Universidade Federal do Rio de Janeiro

Escola Politécnica

Departamento de Eletrônica e de Computação

Estudo das Propriedades de Algoritmos de Quantização Vetorial baseados em Kernel PCA

Autor:

Vitor Rosa Meireles Elias

Orientador:

Prof. José Gabriel Rodríguez Carneiro Gomes, Ph. D.

Examinador:

Prof. Antonio Petraglia, Ph. D.

Examinador:

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

DEL

Agosto de 2013

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
Escola Politécnica - Departamento de Eletrônica e de Computação
Centro de Tecnologia, bloco H, sala H-217, Cidade Universitária
Rio de Janeiro - RJ CEP 21949-900

DEDICATÓRIA

Aos meus pais.

AGRADECIMENTO

Agradeço à minha família pelo apoio incondicional para que eu pudesse me dedicar à faculdade. Especialmente, agradeço à minha mãe Sandra por ser a pessoa mais forte deste mundo e por dedicar sua vida para que eu tivesse educação, caráter e tanto carinho. Especialmente, também, agradeço ao meu pai Roberto, por todo sacrifício e esforço para me dar condições necessárias para minha formação.

Agradeço aos meus amigos de faculdade, que estiveram ao meu lado por todos estes anos de graduação. Em especial, agradeço a Vitor Borges, Luciana Reys, Vitor Tavares, Jonathan Gois e Nilson Carvalho, que romperam as barreiras da faculdade e se tornaram pessoas importantes para toda a vida. Agradeço aos meus amigos de fora da faculdade, que sempre entenderam meus motivos para sumir ao início de cada período e provaram o quão forte é a amizade, em especial Vitor Santos, Pedro Marcus e Jefferson Oliveira.

Aos colegas do Laboratório de Processamento Analógico e Digital de Sinais, por toda ajuda e pelo apoio que me deram com o projeto.

Agradeço aos professores do Departamento de Engenharia Eletrônica e de Computação, todos fundamentais para minha formação. Especialmente, agradeço ao meu orientador José Gabriel Rodríguez Carneiro Gomes, por toda paciência e dedicação em tirar minhas dúvidas e por tudo que me ensinou.

RESUMO

Esta dissertação apresenta um estudo de métodos de quantização vetorial, introduzindo propriedades de quantizadores baseados na análise de componentes principais usando funções Kernel e os comparando a métodos já usualmente aplicados na engenharia. O estudo investiga a viabilidade da implementação destes quantizadores em métodos de compressão de dados.

Os quantizadores são detalhados em sua teoria e são apresentadas simulações dos mesmos. As propriedades estudadas incluem complexidade e relação entre desempenho de compressão e entropia.

Palavras-Chave: quantização vetorial, funções Kernel, análise de componentes principais, compressão de dados.

ABSTRACT

This dissertation presents a study of vector quantization methods, introducing properties of quantizers based on principal component analysis with Kernel functions and comparing these to the methods usually applied in engineering. This study shows the feasability of the implementation of these quantizers in data compression methods.

The quantizers are detailed in their theory and their simulations are presented. The properties under study include complexity and the relationship between compression perfomance and entropy.

Keywords: vector quantization, Kernel functions, principal component analysis, data compression.

SIGLAS

AMS - Austriamicrosystems

CMOS - *Complementary Metal-Oxide Silicon*

COPPE - Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia

DPCM - *Differential Pulse Code Modulation*

ECVQ - *Entropy Constrained Vector Quantization*

GLA - *Generalized Lloyd Algorithm*

LBG - Linde-Buzo-Gray *Algorithm*

PADS - Laboratório de Processamento Analógico e Digital de Sinais

PCA - *Principal Component Analysis*

pdf - *Probability Density Function*

SNR - *Signal-to-Noise Ratio*

SQ - *Scalar Quantization*

UFRJ - Universidade Federal do Rio de Janeiro

VQ - *Vector Quantization*

WTA - *Winner-takes-all*

XOR - *Exclusive Or*

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Tema

Este trabalho aborda conceitos de processamento de sinais aplicados à engenharia eletrônica. Está voltado à otimização e proposta de algoritmos de compressão de dados, para a implementação em circuitos eletrônicos dedicados ao processamento de imagens. Dentro da área de compressão de dados, o foco do projeto é o estudo dos métodos de quantização.

1.2 Delimitação

Muitos dos conceitos e algoritmos aqui apresentados possuem ampla área de utilização, sendo envolvidos em diversas áreas do processamento de sinais. Os métodos de quantização, feitas as devidas considerações e adaptações, podem ser empregados desde em circuitos que realizam conversão analógico-digital a rebuscados sistemas de reconhecimento de padrões e clusterização. Da forma como são propostos neste trabalho, estes métodos encontram-se dedicados ao uso em compressão de imagens.

1.3 Justificativa

Uma câmera CMOS (*complementary metal-oxide silicon*) foi desenvolvida em um projeto anterior no PADS (Laboratório de Processamento Analógico e Digital de Sinais, COPPE/UFRJ) e fabricada através da AMS (Austriamicrosystems). Esta

câmera captura imagens em escala de cinza e realiza seu processamento e compressão ainda no plano focal, ou seja, antes da transmissão dos dados para fora da matriz de foto-sensores [1].

A saída desta câmera é unicamente digital. As imagens capturadas têm resolução de 32 x 32 pixels e as técnicas de compressão são aplicadas sobre blocos de 4 x 4 pixels. As técnicas aplicadas nesta câmera são *differential pulse code modulation* (DPCM), transformação linear e quantização vetorial. Como o processamento é realizado no plano focal, todas as operações aritméticas relacionadas à utilização destas técnicas estão implementadas em *hardware* analógico conectado diretamente aos foto-sensores.

Visando projetos futuros que possam aperfeiçoar esta câmera, uma das formas de prover aprimoramentos é estudando novos métodos de processamento de sinais que possam ser implementados de modo a substituir ou complementar aqueles já existentes. Com isso, surge a necessidade do estudo abordado neste projeto. Como dito anteriormente, projetos de desenvolvimento em cima de técnicas de quantização vetorial possuem vasta área de aplicação, não estando restritos ao uso para a câmera CMOS.

1.4 Objetivos

A proposta principal deste projeto é o estudo de quantizadores vetoriais baseados em análise de componentes principais usando funções Kernel. Espera-se chegar a resultados que mostrem ser possível a realização destes quantizadores, de modo que seu desempenho apresente competitividade frente aos métodos de quantização tradicionais, como aquele já empregado na câmera CMOS. Além disso, se espera obter, a partir deste trabalho, informações sobre a complexidade associada à implementação para mensurar a viabilidade de aplicação real do quantizador em questão.

1.5 Descrição

O Capítulo 2 apresenta a teoria envolvida no estudo dos métodos de quantização, incluindo conceitos básicos para o entendimento do trabalho. A metodologia de implementação destes métodos e abordagem dos conceitos é descrita no Capítulo 3. O Capítulo 4 mostra os resultados obtidos para as simulações e estudos realizados. Ao final, o Capítulo 5 traz a conclusão do trabalho e as atividades futuras relacionadas a este projeto.

Capítulo 2

Teoria

Neste capítulo, serão apresentados os conceitos teóricos abordados durante o projeto e necessários para o entendimento deste texto. Será introduzido o conceito de quantização, enfatizando as técnicas de quantização escalar e vetorial, empregadas nos outros capítulos, bem como a quantização vetorial baseada em funções Kernel, foco deste trabalho. Os conceitos de taxa variável, complexidade de implementação dos algoritmos e análise de componentes principais também serão abordados como complementos para os tópicos principais.

2.1 Quantização

Quantização, no âmbito de processamento de sinais, é um processo que consiste na representação de um conjunto de dados relativamente grande, possivelmente infinito, por um conjunto de dados relativamente menor e controlado. Sua aplicação é amplamente encontrada em sistemas de compressão de dados com perdas. Este processo é realizado através de um mapeamento do conjunto de dados original para um novo conjunto, muitas vezes levando em consideração características estocásticas da entrada. Um algoritmo ou dispositivo que implemente esta função é denominado *quantizador*.

O conceito da quantização não está muito distante de aplicações cotidianas. Quando um preço de \$1299,90 é tratado como \$1300, o arredondamento caracteriza um processo de quantização. Quando uma idade é apresentada como “18 anos” e não por

uma representação que pode ser considerada completa, como "18 anos, 5 meses, 23 dias, etc.", também é empregado este conceito. Nos dois casos, pode-se notar uma consequência da quantização, que é a perda de parte da informação. Porém, para estes casos, toda a parte essencial da informação foi mantida. Um bom quantizador deve ser capaz de manter um compromisso entre a compressão de dados e a perda de informação resultante.

A entrada de um quantizador pode ser definida por um vetor-coluna $\mathbf{x}$ que será mapeado em um novo vetor $\hat{\mathbf{x}}$ segundo uma matriz $\mathbf{Y}$ que contém K vetores-coluna $\mathbf{y}$. Temos que $\mathbf{x}$, $\hat{\mathbf{x}}$ e $\mathbf{y} \in \mathbb{R}^M$, sendo M a dimensão dos dados que o quantizador deve comprimir. Fazendo com que a entrada seja conjunto de vetores, podemos definir a matriz $\mathbf{X}$ como uma matriz de vetores de entrada $\mathbf{x}_n$, $n = 1, \ldots, N$. Esta matriz $\mathbf{X} \in \mathbb{R}^{M \times N}$ recebe o nome de *matriz de dados*. A matriz $\mathbf{Y} \in \mathbb{R}^{M \times K}$ é denominada *dicionário* ou *codebook*.

O quantizador realiza o mapeamento $\hat{\mathbf{x}}_n = Q(\mathbf{x}_n)$, que pode ser definido como a associação em cascata de um codificador e um decodificador [2], como na Figura 2.1. O codificador α representa a *partição* e será responsável por mapear o vetor $\mathbf{x}_n$ em um número inteiro $j_n = \alpha(\mathbf{x}_n)$. O decodificador β busca no dicionário o vetor $\mathbf{y}_k$ associado ao índice j_n. Temos, portanto, $\hat{\mathbf{x}}_n = Q(\mathbf{x}_n) = \beta(\alpha(\mathbf{x}_n))$. Em outras palavras, o quantizador Q particiona o espaço em *células* ou *regiões*, que são representadas por um único vetor de reconstrução.

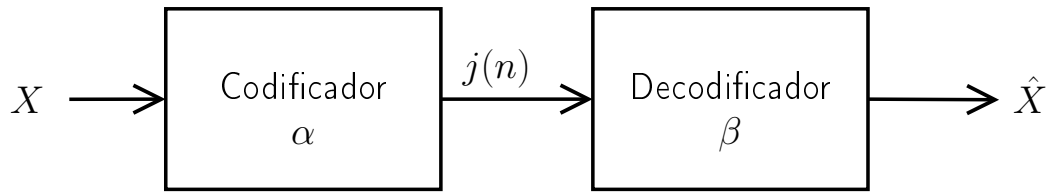


Figura 2.1: Diagrama de representação de um quantizador como codificador e decodificador.

As perdas de um quantizador (e, consequentemente, sua qualidade com relação a desempenho) são definidas em função do *erro de quantização* gerado no mapeamento. O erro de quantização representa a diferença entre o valor real $\mathbf{x}_n$ e o valor

quantizado $\hat{\mathbf{x}}_n$. Para quantificar este valor e caracterizar a quantização, a *distorção* D de um quantizador será definida como o *erro médio quadrático* do mapeamento de N vetores:

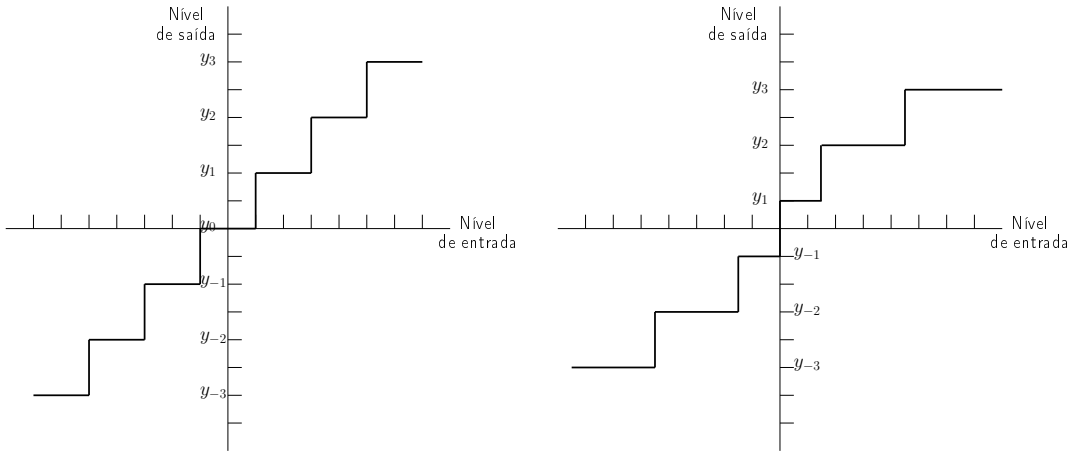
$$D = \frac{1}{N} \sum_{n=1}^{N} ||\mathbf{x}_n - \hat{\mathbf{x}}_n||^2. \tag{2.1}$$

Para o caso da representação com palavras binárias de comprimento fixo, a quantidade K de vetores do dicionário é definida pela *taxa de bits* R do quantizador, de modo que $K = 2^R$. Quanto maior a quantidade de bits utilizada no quantizador, maior o tamanho do dicionário e portanto, melhor pode ser o resultado da compressão em termos de distorção. Para um dicionário com tamanho K igual à quantidade de vetores de entrada N, pode-se atribuir um elemento $\mathbf{y}_k$ a cada $\mathbf{x}_n$, resultando em distorção zero. Deseja-se, porém, reduzir K, mantendo, ainda, distorção aceitável para determinada aplicação.

2.2 Quantização Escalar

Quando o conjunto de dados possui dimensão $M = 1$, o processo de quantização é denominado *quantização escalar* (SQ, de *scalar quantization*, ou *quantizer*, conforme o contexto). Portanto, o quantizador escalar trabalha com valores escalares x_n na entrada e $\hat{x}_n$ na saída. Esta técnica é utilizada, por exemplo, para a conversão analógico-digital de sinais. É o método mais simples de quantização e é o aplicado nos exemplos do início da Seção 2.1. Nos exemplos, tem-se um conjunto de dados onde a única dimensão é o preço e outro conjunto de dados onde a única dimensão é o tempo. Ou seja, o processo de arredondamento é um processo de quantização escalar.

