentes nas bandas $V_i, i = 1, \ldots, 4$.

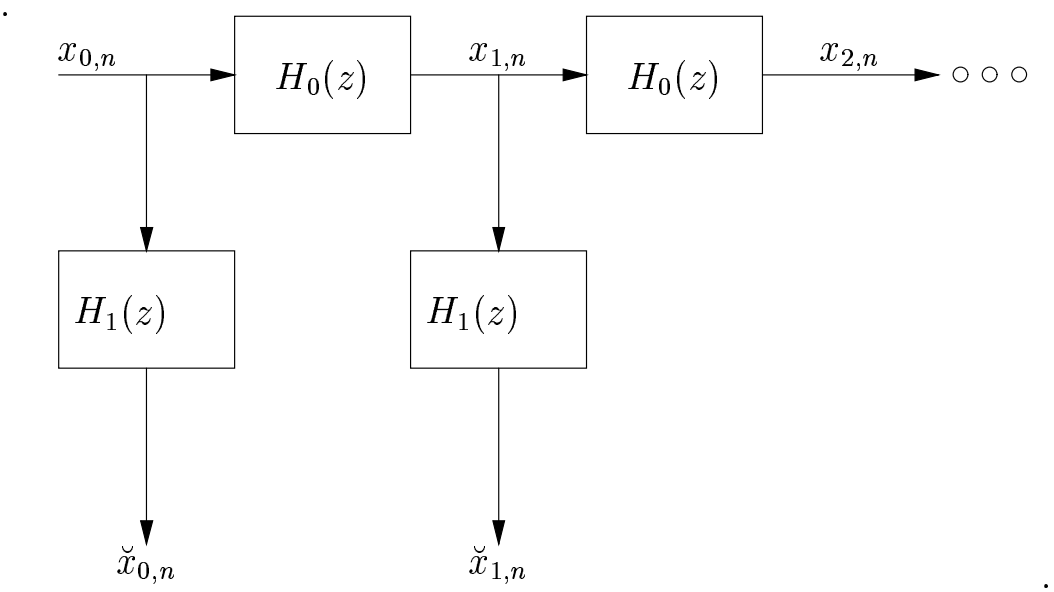

Figura A.5: Banco de filtros não-subamostrado para geração da tabela $\langle f; \vec{v} \rangle$.

Apêndice B

Cálculo da PSNR

A PSNR (*Peak Signal to Noise Ratio*) é uma medida da distorção entra duas imagens. Essa relação é muito utilizada para se ter uma medida quantitativa da eficiência de algum algoritmo de processamento de imagens, principalmente algoritmos de compressão. Ela é definida como

$$\text{PSNR} = 10 \log_{10} \left(\frac{255^2}{\text{MSE}} \right) \text{ dB} \tag{B.1}$$

onde MSE é o erro médio quadrático entre as duas imagens, definido como

$$\text{MSE} = \frac{1}{N} \sum_i \sum_j (p_{i,j} - \hat{p}_{i,j})^2 \tag{B.2}$$

onde N é o número de *pixels* da imagem, $p_{i,j}$ corresponde ao *pixel* de coordenadas (i, j) da imagem original e $\hat{p}_{i,j}$ da imagem processada.

Referências Bibliográficas

[1] SHAPIRO, J. M., "Embedded image Coding using zerotrees of wavelet coefficients", *IEEE Transactions on Acoustic, Speech and Signal Processing*, v. 41, pp. 3445–3462, 1993.

[2] SAID, A., PEARLMAN, W. A., "A new, fast and efficient image codec based on set partitioning in hierarchical trees", *IEEE Trans. Circuits Syst. Video Technologies*, pp. 243–250, Junho 1996.

[3] CRAIZER, M., da SILVA, E. A. B., RAMOS, E. G., "Convergent Algorithms for Successive Approximation Vector Quantization with Applications to Wavelet Image Compression", *IEE Proc.-Visual Image Signal Processing*, v. 146, n. 3, June 1999.

[4] da SILVA, E. A. B., *Image Compression using Wavelet TRansforms*. Ph.D. dissertation, University of Essex, 1994.

[5] GERSHO, A., GRAY, R. M., *Vector Quantization And Signal Compression*, Engineering and Computer Science. 1 ed. Kluwer Academic Publishers, 1992.

[6] da SILVA, E. A. B., SAMPSON, D. G., GHANBARI, M., "A Successive Approximation Vector Quantizer for Wavelet Transform Image Codind", *IEEE Transactions on Image Processing, Special Issue on Vector Quantization*, v. 5, n. 2, pp. 299–310, Fevereiro 1996.

[7] MALLAT, S., *A Wavelet Tour of Signal Processing*. Academic Press, 1998.

[8] LAN, T.-H., TEWFIK, A. H., "Multigrid Embedding (MGE) Image Coding". In: *Proceedings of the 1999 International Conference on Image Processing*, Kobe, 1999.

[9] C.BELL, T., CLEARY, J. G., WITTEN, I. H., *Text Compression*. Prentice Hall, 1990.

[10] MALLAT, S., ZANG, Z., "Matching Pursuits With Time-Frequency Dictionaries", *IEEE Transactions on Signal Processing*, v. 41, n. 12, pp. 3397–3415, Dezembro 1993.

[11] VETTERLI, M., KOVACEVIĆ, J., *Wavelets and Subband Coding*. Prentice Hall, 1995.