Um SQ pode particionar o espaço de forma linear (uniforme) ou não-linear (não-uniforme), como apresentado na Figura 2.2. A quantização linear cria partições de mesmo tamanho Δ. Por exemplo, um quantizador de 8 bits terá $2^8 = 256$ intervalos. Se o sinal de entrada for um sinal de tensão de -1 V a 1 V, este espaço de 2 V será dividido em intervalos iguais de $\frac{2}{256}$ V.



(a) (b)

Figura 2.2: (a) Quantizador *meio-piso* linear; (b) Quantizador *meio-degrau* não linear.

Como a quantização linear mapeia intervalos Δ iguais da entrada, o erro de quantização será o mesmo para todos estes intervalos e pode ser estimado teoricamente. Usualmente, o valor de reconstrução definido para uma partição de um SQ está localizado no meio do intervalo delta. O erro de quantização é dado pela diferença entre o valor real e o valor após a quantização. Assumindo que o valor real pode ter, uniformemente, qualquer valor dentro do intervalo Δ, o erro de quantização terá valores distribuídos uniformemente entre $-\Delta/2$ e $+\Delta/2$ para todos os intervalos. Isto faz com que a relação sinal-ruído (SNR, de *signal-to-noise ratio*) seja diretamente dependente da amplitude do sinal de entrada do quantizador uniforme. Este efeito, algumas vezes, não é desejado. A quantização não-linear muda esta característica, aumentando o tamanho dos intervalos, e consequentemente o ruído, conforme aumenta o valor da entrada, de modo que a SNR fique constante [2].

2.3 Quantização Vetorial

A quantização vetorial (VQ, de *vector quantization*, ou *quantizer*) é uma generalização da quantização escalar, em que se trabalha com dados de dimensão $M > 1$, como no caso da Figura 2.3. Com isso, a entrada do quantizador deixa de ser um valor x_n e passa a ser um vetor, representado como $\mathbf{x}_n$. A possibilidade de trabalhar com mais de uma dimensão permite significativa evolução no processamento de

sinais, possibilitando a exploração de dados mais complexos ou mesmo uma análise mais profunda de dados relativamente simples que poderiam ser representados em uma única dimensão. A VQ é utilizada, frequentemente, para compressão de dados. Este é o caso estudado neste trabalho.

Da mesma forma que um SQ, o quantizador vetorial define partições e centroides para mapear os dados de entrada. Com a diferença de que, agora, dados, partições e centroides são multidimensionais. A estrutura geral é a mesma, podendo ser representada como um codificador em cascata com um decodificador. O vetor de entrada $\mathbf{x}_n$ é representado por um índice j_n, que faz a associação com um dos vetores $\mathbf{y}_k$ do dicionário $\mathbf{Y}$. A saída, então, será $\hat{\mathbf{x}}_n = \mathbf{y}_{j_n}$.

O VQ sempre alcança desempenho igual ao superior ao de qualquer outra forma de quantização, quando se refere à distorção na reconstrução do sinal. Para mostrar isso, seja uma técnica de codificação qualquer, que tenha como saída um vetor entre K possíveis vetores $\mathbf{y}_k$. Tomando estes vetores, podemos definir o dicionário $\mathbf{Y}$ de um VQ com os mesmos valores. Podemos definir também que o codificador α seja idêntico ao desta outra técnica. Desta forma, o VQ alcançaria o mesmo desempenho desta técnica de codificação qualquer.

Em relação à quantização escalar, o VQ se faz melhor uma vez que utiliza informações conjuntas entre as dimensões das amostras para definir os parâmetros da quantização. A SQ trata as dimensões de forma independente. Para dicionários relativamente pequenos, a VQ pode ser uma técnica viável, mas para algumas aplicações, como por exemplo técnicas de codificação de voz, que podem utilizar dicionários com $K = 4096$, esta técnica torna-se de alto custo computacional.

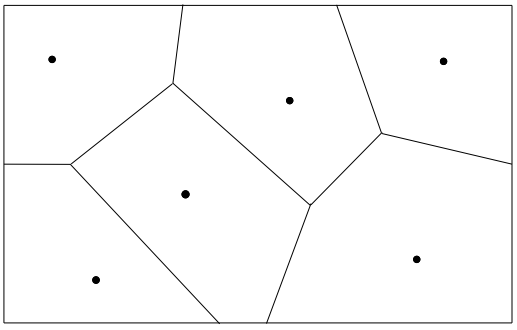


Figura 2.3: Um VQ de duas dimensões.

2.4 Quantização Vetorial com Restrição de Entropia

Quando é empregado um VQ para quantizar um vetor $\mathbf{x}$, com o dicionário adequado para a densidade de probabilidade deste vetor, pode-se esperar que o desempenho obtido seja o melhor possível de se obter num processo de quantização. O desempenho de um VQ, porém, está preso ao tamanho do seu dicionário. Quando tratamos de transmissão de dados por um canal, o índice do dicionário pode ser escrito como uma palavra binária. Com isso, o número de bits necessário para indexar um vetor de um dicionário de um VQ com K vetores $\mathbf{y}_k$ é $R = \log_2 K$.

2.4.1 Taxa Variável

Até agora, tanto no SQ como no VQ, assumimos que a taxa de bits do quantizador era fixa. Ou seja, em um VQ ou SQ com taxa R bits, todas as palavras binárias que codificam as posições do dicionário possuem R bits. Com o estudo das características probabilísticas do sinal de entrada, é possível utilizar informações sobre esta distribuição de modo a aperfeiçoar a representação binária dos vetores.

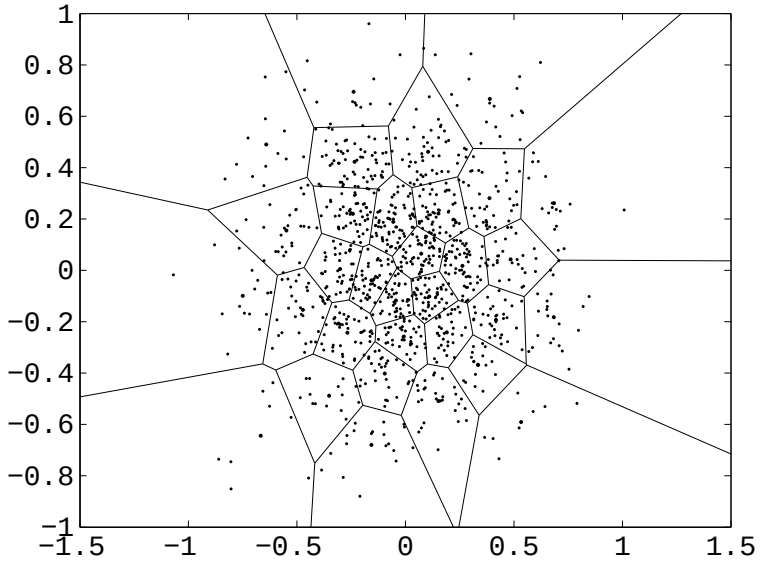


Figura 2.4: Quantização vetorial de amostras de uma distribuição gaussiana.

Dada uma certa distribuição, existem regiões do espaço onde a probabilidade de existirem amostras é maior. Desta forma, uma célula do quantizador localizada nestas regiões apresenta um maior número de ocorrências. Isto significa que o centroide desta célula será utilizado com maior frequência que o centroide de uma célula numa região onde a probabilidade de ocorrência das amostras é menor. Na Figura 2.4, por exemplo, observa-se que amostras de uma distribuição gaussiana normalizada são muito mais prováveis em torno da origem, de modo que as células do quantizador, nesta região, tornam-se mais densas. Essa informação pode ser aproveitada dentro do VQ.

O método de aplicação da *taxa variável* consiste em atribuir, a uma célula cuja probabilidade de receber um vetor de entrada seja relativamente elevada, um menor número de bits para a indexação binária do vetor correspondente no dicionário. Isto é, a alocação de bits prioriza células com menor densidade de amostras da entrada. Quando for necessária a transmissão destes dados, os centroides com representação mais longa serão transmitidos menos vezes, enquanto poucos bits serão transmitidos para os centroides muito prováveis.

Este conceito de taxa variável não está relacionado com variações da taxa R de um quantizador, mas sim com a busca por representações binárias de vetores do dicionário que otimizem, na média, a taxa de transmissão dos dados quantizados.

Tabela 2.1: Exemplo de codificações de taxa fixa e taxa variável.

Região	Probabilidade	Cod. Taxa Fixa	Cod. Taxa Variável
1	0,5	00	0
2	0,25	01	10
3	0,125	11	110
4	0,125	10	111

Quando as probabilidades de ocorrência de amostras de entrada forem iguais em todas as células, o resultado para taxa variável será equivalente ao da taxa fixa. No exemplo da Tabela 2.1, foi utilizada a *codificação de Huffman* (uma breve explicação encontra-se no Apêndice A) [3]. Para as probabilidades apresentadas, obtém-se o

comprimento médio $\overline{L} = [0{,}5\times(1\text{ bit})] + [0{,}25\times(2\text{ bits})] + [0{,}25\times(2\text{ bits})] + [0{,}125\times(3\text{ bits})] = 1{,}75$ bit, contra $\overline{L} = 2$ bits para a codificação com taxa fixa.

Sendo $p(k)$ a probabilidade associada a cada célula do dicionário, a entropia associada à variável aleatória k_n é:

$$H = -\sum_{k} p(k) \log_2 p(k). \tag{2.2}$$

A redução do comprimento médio utilizando taxa variável está limitada pela entropia da variável aleatória k_n. Para o exemplo dado na Tabela 2.1, a entropia é 1,75 bits e por isso não há método que dê taxa menor.

2.4.2 Estrutura do Quantizador

A partir do conceito de taxa variável, levando a idéia de redução de entropia para dentro do quantizador, chega-se à *quantização vetorial com restrição de entropia* (ECVQ, de *entropy-constrained vector quantization*). Neste caso, o quantizador não considera apenas o desempenho sobre a distorção, como no caso do VQ apresentado na Seção 2.3. Um ECVQ tem partição e dicionário definidos de modo que a entropia resultante na saída do quantizador também seja minimizada. A diferença em relação ao VQ está na função custo, definida como:

$$J = D + \lambda H. \tag{2.3}$$

Tanto a distorção D como a entropia H são significativos para a determinação das partições. O multiplicador de Lagrange λ é escolhido de forma a representar o compromisso entre taxa e distorção. Para $\lambda = 0$, temos o caso do VQ com taxa fixa sem restrição de entropia. Se λ for muito alto, temos $H = 0$ e distorção máxima.

2.5 Algoritmos para Projeto de VQ

Como visto, o codificador e o decodificador são as partes básicas de um quantizador. Portanto, a forma como um quantizador é projetado influencia diretamente no seu desempenho geral.

2.5.1 Complexidade

A complexidade de um quantizador se refere ao esforço computacional necessário para realizar o mapeamento $\hat{\mathbf{x}}_n = Q(\mathbf{x}_n)$. Está associada ao número de operações matemáticas necessárias, como multiplicações ou cálculos de outras funções que podem ser necessárias na implementação do quantizador.

Aumentando-se a taxa de bits, por exemplo, é de se esperar que sejam necessários mais cálculos, uma vez que o dicionário é maior. Isto se reflete em um maior número de componentes necessários para a implementação de um quantizador, mais tempo necessário para realizar o mapeamento e maior dificuldade no desenvolvimento do projeto. Tudo afeta diretamente o custo do processo. Portanto, a complexidade é um dos principais fatores a serem considerados para o projeto de um bom quantizador.

2.5.2 Algoritmo de Lloyd

O algoritmo de Lloyd é um dos procedimentos mais conhecidos para a busca de um quantizador que apresente as condições ótimas para o mapeamento, limitando o dicionário a um tamanho K. Este algoritmo foi amplamente aplicado em projetos de quantizadores de uma dimensão desde sua publicação em 1957 por Stuart P. Lloyd [4] e em 1960 por Joel Max [5]. O parâmetro básico para o desenvolvimento deste algoritmo é a distorção do quantizador, apresentada na Equação (2.1). Para um dado codebook $\mathbf{Y}$ com uma distorção D, procura-se um dicionário vizinho que reduza esta distorção. É um algoritmo iterativo e de implementação simples e foi detalhado na Tabela 2.2.

2.5.3 Algoritmo de Lloyd Generalizado

O Algoritmo de Lloyd original se aplica ao projeto de um SQ, ou seja, para dados em uma dimensão. Mas seu conceito pode ser estendido e aplicado a mais de uma dimensão, podendo ser empregado em VQ. Neste caso, o algoritmo é conhecido como Algoritmo de Lloyd Generalizado (GLA, de *Generalized Lloyd Algorithm*) ou Algoritmo de Linde-Buzo-Gray (LBG) [6].

O LBG tem a mesma estrutura do algoritmo de Lloyd, com a única diferença de que os elementos são, agora, multidimensionais e, portanto, a distância calculada será a distância euclidiana entre os vetores de entrada e os vetores do dicionário.

Tabela 2.2: Algoritmo de Lloyd.

1. **Inicialização**: Gerar um dicionário inicial $\mathbf{Y}$

   - Possívelmente aleatório. A inicialização pode facilitar o processo, então é desejável que a fonte de dados seja a mesma das amostras de entrada.

2. **Condição de partição**: Dividir a entrada $\mathbf{X}$ segundo o dicionário $\mathbf{Y}$.

   - Gerar células contendo os valores de entrada mais próximos de cada valor do dicionário.
   - Como parâmetro, calcular a distorção D_a associada.

3. **Condição de centroide**: Recalcular os centroides de cada célula e atualizar $\mathbf{Y}$.

   - O cálculo dos novos valores y_k pode ser a média dos valores de entrada associados a cada célula.
   - Calcular a nova distorção D_b associada.

4. **Iteração**: Repetir os passos 2 e 3 até que se atinja a situação desejada.

   - A repetição pode acontecer até que a distorção convirja para um valor final, ou por quantas repetições forem determinadas.
   - Pode-se considerar também um valor ϵ, de modo que a iteração acabe quando $|D_a - D_b| < \epsilon$.

2.6 Quantização Vetorial baseada em Kernel PCA

Os métodos de quantização apresentados até agora realizavam a codificação diretamente sobre o espaço original dos dados da entrada. A proposta deste trabalho é introduzir um mapeamento intermediário, levando os dados a um espaço onde as informações possam ser tratadas de forma diferente, para depois realizar a quantização. Primeiramente, é necessário apresentar o conceito de Análise de Componentes Principais (PCA, de *Principal Component Analysis*).

2.6.1 PCA Linear

A Análise de Componentes Principais faz o uso de uma transformação linear sobre os dados, de modo a reorganizar a energia da densidade ao longo de direções ortogonais entre si. A PCA concentra a energia de modo decrescente com relação às novas coordenadas, isto é, a primeira coordenada é aquela que representa a parcela de maior energia, enquanto a última coordenada é a parcela de menor energia da densidade de probabilidade (Figura 2.5).

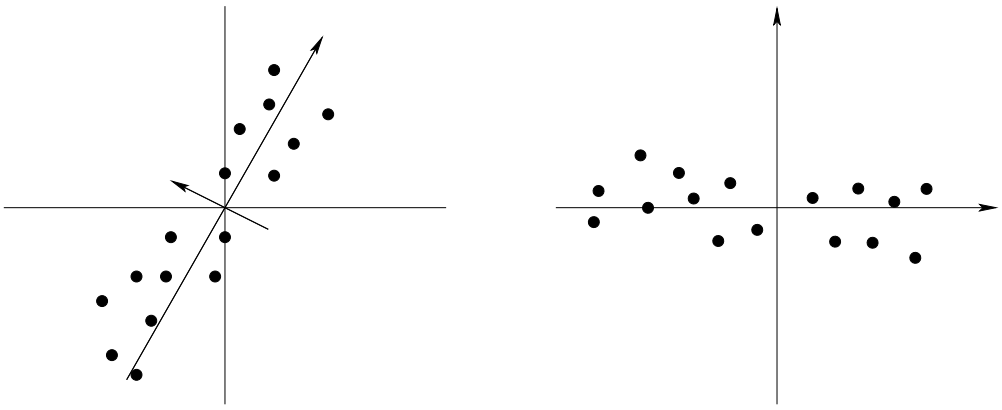
Seja um conjunto de dados normalizado (média zero e desvio padrão unitário) $\mathbf{X}$, sendo $\mathbf{X} \in \mathbb{R}^{M \times N}$, onde M é a dimensão dos N vetores de entrada. Sua matriz de covariâncias $\mathbf{C}_X \in \mathbb{R}^{M \times M}$, que, no caso da média zero, coincide com a matriz de autocorrelação $\mathbf{R}_X$, é dada por:

$$\mathbf{C}_X = \frac{\mathbf{X}\mathbf{X}^T}{N}. \tag{2.4}$$

Calculando os autovalores e autovetores de $\mathbf{C}_X$, pode-se obter as características de energia da distribuição. Autovalores são proporcionais à variância dos dados ao longo das direções definidas pelos respectivos autovetores. Estes autovetores são as *componentes principais* ou *features* da distribuição. O conjunto destas componentes forma a matriz $\mathbf{W} \in \mathbb{R}^{M \times M}$, organizada segundo os autovalores obtidos, em ordem decrescente de energia.

Com relação à informação contida nos dados, as componentes de mais energia são as mais importantes. Desta forma, eliminar componentes principais de baixa

energia significa reduzir a quantidade de dados minimizando a perda de informação, formando um conjunto de *features* apenas com as componentes principais desejadas.



(a) (b)

Figura 2.5: (a) Conjunto de dados e suas componentes principais; (b) conjunto de dados após PCA.

O conjunto de dados de entrada pode ser então mapeado para um novo espaço, com ou sem compressão de dados, denominado *espaço de features* $\mathcal{F}$ (*feature space*, em inglês). Isto gera uma nova representação para os vetores. Este mapeamento é feito projetando-se os vetores de entrada sobre as componentes principais.

$$\mathbf{f} = \mathbf{W}^T\mathbf{X}. \tag{2.5}$$

2.6.2 Kernel PCA

Na Kernel PCA, antes de encontrarmos as componentes principais das amostras na entrada, é realizado um mapeamento não linear $\Phi : \mathbb{R}^M \to \mathbb{R}^I$ sobre este conjunto de dados, onde a quantidade I de dimensões é um número grande, possivelmente infinito. Cada vetor $\mathbf{x}_n$ é mapeado em um vetor-coluna $\mathbf{q}_n = \Phi(\mathbf{x}_n)$, formando a matriz $\mathbf{Q} = [\mathbf{q}_1\ \mathbf{q}_2\ ...\ \mathbf{q}_N]$. A partir daí, pode-se calcular até N componentes principais deste novo conjunto de dados.

Quando a matriz de covariâncias é calculada, substituindo $\mathbf{Q}$ no lugar de $\mathbf{X}$ na Equação (2.4), obtem-se a matriz $\mathbf{C}'_X \in \mathbb{R}^{I\times I}$. Calcular os autovetores e autovalores desta matriz através da equação $\mathbf{C}'_X\mathbf{v}_j = \rho_j\mathbf{v}_j$ é uma tarefa difícil, uma vez que $\mathbf{C}'_X$

tem dimensões arbitrariamente grandes. Portanto, serão calculados indiretamente. Sabe-se que os autovetores podem ser expressos como uma combinação linear das colunas de $\mathbf{Q}$, ou seja, $\mathbf{v}_j = \sum_i a_{ij}\mathbf{q}_i = \mathbf{Q}\mathbf{a}_j$. Unindo as duas últimas equações:

$$\begin{aligned} \mathbf{C}'_X\mathbf{v}_j &= \rho_j\mathbf{v}_j, \\ \mathbf{C}'_X\mathbf{Q}\mathbf{a}_j &= \rho_j\mathbf{Q}\mathbf{a}_j, \\ \frac{\mathbf{Q}\mathbf{Q}^T}{N}\mathbf{Q}\mathbf{a}_j &= \rho_j\mathbf{Q}\mathbf{a}_j. \end{aligned} \tag{2.6}$$

Multiplicando os dois lados da equação por $\mathbf{Q}^T$:

$$\mathbf{Q}^T\frac{\mathbf{Q}\mathbf{Q}^T}{N}\mathbf{Q}\mathbf{a}_j = \rho_j\mathbf{Q}^T\mathbf{Q}\mathbf{a}_j. \tag{2.7}$$

De onde é possível fazer a substituição $\mathbf{K} = \mathbf{Q}^T\mathbf{Q}$. Com isso, obtém-se uma nova equação e sua simplificação, como apresentado por Schölkopf em [7]:

$$\begin{aligned} \mathbf{K^2}\mathbf{a}_j &= N\rho_j\mathbf{K}\mathbf{a}_j, \\ \mathbf{K}\mathbf{a}_j &= N\rho_j\mathbf{a}_j. \end{aligned} \tag{2.8}$$

Para calcular o produto interno, no espaço das *features*, $\mathbf{q}_i^T\mathbf{q}_j$, utiliza-se uma função Kernel sobre os vetores de entrada $\mathbf{x}_i$ e $\mathbf{x}_j$. Uma função Kernel é um método eficiente de computar produtos internos no espaço $\mathcal{F}$, através da reformulação do mapeamento em função dos vetores no espaço original. Isto é, sendo um mapeamento arbitrário $\Phi : \mathbb{R}^M \to \mathbb{R}^I$, o produto $\Phi(\mathbf{x}).\Phi(\mathbf{y})$ passa a ser definido por uma função $k(\mathbf{x}, \mathbf{y})$. Alguns exemplos de funções Kernel estão na Tabela 2.3.

Tabela 2.3: Alguns exemplos de funções Kernel.

Tipo de Kernel	Função Kernel
Polinomial	$k(\mathbf{x}, \mathbf{y}) = (\alpha\mathbf{x}^T\mathbf{y} + c)^d$
Radial-basis-function (RBF)	$k(\mathbf{x}, \mathbf{y}) = \exp\left(\frac{\|\|\mathbf{x}-\mathbf{y}\|\|^2}{\gamma}\right)$
Sigmoide	$k(\mathbf{x}, \mathbf{y}) = \tanh(\alpha\mathbf{x}^T\mathbf{y} + c)$

É possível calcular os N vetores $\mathbf{a}_j$ como autovetores da Equação (2.8) e normalizar seus valores de modo que $||\mathbf{a}_j|| = 1$. Assim como na PCA linear, podemos aproveitar apenas as componentes que desejamos, selecionando vetores $\mathbf{a}_j$. Alternativamente, pode-se utilizar apenas parte $N' < N$ dos vetores $\mathbf{x}_n$ da base de dados, que serão denominados $\mathbf{l}_j$.

A partir de então, para se obter obter as projeções sobre as componentes calculadas, a função Kernel é novamente utilizada para calcular os produtos internos no espaço das *features*. Para cada vetor da entrada, as projeções ao longo de cada componente principal compõem um vetor $\mathbf{z}_n$, dado por:

$$\mathbf{z}_n = \begin{bmatrix} k(\mathbf{l}_1, \mathbf{x}_n) \\ \vdots \\ k(\mathbf{l}_{N'}, \mathbf{x}_n) \end{bmatrix}, \tag{2.9}$$

a partir do qual podemos encontrar as N' projeções $f_{n,i}$, onde $i = 1, \ldots, N'$. Para isso, o vetor $\mathbf{z}_n$ é mulitplicado pelo previamente calculado vetor $\mathbf{a}_j$.

$$f_{n,i} = \mathbf{a}_i^T \mathbf{z}_n \rightarrow \mathbf{f}_n = \mathbf{A}^T \mathbf{z}_n. \tag{2.10}$$

Decompondo $\mathbf{z}_n$ na primeira etapa da Equação (2.10) como $\mathbf{Q}^T\mathbf{q}_n$, temos $\mathbf{a}_j^T\mathbf{Q}^T\mathbf{q}_n$. Da consideração feita em relação aos autovetores da matriz $\mathbf{C}'_X$, obtém-se $\mathbf{a}_j^T\mathbf{Q}^T = \mathbf{v}_j^T$, de onde a equação das *features* para Kernel PCA pode ser relacionda com a Equação (2.5) da PCA linear.

Em [7], é mencionado que o vetor $\mathbf{z}_n$ precisa passar por uma etapa de centralização (*zero centering*) antes que se possa obter o valor das *features*, de modo que:

$$\mathbf{K}_c = \mathbf{K} - \mathbf{1}_N\mathbf{K} - \mathbf{K}\mathbf{1}_N + \mathbf{1}_N\mathbf{K}\mathbf{1}_N \tag{2.11}$$

e

$$\mathbf{z}_c = (\mathbf{I} - \mathbf{1}_N)\mathbf{z} - \mathbf{K}^T\mathbf{1}_N + \mathbf{1}_N\mathbf{K}^T\mathbf{1}_N, \tag{2.12}$$

onde $\mathbf{1}_N$ tem todos os elementos com valor $1/N$. Isto resulta em:

$$\mathbf{f}_n = \mathbf{A}^T \mathbf{z}_{cn}. \tag{2.13}$$

Esta etapa pode se tornar implícita em relação a $\mathbf{z}_n$ caso a incorporemos à matriz de pesos e adicionemos um *bias* na Equação (2.10).

$$\mathbf{f}_n = \mathbf{W}\mathbf{z}_n + \mathbf{b}. \tag{2.14}$$

2.6.3 Quantizador Vetorial Baseado em Kernel PCA

Um quantizador vetorial baseado em Kernel PCA utiliza os conceitos apresentados na Seção 2.6.2 para explorar a uma correlação entre as dimensões.

Para quantizar dados utilizando esta técnica, um quantizador é treinado utilizando uma base de dados relativamente pequena para gerar a matriz $\mathbf{K}$ e seus autovetores $\mathbf{a}_j$, de acordo com a distribuição da entrada que se deseja quantizar. A partir daí, os vetores de entrada são projetados sobre a base de treino, diretamente no espaço $\mathcal{F}$, como demonstrado na Equação (2.9), e os valores das *features* são calculados segundo a Equação (2.14). De modo equivalente à PCA linear, as *features* representam a energia da densidade de probabilidade de maneira melhor que o espaço original. O passo seguinte é selecionar as *features* desejadas, que são, normalmente, as de maior energia. Aplica-se, então, uma SQ sobre elas, comparando os valores das projeções com os limiares do quantizador.

A quantidade de limiares destinados a cada *feature* depende da taxa R do quantizador e a quantidade de *features* depende da aplicação e da distribuição utilizada. Tendo seus valores quantizados e sabendo qual valor $\mathbf{x}_n$ está associado a cada projeção, pode-se definir as células e os centroides da quantização.

Capítulo 3

Metodologia

Este capítulo apresentará os detalhes referentes às formas de aplicação da teoria para buscarmos os resultados esperados para este trabalho. Mostraremos os detalhes de programação, apresentando os algoritmos implementados nas simulações, detalhes sobre a base de dados utilizada nos quantizadores e outras características sobre notação e considerações importantes.

3.1 Base de Dados

Para realizar os testes dos quantizadores, consideramos uma base de dados gerada a partir de uma fonte de amostras com uma densidade de probabilidade exponencial. Esta distribuição tem média unitária e a fonte, que gera amostras aleatórias seguindo tal distribuição, é fornecida pelo próprio pacote de cálculo numérico utilizado. A função densidade de probabilidade (pdf) em questão pode ser escrita como:

$$f(x) = \begin{cases} e^{-x} & \text{se } x \geq 0, \\ 0 & \text{se } x < 0. \end{cases} \tag{3.1}$$

As amostras para este estudo possuem $M = 2$ dimensões, como na Figura 3.1. O número de amostras utilizado varia conforme mudam os parâmetros das simulações. Quando desejamos definir partições para os quantizadores, o ideal é fazermos o número de amostras muito maior que o número de células possível. Por exemplo, para uma quantização que gere até 32 células, uma quantidade de amostras para tal caso está em torno de 3000 amostras.

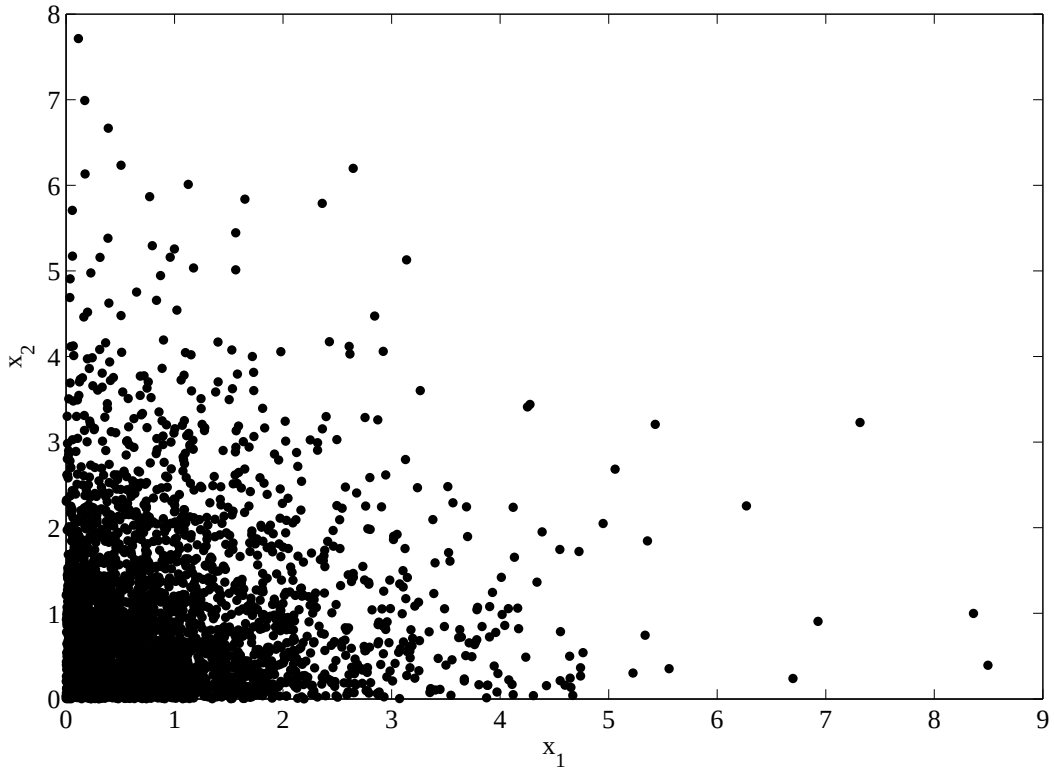


Figura 3.1: Amostras de uma distribuição exponencial em duas dimensões, como utilizadas nas simulações.

3.2 Algoritmos de Projeto

Foi adotada uma ordem de simulação para os possíveis algoritmos de quantização, de forma que os resultados de uma rodada de projeto pudessem ajudar nas rodadas seguintes, conforme será visto. Isto é feito considerando que inicializações diferentes geram resultados diferentes para cada algoritmo, o que restringe sua otimização a mínimos locais. Tais algoritmos serão apresentados na ordem em que foram executados e o motivo destas escolhas será esclarecido conforme cada um é explicado.

3.2.1 Quantização Escalar

O algoritmo de quantização escalar foi escolhido como o primeiro a ser simulado entre os algoritmos tradicionais de quantização. Isto porque não há inicialização de um dicionário aleatório. Após a seleção dos vetores de entrada, como amostras de uma distribuição exponencial, cada uma das dimensões é tratada separadamente e é considerada independente das outras.

Sobre cada uma das dimensões será aplicada uma busca pela posição de limiares ótimos no sentido de minimizar a distorção global do quantizador. É feita uma varredura para o primeiro limiar introduzido, de forma que possamos analisar o desempenho do quantizador em todas as possíveis posições desse limiar. Esta varredura posiciona o limiar entre todos os valores que aquela dimensão pode assumir e, para cada posição, calcula os centroides gerados a partir daquele limiar e, em seguida, a distorção proveniente do uso destes centroides. Acrescentando mais um bit na mesma dimensão, adicionamos dois novos limiares, que serão otimizados de forma semelhante ao primeiro. Porém, o limiar já calculado é mantido fixo e a varredura é feita a apenas no segundo limiar e, depois, com o segundo fixo, no terceiro. Também é possível acrescentar um bit ao longo da dimensão que ainda não passou por quantização. Neste caso, há, inicialmente, somente um limiar novo.

No cálculo dos limiares, para contornar a limitação que impomos ao fazer que os limiares anteriores sejam fixos, aplicamos o algoritmo de *Simulated Annealing* tendo como estado inicial o conjunto de limiares calculados no processo explicado.

1. Definir o estado inicial $s = (limiar_1,\ limiar_2,\ \dots)$ do algoritmo e o parâmetro $e = D + \lambda H$ a ser avaliado e otimizado.

2. Estabelecer uma *temperatura* inicial T e um *fator de resfriamento* c.

3. Loop de otimização:

    - Repetir enquanto $T \geq Tmin$ ou por tantas vezes quanto desejado;
    - Recalcular temperatura pós-refriamento: $T = c * T$;
    - Criar novo estado a partir de vizinhos do estado inicial: $s' = vizinhos(s)$ e reavaliar o resultado, obtendo novo e';
    - Comparar e' com o parâmetro inicial e:
        - Se e' melhor que e, mudar de estado: $s \leftarrow s'$;
        - Se e' pior que e, mudar de estado quando uma variável aleatória com densidade uniforme $r < \exp\left(-(e' - e)/T\right)$;

Para calcular os centroides, usamos os valores das amostras de entrada localizados entre os limiares. Depois de calculados os centroides, o cálculo da distorção e entropia resultante é feito com as mesmas amostras utilizadas na definição das partições.

Este método para projeto da partição do SQ foi utilizado, ao invés de utilizarmos o algoritmo de Lloyd para uma dimensão, uma vez que o algoritmo de Lloyd não gera limiares explicitamente conhecidos para a partição do quantizador. Com o método empregado, é possível obter a informação sobre em qual célula um dado de entrada está localizado realizando comparações com os limiares do quantizador. Estas comparações podem ser implementadas com menor complexidade que os cálculos de distâncias euclidianas que seriam necessários caso o projeto fosse baseado no algoritmo de Lloyd.

Todo o processo é repetido para diversas inicializações diferentes, de modo a criarmos uma nuvem de dados e eliminar possíveis dependências da aleatoriedade da fonte de amostras. A cada iteração, armazenamos a posição ótima encontrada para os centroides e as condições em que foram encontradas, para que estas informações sejam aproveitadas em outros quantizadores, além de armazenarmos os dados de desempenho. Um diagrama de blocos (Figura 3.2) apresenta os passos do algoritmo para projeto do SQ.

3.2.2 Quantização Vetorial

Para a implementação e teste de VQ, foram empregadas algumas das idéias utilizadas para estudar a SQ. Foram mantidas as iterações para diversas inicializações diferentes e, da base de dados utilizada para treinarmos o quantizador, tiramos os vetores que são utilizados para medir o desempenho da otimização. No VQ, porém, o treino é realizado com o algoritmo LBG, isto é, diferente do treino feito no SQ, onde otimizamos diretamente os limiares das partições. O treino precisa ser inicializado com centroides, possivelmente aleatórios. Mas como já foram calculados centroides otimizados para SQ, aproveitamos estes dados para inicializar de forma mais eficiente o novo treino.

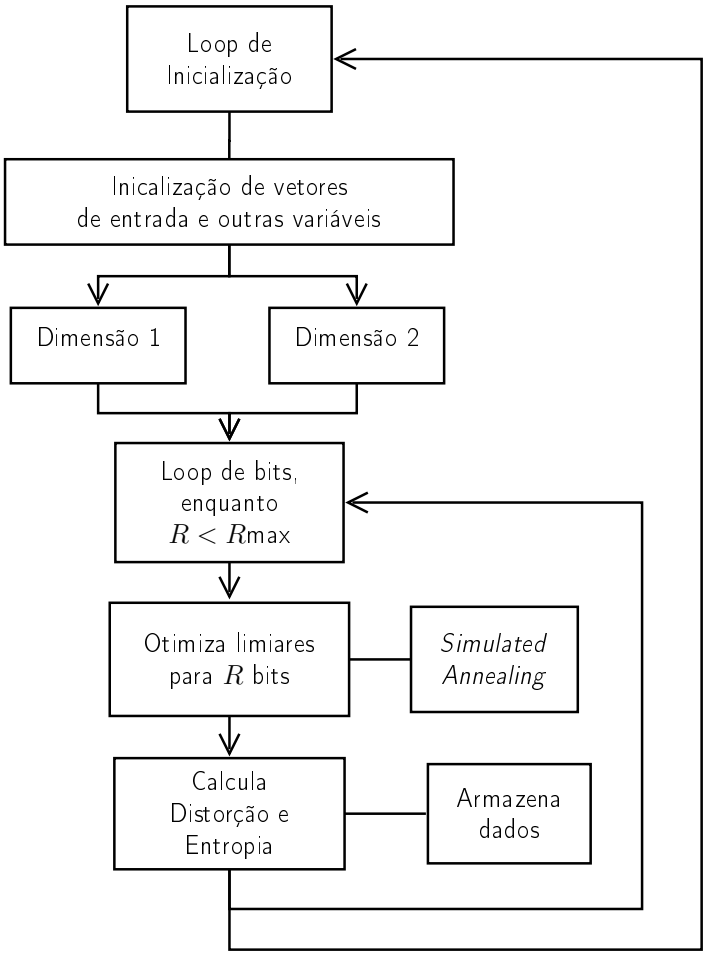


Figura 3.2: Diagrama de blocos do algoritmo utilizado para SQ.

Na simulação de VQ, fazemos:

1. Inicialização a partir dos dados da simulação com SQ;

2. Aplicação do algoritmo de LBG para otimizar a posição dos centroides;

3. Teste de distorção e entropia para centroides otimizados;

Novamente, dados além dos resultados de desempenho são armazenados, para uso em outros quantizadores.

Algumas considerações podem ser feitas a respeito da implementação do algoritmo de LBG. Para buscarmos eficiência de código, a distância de cada vetor de entrada para os centroides não é calculada individualmente. O vetor de entrada é replicado K vezes (sendo K o tamanho do dicionário) e, sobre a matriz resultante, calculamos as K distâncias euclidianas dos elementos do dicionário paralelamente. A partir

delas, identificamos o centroide mais próximo e o associamos seu índice ao vetor de entrada. A convergência do algoritmo é considerada válida quando a diferença entre a distorção calculada em um dos passos de ajuste (tanto das partições como do dicionário) e a distorção calculada no passo anterior é menor que um determinado valor pequeno, ϵ.

3.2.3 Quantização Vetorial com Restrição de Entropia

De forma semelhante à aplicada na passagem SQ para a VQ, aproveitaremos os resultados desta última para otimizar os testes de VQ com restrição de entropia. Portanto, a minimização do custo $J = D + \lambda H$ será realizada sobre os centroides que minizavam o custo $J = D$ da VQ.

Diferentemente do proposto em [8], onde se repete a minimização do custo para diferentes valores de λ a fim de formar a casca convexa (será explicada na Seção 3.4) , utilizamos uma abordagem diferente sobre o multiplicador de Lagrange. No treino do ECVQ, já dispomos de um dicionário gerado no VQ. Com isso, para cada inicialização diferente feita no VQ, obtivemos uma curva de distorção × entropia. A partir destas curvas, calculamos o λ para cada novo ponto que pretendemos otimizar sob as metas do ECVQ. O multiplicador de Lagrange será, como exibido nas Figuras 3.3 e 3.4, calculado da seguinte forma:

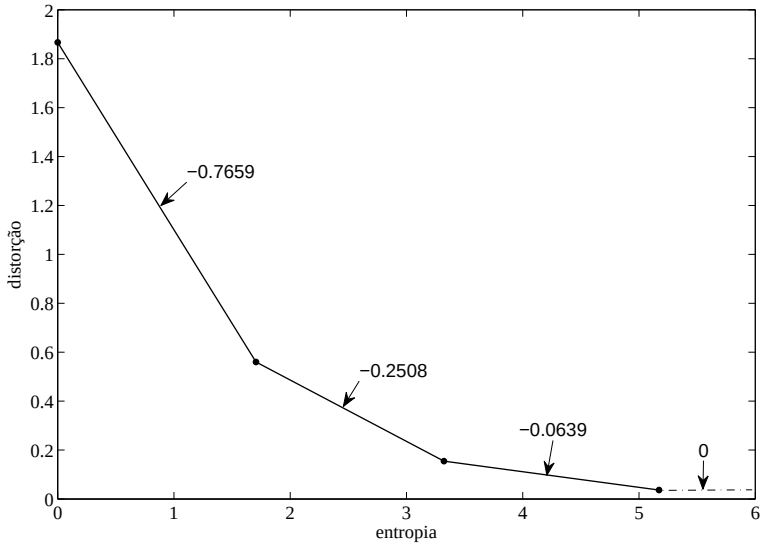


Figura 3.3: Casca convexa do VQ e inclinações entre os pontos.

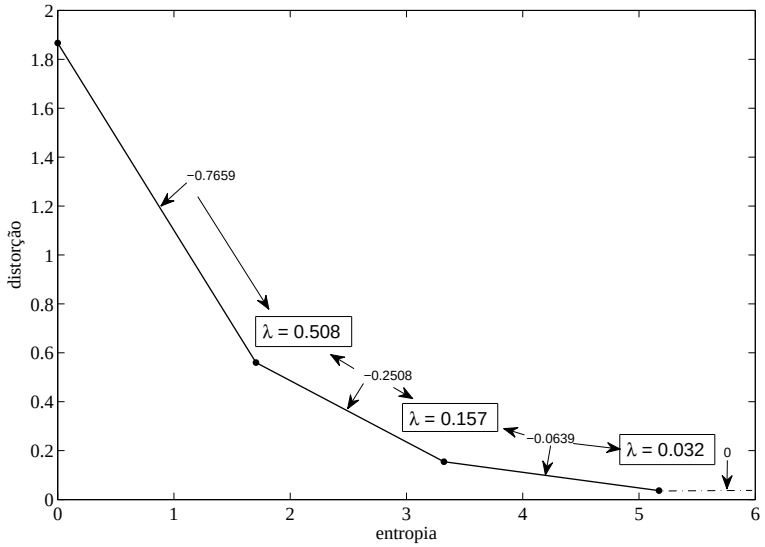


Figura 3.4: Cálculo do λ para cada ponto.

1. Otimização dos dados com VQ para determinada inicialização;

2. Calcular, na casca convexa da curva distorção $\times$ entropia, inclinações dos segmentos que conectam os pontos obtidos;

3. Para o último ponto, assumir que a inclinação à sua direita é zero.

4. Calcular o multiplicador de Lagrange λ para cada ponto, como sendo a média das inclinações adjacentes:

$$\lambda_i = -\frac{\left(\frac{D_{i+1}-D_i}{H_{i+1}-H_i} + \frac{D_i-D_{i-1}}{H_i-H_{i-1}}\right)}{2}. \tag{3.2}$$

O valor de H, no custo $J = D+\lambda H$, não foi estimado com base no resultado direto da codificação de Huffman. Ao invés disso, trabalhamos diretamente com fatores $-\log_2 p(k)$ e evitamos que células com probabilidades próximas, mas ainda assim diferentes, recebam o mesmo tratamento na minimização do custo. Quando uma célula tem probabilidade $p(k) = 0$, esta célula é eliminada e o projeto do quantizador prossegue com uma quantidade menor de células. Caso ocorra um vetor de entrada presente na região onde havia originalmente uma célula que foi eliminada, a distorção associada a este vetor será maior do que a distorção caso esta célula fosse mantida.

3.3 Quantização Vetorial baseada em Funções Kernel

O quantizador baseado em funções Kernel é abordado de forma independente dos quantizadores anteriores. Sua inicialização não leva em conta um dicionário inicial, como acontece nos dois VQs apresentados. Os outros dados não são relevantes para a determinação deste quantizador.

A função Kernel é aplicada sobre dos vetores que seguem a distribuição dada, gerando uma matriz de produtos internos $\mathbf{K}$, cujos autovetores correspondem aos vetores $\mathbf{a}_j$ adaptados para VQ na Seção 2.6.3. Aplicando novamente a função Kernel conforme as Equações 2.9 e 2.13, projetamos vetores de treino sobre as componentes principais, já no espaço das *features* e obtemos, com isso, o vetor $\mathbf{z}(n)$. O produto de $\mathbf{z}(n)$ por $\mathbf{a}_j$ resulta nas *features*, das quais são selecionadas aquelas com as quais queremos trabalhar. Para um determinado mapeamento, são utilizados diversos conjuntos de vetores $\mathbf{l}_j$, gerando diferentes otimizações de limiares, de onde são aproveitados os melhores resultados.

A estrutura da simulação para a quantização baseada em Kernel é focada na busca por uma melhor configuração do quantizador. Arbitrariamente, decidimos que o projeto do VQ com funções Kernel ficaria restrito às três *features* de maior energia. É necessário, para um dado conjunto de vetores e uma taxa de bits máxima, que os bits sejam alocados entre estas três dimensões da melhor forma possível.

A decisão de como será feita a alocação dos bits é tomada após uma análise das possibilidades. Inicialmente, dispondo de apenas um bit, este é alocado em cada *feature* de uma vez e executamos o processo de otimização de limiar (Seção 3.2.1) para cada uma das três possibilidades. A alocação de bits é representada por um vetor com três elementos, cada um correspondendo ao número de bits de cada *feature*. Guardamos os vetores que geram os dois melhores resultados. Na segunda rodada, um segundo bit é alocado sobre os vetores dos resultados obtidos na primeira etapa, da mesma forma como feito inicialmente. Novamente, são guardados os dois melhores resultados. Este processo se repete até que seja alcançado o número

máximo de bits para o quantizador. Na Tabela 3.1, este processo é exemplificado, de modo que cada rodada representa uma quantidade total de bits para o projeto do quantizador. Cada linha representa um projeto para aquela quantidade de bits, com um novo bit sendo atribuido a cada uma das *features*.

Tabela 3.1: Exemplo de alocação de bits, indicando o número de bits atribuido a cada *feature* f. Sequências marcadas com negrito indicam melhores projetos e cada rodada representa uma quantidade total de bits.

Alocação de bits

Rodada 1			Rodada 2			Rodada 3		
f_1	f_2	f_3	f_1	f_2	f_3	f_1	f_2	f_3
1	**0**	**0**	**2**	**0**	**0**	**3**	**0**	**0**
0	**1**	**0**	1	1	0	2	1	0
0	0	1	**1**	**0**	**1**	2	0	1
-	-	-	1	1	0	2	0	1
-	-	-	0	2	0	**1**	**1**	**1**
-	-	-	0	1	1	1	0	2

Dentro de cada uma das rodadas para decisão da alocação de bits, projetamos um VQ baseado em funções kernel. Para cada *feature*, estabelecemos limiares, tantos quantos a alocação de bits permitir àquela *feature*. O posicionamento destes limiares é feito, primeiramente, de maneira indivual. Começando com apenas um limiar, faz-se a varredura de modo a encontrarmos sua melhor posição entre os valores daquela *feature*. Depois, fixa-se este limiar e se repete o processo para otimizar a posição do segundo limiar. Este processo se repete até que o número de bits alocado a tal *feature* seja alcançado. Após este processo, aplica-se o algoritmo de *Simulated Annealing* sobre os limiares obtidos.

De forma semelhante à explicada na Seção 3.2.1, este método foi utilizado de forma a reduzir a complexidade do codificador α durante a quantização dos dados de entrada.

Tabela 3.2: Implementação do cálculo da partição do VQ baseado em funções Kernel.

Para manipular variáveis associadas às *features* do quantizador foram utilizados vetores binários com comprimento igual à quantidade de vetores de entrada. Quando uma determinada *feature* passa por um SQ, são geradas células naquela *feature*, de acordo com a quantidade de limiares criados. Cada uma destas células é representada por um vetor binário como o citado.

Cada um destes vetores é inicializado com zeros. Como cada um dos elementos destes vetores é associado a um dos vetores de entrada, aqueles elementos cujos vetores de entrada se localizam naquela célula recebem o valor 1. Isto é, dada uma célula em uma *feature*, existem vetores de entrada do espaço original que estão entre as superfícies de corte associadas a tal célula e são representados pelo valor 1 nos vetor binário desta célula.

Para se obter o conteúdo de uma célula formada pelo conjunto de superfícies de corte geradas pelos limiares de todas as *features*, é utilizada uma operação lógica AND entre os vetores binários entre as diferentes *features*. Ou seja, um dos vetores binários de uma *feature* é combinado com um dos vetores da segunda e um dos vetores da terceira *feature*.

$$[1\,0\,1\,1\,0] \text{ and } [0\,0\,1\,1\,0] \text{ and } [0\,1\,1\,1\,1] = [0\,0\,1\,1\,0] \tag{3.3}$$

A Equação (3.3) mostra um exemplo simplificado para 5 vetores de entrada. Cada um dos vetores à esquerda da igualdade representa uma das células de uma das três *featues*. A combinação destes três resulta em uma célula do espaço original, que contém, como representado pelo vetor resultante, apenas o terceiro e o quarto vetores de entrada.

Cada limiar criado nas *features* terá uma superfície de corte correspondente no espaço original. O conjunto destas superfícies de corte, combinando as superfícies associadas aos limiares das três *features* diferentes, define a partição do VQ baseado em funções Kernel.

Com os vetores de entrada pertencentes a cada uma das células resultantes, encontramos seus centroides (o vetor médio dos vetores de cada célula) e calculamos a distorção e entropia resultantes. Estes resultados são armazenados e também utilizados para se obter a melhor alocação de bits, como descrito anteriormente. Ao se alcançar a quantidade de bits estabelecida, o quantizador para o determinado mapeamento e conjunto de vetores $\mathbf{l}_j$ está parcialmente otimizado. Após todos os conjuntos de vetores $\mathbf{l}_j$ serem utilizados, os melhores pontos são extraídos do conjunto de resultados gerados e otimizados em função do custo $J = D + \lambda H$ utilizando o algoritmo de *Simulated Annealing*.

3.4 Casca Convexa

Para uma determinada técnica de quantização, são realizados diversos projetos de quantizador, já que o projeto e os resultados da quantização são consideravalmente dependentes dos vetores de entrada. Uma vez projetada a partição de determinado projeto, vetores de entrada da mesma fonte de dados do projeto são utilizados para calcular o desempenho do quantizador. Este cálculo resulta em diversos vetores formados pelos pares (entropia, distorção), um para cada projeto, que, quando representados graficamente, geram uma distribuição de pontos como apresentada na Figura 3.5.

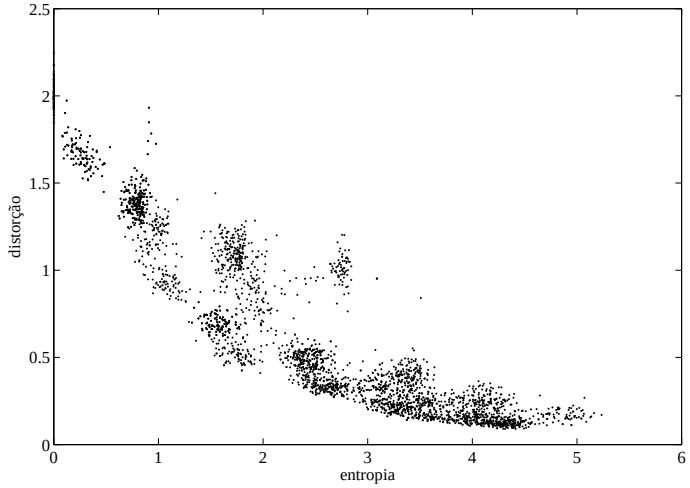


Figura 3.5: Possíveis pontos (H, D) obtidos ao longo do projeto de quantizadores.

Foi desenvolvida uma função que seleciona apenas os melhores pontos dessa distribuição. Estes pontos são aqueles pertencentes à casca convexa inferior da distribuição. Isto é, são os pontos selecionados de modo que a curva por eles formada nunca apresente uma concavidade para baixo e que não haja pontos abaixo da curva. Esta função é aplicada a todos os quantizadores projetados e, assim, os seus melhores resultados podem ser comparados.

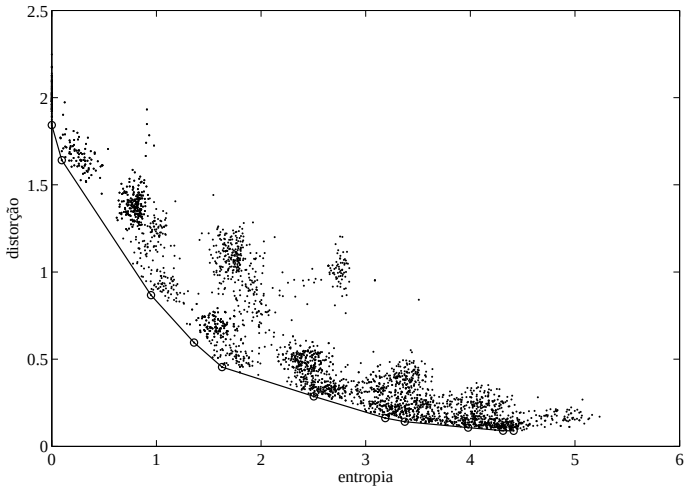


Figura 3.6: Casca convexa da distribuição de pontos (H, D).

3.5 Cálculo de Complexidade

Para realizar o estudo de complexidade dos sistemas abordados, foram consideradas unidades de complexidade baseadas na implementação de *hardware*, em função do número de transistores utilizados para um circuito analógico que implemente determinado processo. Esta abordagem pode ser utilizada também como base para um estudo teórico da complexidade de sistemas.

Os principais blocos utilizados nos quantizadores estudados são apresentados a seguir, assumindo que todos podem ser implementados com transistores CMOS analógicos. Mais detalhes sobre as implementações podem ser encontrados em [9].

3.5.1 Comparadores

Comparadores são utilizados na SQ para comparar os vetores de entrada com os limiares do quantizador. Como o número de limiares de um SQ é $2^R - 1$, a complexidade de um bloco comparador será de $(2^R - 1)T_{\text{comp}}$, onde T_{comp} é o número de unidades necessárias para realizar a comparação entre dois escalares e retornar uma corrente positiva caso o primeiro escalar seja maior ou negativa caso o primeiro escalar seja menor que o segundo. Será assumido que a implementação com CMOS pode ser realizada com $T_{\text{comp}} = 1$ unidade.

3.5.2 Função OU Exclusivo

Para se obter a palavra binária correspondente à celula resultante de um SQ, as saídas dos comparadores passam por uma breve lógica combinacional implementada com portas OU Exclusivo (XOR, de *Exclusive Or*). A quantidade de operações XOR realizadas em um SQ de R bits é $2^R - R - 1$, portanto, a complexidade adicionada por estas operações é $(2^R - R - 1)T_{\text{xor}}$, onde será assumido que $T_{\text{xor}} = 4$.

3.5.3 Produto Interno

Para realizar o produto interno de um vetor $\mathbf{x}(n)$ de comprimento M por um vetor constante $\mathbf{y}(k)$, são utilizadas M unidades para as multiplicações e um *current conveyor* ("transportador de corrente") [10] para realizar a soma das correntes resultantes. Uma possível implementação para o *current conveyor* utiliza uma quantidade $T_{\text{cc}} = 2$ de unidades. Portanto, a complexidade para um bloco que realiza o produto interno será de $M + T_{\text{cc}}$.

3.5.4 Função Mínimo

A função utilizada para selecionar o vetor do dicionário que apresenta a menor distância para o vetor de entrada é implementada com um circuito *winner-takes-all*, apresentado em [11]. A complexidade deste circuito é $T_{\text{wta}} = 2$ unidades para cada vetor comparado. Para um dicionário de tamanho 2^R, a complexidade estimada é de $2^R T_{\text{wta}}$.

3.5.5 Função Quadrado

Para se obter, a partir de um escalar $x(n)$, o valor de $x^2(n)$, utiliza-se a função quadrado com $T_{\text{quad}} = 3$ unidades na implementação em CMOS [12].

3.5.6 Multiplicador de Gilbert

O multiplicador de Gilbert é necessário para realizar multiplicações entre dois valores em casos onde nenhum dos dois seja previamente conhecido. Será assumido que sua complexidade $T_{\text{gm}} = 9$ unidades, mas este bloco básico não será utilizado.

3.5.7 Função Tangente Hiperbólica

A função tangente hiperbólica, utilizada para a computação de um dos possíveis mapeamentos Kernel, conforme a Tabela 2.3, é implementada com o mesmo circuito utilizado para a comparação. Esta função retorna o valor da operação $\tanh(q(n))$, onde $q(n)$ é um escalar. Seu custo, portanto, é estimado $T_{\text{tanh}} = 4$ unidades.

3.5.8 Função Exponencial

A função exponencial é utilizada na computação da função Kernel RBF, como apresentado na Tabela 2.3. Sua complexidade se aproxima à complexidade de uma multiplicação. Será assumido que $T_{\text{exp}} = 1$ unidade.

Capítulo 4

Resultados

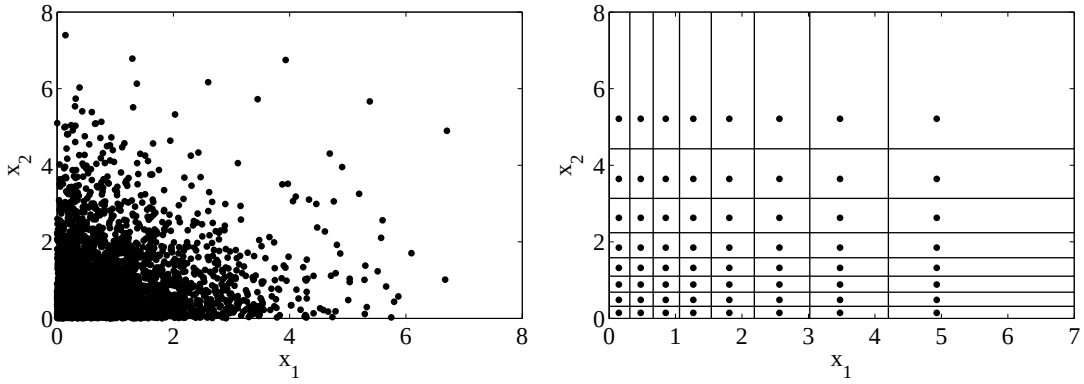
Neste capítulo, são apresentados resultados simulados para os quantizadores estudados neste trabalho. Inicialmente, são vistas as características das partições de cada quantizador, mostrando como cada um deles atua sobre os dados de entrada. A seguir são apresentados os resultados para o desempenho das técnicas abordadas e uma comparação a respeito da complexidade de suas implementações.

4.1 Definição de Partições

Uma forma de observar o efeito dos diferentes métodos de quantização sobre os dados de entrada é observar como são definidas as partições construídas por cada quantizador. Isto é, observar como o quantizador divide os vetores de entrada nas células que serão codificadas. No caso de duas dimensões, é possível analisar este efeito graficamente, exibindo as linhas que definem os limites de cada célula.

4.1.1 Quantização Escalar

Este resultado esclarece a relação de independência que a SQ impõe sobre as dimensões dos vetores de entrada. Cada uma das partições geradas é associada a um único valor de uma das dimensões, ou seja, as linhas que definem as partições são retas perpendiculares ao eixo da dimensão a que se referem, como na Figura 4.1.



(a) (b)

Figura 4.1: (a) Amostras da distribuição exponencial usadas no projeto do SQ; (b) partição e centroides gerados pelo SQ.

4.1.2 Quantização Vetorial e Quantização Vetorial com Restrição de Entropia

Tanto no caso do VQ como no caso do ECVQ, o quantizador analisa todas as dimensões dos vetores de entrada de forma conjunta. Assim, as partições não se apresentam de forma tão comportada como no caso do SQ. As células, para VQ e ECVQ, se apresentam como poliedros convexos, de forma que se assemelham a favos de mel, vistos na Figura 4.2. Buscam, localmente, agrupar os vetores de entrada de forma a reduzir a distorção total e a entropia (no caso do ECVQ).

4.1.3 Quantização Vetorial baseada em Funções Kernel

Quando utilizamos a análise de componentes principais com funções Kernel, a formação das partições se dá de forma mais abstrata do que as vistas nos métodos de SQ, VQ e ECVQ. Uma vez que os vetores tenham sido mapeados para o espaço das *features*, os limiares são aplicados aos valores de cada *feature*. Estas superfícies de separação, lineares no espaço das *features*, mostram-se não-lineares quando exibidas no espaço original. Isto pode ser observado ao notarmos que um limiar nos valores de uma *feature*, quando representado no espaço original, aparecerá como uma curva de nível. Esta curva de nível representa a separação entre todos os vetores de entrada que, depois de mapeados, são separados por aquele limiar.

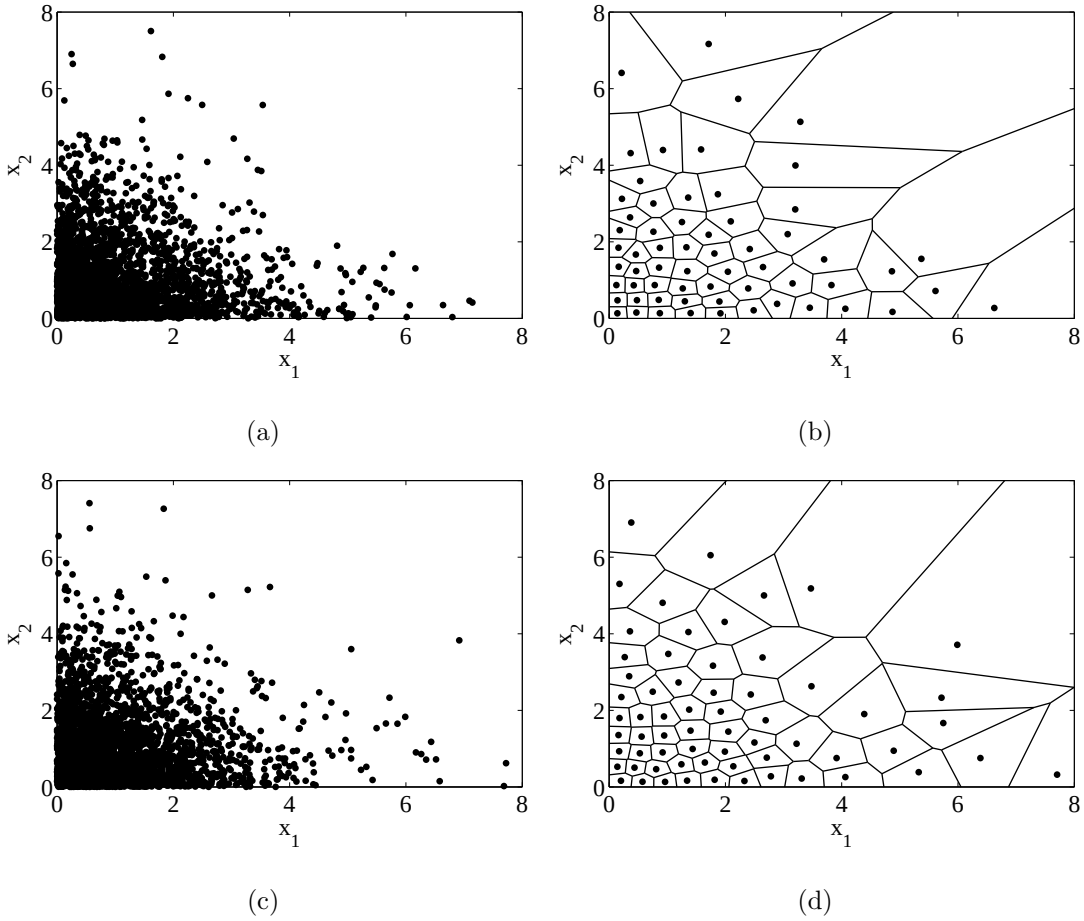


Figura 4.2: (a) (c) Amostras das distribuições exponenciais nas simulações do VQ e ECVQ; (b) partição e centroides gerados pelo VQ; (d) diagrama de Voronoi sobre os centroides gerados por ECVQ. A partição, nesta representação, não leva em consideração a entropia.

A partição efetiva do VQ baseado em funções Kernel é dada pela interseção da partição gerada nas *features* escolhidas. Nesta etapa, algumas regiões formadas pelas diversas curvas ficarão vazias e, portanto, podem ser descartadas. Isto faz com que o número final de centroides não seja exatamente igual ao número de regiões de interseção original. Na simulação cujo resultado é mostrado na Figura 4.3, foram utilizadas 3 *features* e um total de 6 bits, distribuídos da seguinte forma: 3 bits para a primeira *feature*, 2 bits para a segunda *feature*, 1 bit para a terceira *feature*. Por isso são diferentes os números de curvas exibidas em cada *feature*. Seria esperado, portanto, na condição máxima para 6 bits, que fossem obtidas 64 regiões. Foram obtidas, neste caso, 39 regiões, levando em conta, também, que a não-linearidade gera um número menor de interseções pelo formato das curvas resultantes.

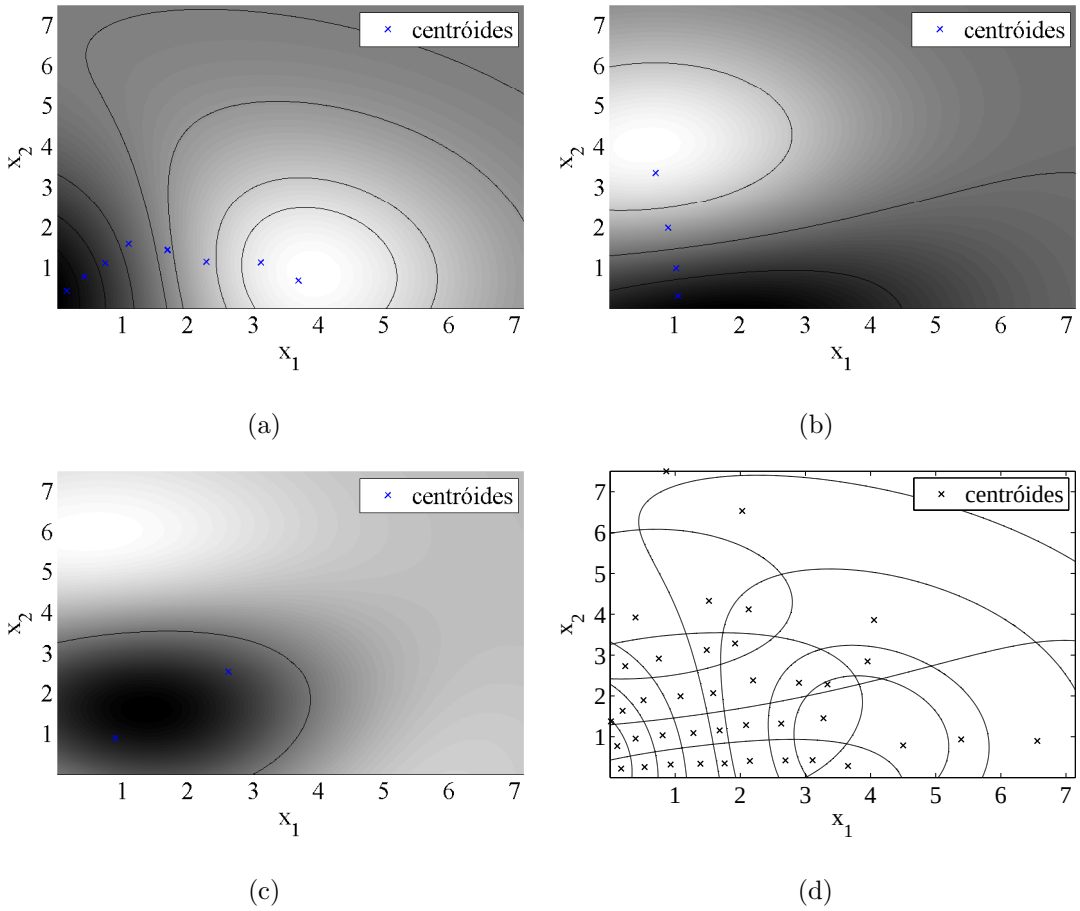
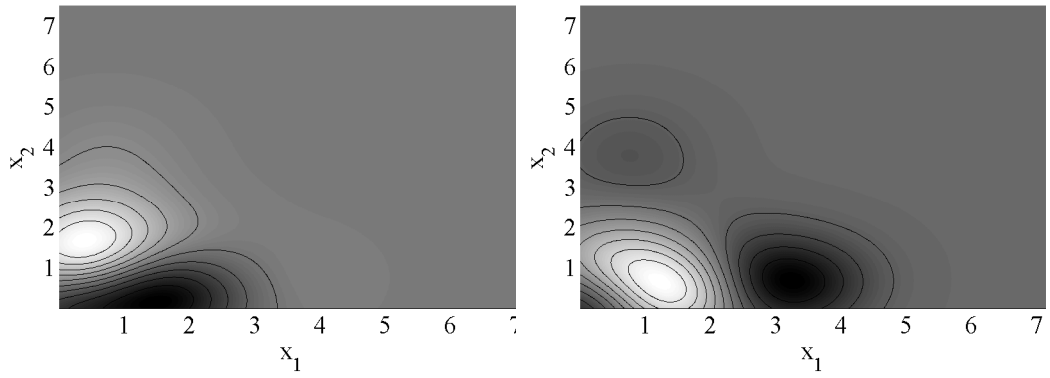


Figura 4.3: Kernel RBF: (a) (b) (c) partição das *features* 1, 2 e 3. Nas regiões mais claras, as features têm maior valor; (d) centroides e interseções das *features*.

Nas Figuras 4.3(a) a 4.3(c), o gradiente de transição das cores, do preto ao branco, revela as direções dos dados que possuem energia mais significativa. As Figuras 4.3(a) e 4.3(b) representam a maior variância provocada pelas duas dimensões, x_1 e x_2, onde ambas são distribuições exponenciais. A Figura 4.3(c) mostra a variância gerada conjuntamente pelas duas dimensões da entrada, sempre de forma não-linear. Além disso, alterando os parâmetros da função Kernel, modificamos o nível de seletividade e não-linearidade do quantizador, como na Figura 4.4.

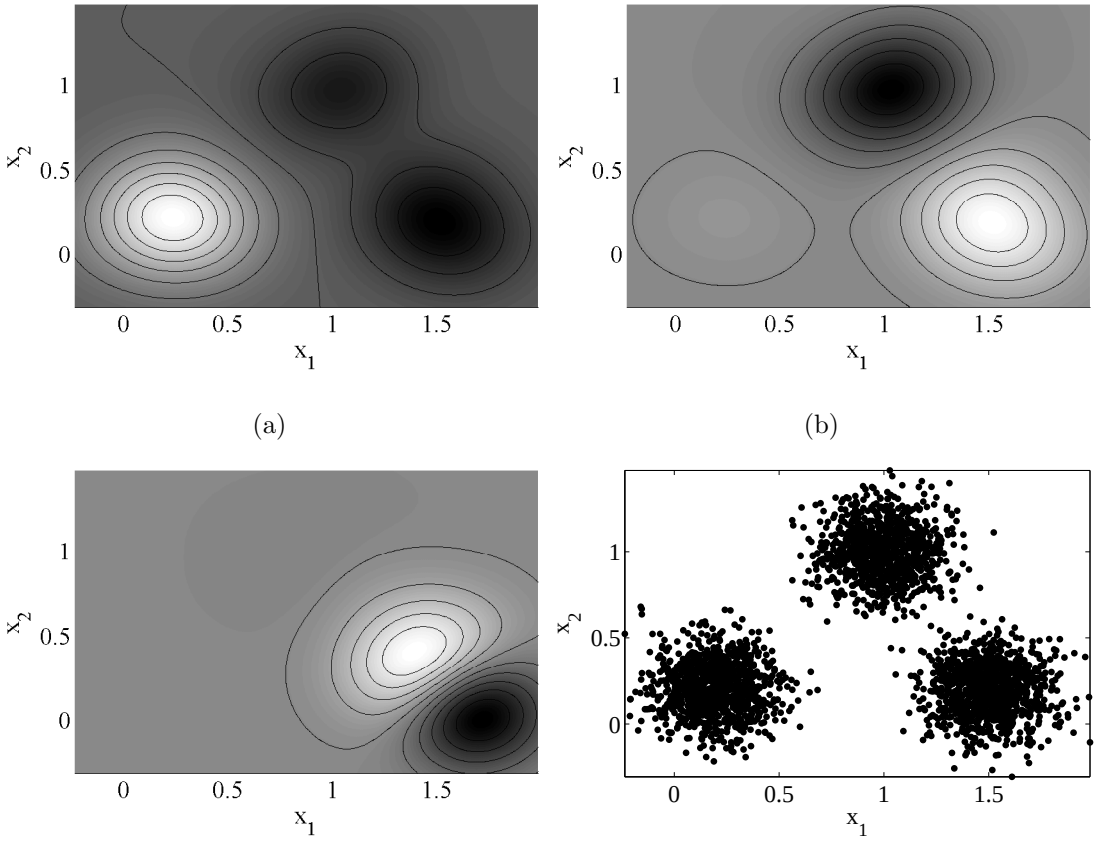
Quando aplicada a certas distribuições diferentes, esta característica de componentes principais se confunde com a análise de *clusters* ou *agrupamentos*, isto é, a classificação dos vetores de entrada em diferentes grupos, de acordo com semelhanças em suas probabilidades (Figura 4.5).



(a) (b)

Figura 4.4: Análise com função Kernel RBF com γ dez vezes menor; duas *features*.

Embora pareça interessante este reconhecimento de *clusters* dentro de um conjunto de vetores de entrada, este efeito será desejado apenas em outras aplicações, como reconhecimento de padrões ou *data mining*, e não em quantizadores, que é o caso do estudo deste projeto.



(a) (b)

(c) (d)

Figura 4.5: Análise com função Kernel RBF: (a) (b) (c) *features*; (d) distribuição em *clusters*.

No processo da quantização, a redução do valor γ em um VQ com função Kernel RBF faz com que as curvas de nível se concentrem em regiões muito específicas do espaço de dados. Isto é, os valores das *features* são mais sensíveis (pequenas variações dos dados de entrada causam grandes variações no espaço $\mathcal{F}$) onde há maior concentração de dados, como se pode ver nas Figuras 4.4 e 4.5. Com isso, fica mais difícil estabelecer os limiares que otimizem o funcionamento do quantizador.

As curvas de nível para um VQ baseado em função Kernel tangente hiperbólica são mostrados na Figura 4.6.

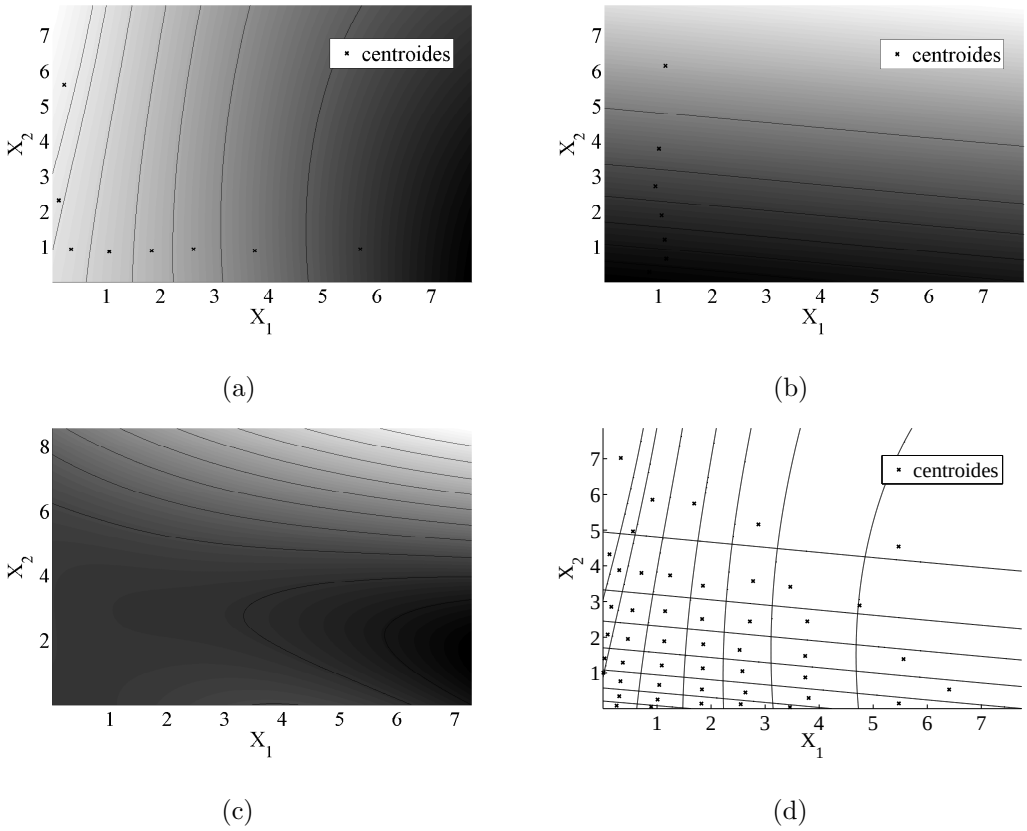


Figura 4.6: Análise com função Kernel tangente hiperbólica: (a) (b) (c) partição das *features* 1, 2 e 3. Nas regiões mais claras, *features* têm maior valor; (d) centroides e interseções das *features*.

Neste caso, apresentado na Figura 4.6, a otimização dos limiares levou a uma situação onde apenas as duas *features* de maior energia foram utilizadas. Isto é, obteve-se desempenho superior alocando todos os bits em apenas duas das três possíveis *features*. Da mesma forma como no caso da análise com funções Kernel RBF, a partição final é formada a partir da partição obtida em cada *feature*.

4.2 Desempenho

Neste campo de aplicação, o que desejamos realmente é que os quantizadores apresentem o resultado mais fiel possível aos dados de entrada. Ou seja, é necessário analisar a perda de informação que ocorre no processo de compressão. Como forma de quantificar esta perda, calculamos a distorção gerada pelos diferentes quantizadores. Além de considerarmos apenas o quanto um método de quantização consegue comprimir os dados, é importante considerar, juntamente, qual a eficiência com relação à taxa de bits utilizada no quantizador quando este realiza a compressão. Por exemplo, queremos dados que digam quantas vezes menor é a distorção para cada bit adicionado no quantizador.

Os gráficos a sequir mostram, portanto, um eixo referente à relação sinal-ruído de quantização (SQNR, de *signal-to-quantization-noise ratio*), baseada na casca convexa obtida para uma técnica de quantização e um eixo referente à entropia. O eixo referente à SQNR mostra o quanto a distorção naquele ponto é melhor, em escala logarítmica, do que a distorção para uma codificação de zero bits. O eixo referente à entropia apresenta o número mínimo de bits que é utilizado para alcançar determinado nível de distorção.

4.2.1 Quantizadores Tradicionais (SQ, VQ e ECVQ)

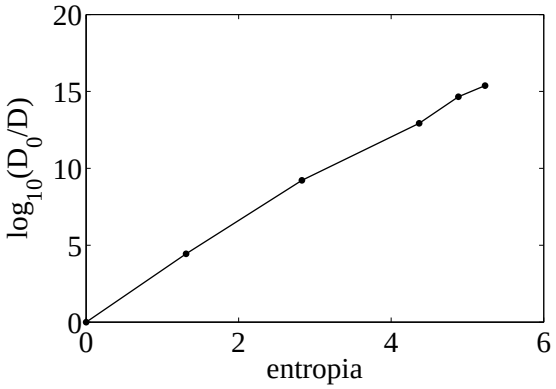
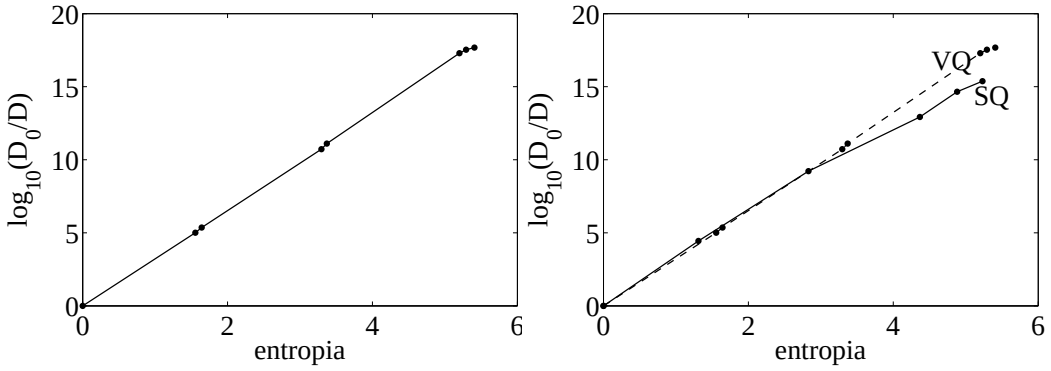


Figura 4.7: Casca convexa para 600 pontos (H, D) do SQ.

Dos métodos examinados, espera-se que o escalar, Figura 4.7, apresente desempenho inferior, uma vez que é o mais simples e apresenta análise apenas superficial do conjunto de dados.

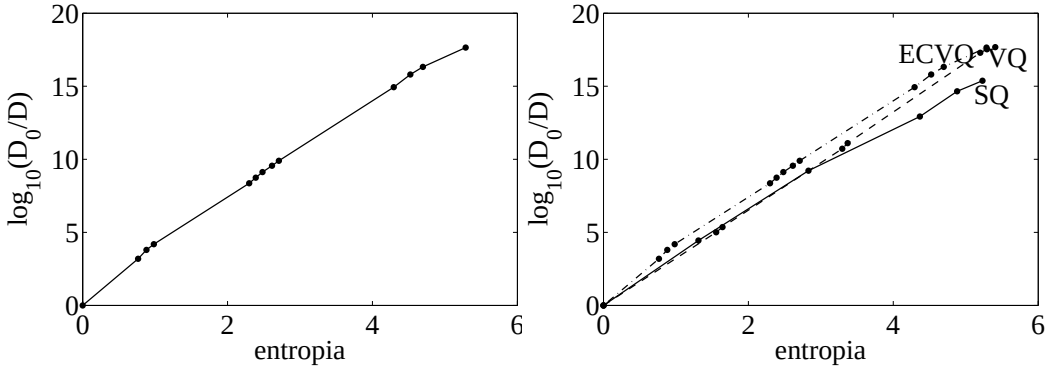


(a) (b)

Figura 4.8: (a) Casca convexa para 600 pontos (H, D) do VQ; (b) comparação com SQ.

A VQ apresenta maior complexidade do que a SQ. Inicializando um VQ com os centroides previamente calculados para o SQ, a VQ vai buscar realocar estes centroides de forma a encontrar, localmente, um ponto onde a distorção seja mínima.

O que se pode notar na Figura 4.8 é que, em baixas entropias, os gráficos de VQ e SQ são praticamente concorrentes, apresentando até uma ligeira superioridade para o SQ. Isto acontece porque o quantizador escalar é otimizado com o algoritmo de *Simulated Annealing* utilizando o fator da entropia na sua função custo. Enquanto isso, o VQ é otimizado em função unicamente de sua distorção. Deste modo, mesmo que o algoritmo de LBG seja inicializado com o dicionário do ponto onde o SQ é superior, pode acontecer de o resultado final perder em relaçâo à entropia.



(a) (b)

Figura 4.9: (a) Casca convexa para 600 pontos (H, D) do ECVQ; (b) comparação com SQ e VQ.

Novamente, utilizando os dados calculados previamente, agora do VQ, inicializamos um VQ com restrição de entropia. Agora, não apenas a distorção é considerada na otimização, mas também a entropia. Podemos esperar que o resultado do ECVQ seja, pelo menos, igual ao do VQ, visto que, para um determinado ponto (H, D) do quantizador vetorial, o método de quantização com restrição de entropia vai buscar, localmente, pontos com menor distorção para uma dada entropia. Este resultando é apresentado na Figura 4.9.

4.2.2 Limites

Com os resultados da Seção 4.2.1, podemos estabelecer os limites dentro dos quais esperamos que o VQ baseado em funções Kernel esteja. Como o método com pior desempenho entre os testados é o método da SQ, desejamos que a VQ baseada em funções Kernel esteja acima da curva de SQ, sendo esta, portanto, nosso limite inferior. Superiormente, o método de ECVQ sempre pode ser inicializado com um dado dicionário, sobre o qual procurará otimizar o desempenho da quantização. Isto é, mesmo que seja possível encontrar uma curva do VQ baseado em funções Kernel com pontos melhores que o ECVQ, existirá um ECVQ que, quando inicializado com o mesmo dicionário do VQ baseado em funções Kernel, apresentará resultados ainda superiores em questão de desemepenho. Com isso, os resultados de SQ e ECVQ, como exibidos na Figura 4.10, serão tomados como os níveis de referência inferior e superior para os resultados esperados para a VQ baseada em funções Kernel.

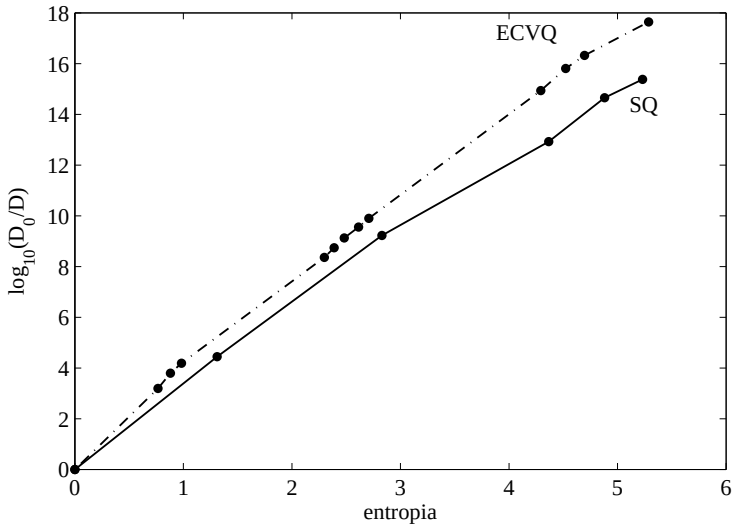


Figura 4.10: Limites esperados pro VQ baseado em funções Kernel.

4.2.3 Quantizador Vetorial baseado em Funções Kernel

Para este método de quantização, uma das funções Kernel usadas foi uma função de base radial (RBF), mostrada na tabela 2.3, que possui como parâmetro variável o valor $2\sigma^2$. Por questão de simplificação, adotamos $\gamma = 2\sigma^2$. Como visto na Seção 4.1.3, alterando os parâmetros de uma função Kernel, afetamos sua seletividade com relação à concentração de dados e também a não linearidade no mapeamento para o espaço das features. E a alteração destas características terá efeito sobre o desempenho do quantizador. Com isso, foi necessário variar o valor de γ a fim de encontrar os melhores pontos possíveis para o traçado da curva taxa-distorção.

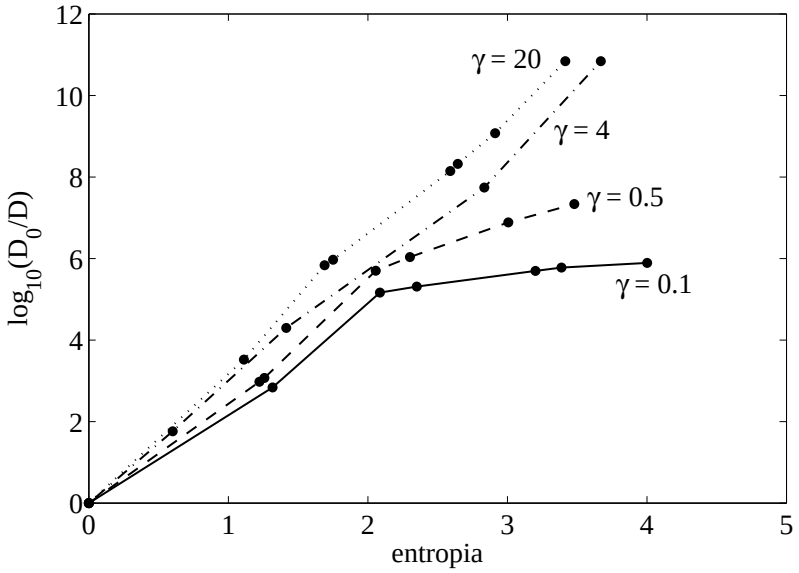


Figura 4.11: Comparação das curvas distorção x entropia conforme o parâmetro γ da função Kernel RBF, onde $k(x, y) = \exp(||x - y||^2/\gamma)$.

Como mostrado na Figura 4.11, o desempenho do VQ baseado na função Kernel RBF é significativamente sensível a variações no parâmetro γ. Para fins de estudo, utilizaremos, portanto, o valor de $\gamma = 20$, que apresentou o melhor resultado nas simulações. De posse das curvas de desempenho dos quantizadores clássicos, que limitam a região dos resultados esperados, conforme a Seção 4.2.2, e com a curva resultante do VQ baseado na função Kernel RBF, é possível avaliar a característica taxa-distorção do quantizador proposto.

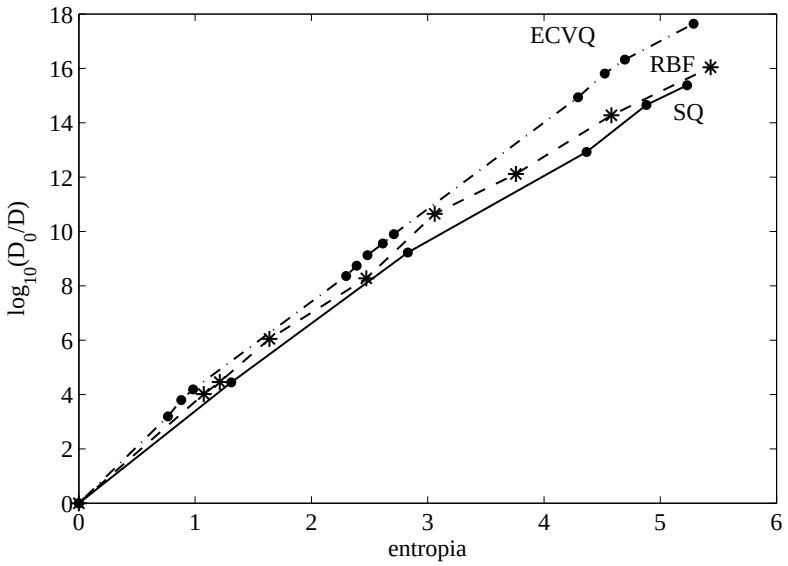


Figura 4.12: Casca convexa de 6450 pontos (H, D) do VQ baseado em função Kernel RBF, comparado com os quantizadores clássicos.

O método de VQ baseado em funções Kernel, conforme o resultado mostrado na Figura 4.12, apresenta desempenho superior ao método SQ, chegando a ser aproximadamente 1dB melhor que a SQ e se aproximando consideravalmente da ECVQ, em alguns pontos deste exemplo. A queda da curva RBF ao final não significa que exista um valor para entropia a partir do qual o método de SQ torna-se superior ao método de quantização proposto. Como observado na Seção 4.1.3, o projeto de um VQ baseado em funções Kernel descarta parte significativa das células com relação ao máximo de células que este é permitido alcançar. Isto significa que, durante o projeto, os pontos de entropia que se aproximam do limite da taxa ocorrem com menor frequência e a distribuição dos pontos (H, D) resultante torna-se menos densa. Portanto, pode-se esperar um resultado que, para estes pontos, não represente bem o desempenho máximo que o quantizador pode apresentar.

Outra função Kernel com a qual se pode alcançar quantizadores com desempenho satisfatório dentro dos limites estabelecidos anteriormente é a função tangente hiperbólica ou função Kernel sigmóide. Neste caso, como visto na Tabela 2.3, o grau de liberdade pra configuração da função Kernel passa de um, do VQ baseado em função Kernel RBF, para dois.

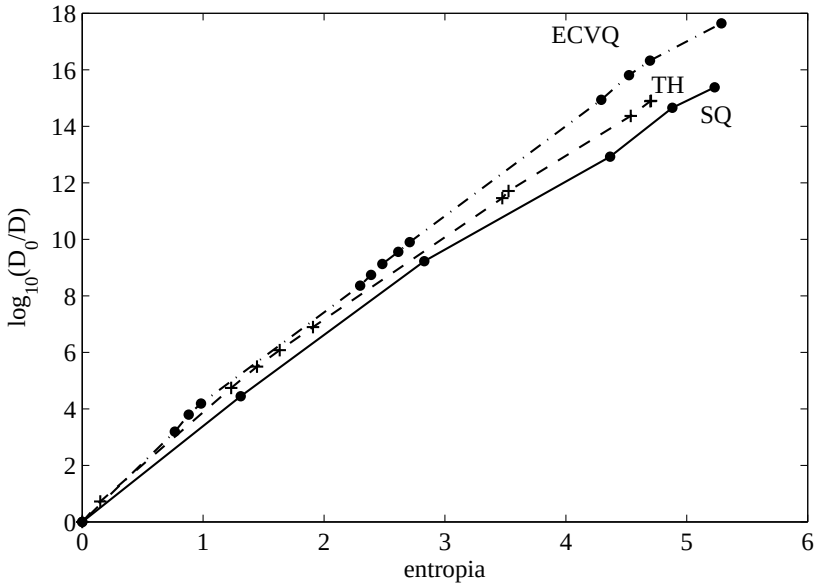


Figura 4.13: Casca convexa de 5550 pontos (H, D) do VQ baseado em função Kernel tangente hiperbólica, comparado com os quantizadores clássicos.

Os resultados de desempenho obtidos para este método, presentes na Figura 4.13, foram superiores aos obtidos para com base na função Kernel RBF, conforme a Figura 4.14. A curva alcança 1 dB de diferença em relação à SQ, mantendo-se relativamente próxima à curva do ECVQ em boa parte da faixa de entropia mais baixa.

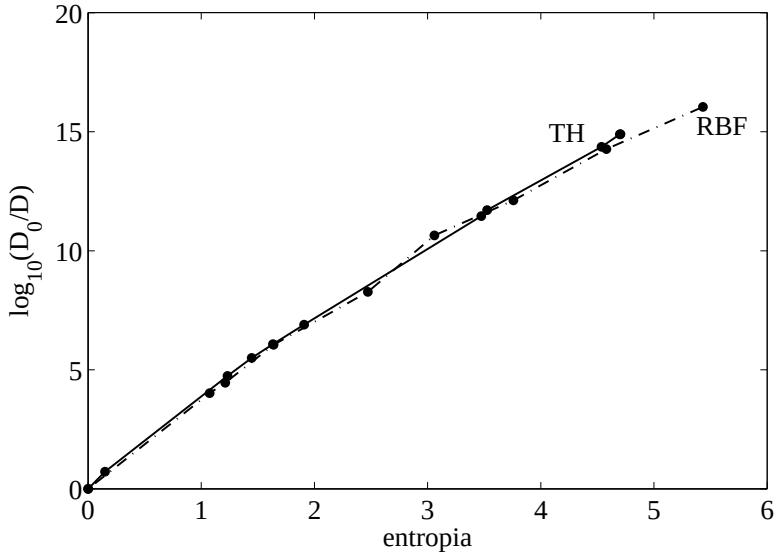


Figura 4.14: Comparaçao entre VQs baseados em função Kernel.

4.3 Complexidade

Utilizando os blocos básicos apresentados na seção 3.5, pode-se expressar a complexidade de cada um dos quantizadores estudados em função das unidades de complexidade definidas para sua implementação.

Quantizador Escalar

O SQ consiste, inicialmente, na comparação de um vetor de entrada $\mathbf{x}_n$ com os limiares do quantizador. A quantidade de limiares depende da taxa de bits R_m utilizada, sendo igual a $2^{R_m}-1$, com $m = 1, \ldots, M$. O SQ começa com complexidade $(2^{R_m} - 1)T_{\text{comp}}$ para cada uma das M dimensões de $\mathbf{x}_n$.

Para se obter a palavra binária resultante da quantização, as saídas dos comparadores são combinadas em portas XOR. A quantidade de portas cresce conforme o valor $\sum_{n=1}^{R_m-1} 2^n - 1 = 2^{R_m} - R_m - 1$. Por exemplo, para 2 bits, é necessário apenas um XOR. Para 3 bits, são necessários três XORs no primeiro nível e mais um no segundo nível. Isso adiciona $(2^{R_m} - R_m - 1)T_{\text{XOR}}$, por dimensão, ao quantizador escalar.

Como o imageador descrito na Seção 1.3 funciona somente até o cálculo dos índices de quantização, a complexidade associada à implementação da codificação de entropia não será considerada neste trabalho.

Quantização Vetorial

Para o VQ, a distância entre um vetor de entrada $\mathbf{x}_n$ e o k-ésimo vetor do dicionário, $k = 1, \ldots, K$, é calculada da seguinte forma:

$$d(n, k) = \mathbf{x}_n^T \mathbf{x}_n - \mathbf{x}_n^T \mathbf{y}_k + \mathbf{y}_k^T \mathbf{y}_k. \tag{4.1}$$

Como para um mesmo vetor de entrada o primeiro termo do lado direito da Equação 4.1 é sempre igual, este não precisa ser calculado. O terceiro termo depende apenas dos valores do dicionário e, portanto, pode ser pré-calculado. Com isso, é necessário encontrar apenas o resultado para o segundo termo, o que pode

ser obtido com o bloco de produto interno. Os vetores possuem comprimento M e a operação precisa ser realizada para todos os vetores do dicionário. Assim, a complexidade adicionada pelo cálculo das distâncias euclidianas é $(M + T_{\text{cc}} + 1)K$ unidades. O acréscimo de uma unidade ao cálculo de cada distância se deve ao *bias* que representa o termo pré-calculado. Depois disso, a função mínimo, acrescentando KT_{wta}, retorna a menor distância.

VQ baseado em Função Kernel Tangente Hiperbólica

Escalares $q(n)$ são gerados multiplicando-se o vetor de entrada $\mathbf{x}(n)$ pelos K $\mathbf{l}_j$ e somando a um *bias* definido. Considerando que estes vetores possuem dimensão M, são necessários $K(M + T_{\text{cc}} + 1)$ unidades. O bloco que implementa a função tangente hiperbólica é utilizado para realizar o mapeamento $\tanh(q(n))$ do vetor de entrada $\mathbf{x}_n$ sobre cada um dos vetores $\mathbf{l}_j$. O custo desta operação é $KT_{\tanh}$.

A operação de Kernel PCA, que consiste na multiplicação $\mathbf{Wz}(n)$, onde $\mathbf{W}$ tem N' vetores de peso, mais a adição de um *bias* $\mathbf{b}$, vistos na Equação (2.14), tem custo $(K+T_{cc})$ devido aos produtos interos e o acréscimo de uma unidade de complexidade pelo *bias*, por cada vetor de peso. Isto é, $N'(K + T_{\text{cc}} + 1)$ unidades. Finalmente, o SQ aplicado em cada *features* acrescenta o custo dos comparadores. Como, para cada *feature*, é alocada uma quantidade de bits, o custo total pode ser expresso por $\sum_w (2^{R_w} - 1)T_{\text{comp}} + (2^{R_w} - R_w - 1)T_{\text{xor}}$, onde $w = 1, \ldots, N'$ é o índice de cada *feature*.

VQ baseado em Função Kernel RBF

Paro o VQ baseado em função Kernel RBF, o argumento da função exponencial pode ser escrito como na Equação (4.1), resultando na multiplicação de três exponenciais. O termo central adiciona complexidade $K(M+T_{\text{cc}})$, pois necessita do produto interno $\mathbf{x}^T\mathbf{y}_k$, onde $k = 1, \ldots, K$ é o número de vetores utilizados para o mapeamento. O primeiro termo, corresponente a $\mathbf{x}^T\mathbf{x}$ precisa ser calculado apenas uma vez por vetor de entrada e utiliza a função quadrado, adicionando $(MT_{\text{quad}} + T_{\text{cc}})$. Somando os dois termos e realizando a função exponencial, adicionamos $K(T_{\text{exp}}+T_{\text{cc}})$

unidades. O termo dependente de $\mathbf{y}_k^T \mathbf{y}_k$ pode ser pré-calculado e incorporado à matriz da análise de componentes principais.

Para realizar a etapa $\mathbf{Wz} + \mathbf{b}$ da Equação (2.14), são necessárias mais $N'(K + T_{\text{CC}} + 1)$ unidades, onde N' corresponde à quantidade de *features* utilizada. Para a etapa do SQ, aplicado nas *features*, são incluidos os custos das comparações e das portas XOR $\sum_w (2^{R_w} - 1)T_{\text{comp}} + (2^{R_w} - R_w - 1)T_{\text{xor}}$.

Com base nas características de complexidade obtidas para os quantizadores, é possível calcular quantas unidades são necessárias para se obter os pontos (H, D) exibidos na Seção 4.2. Estes resultados são apresentados nas Figuras 4.15 e 4.16.

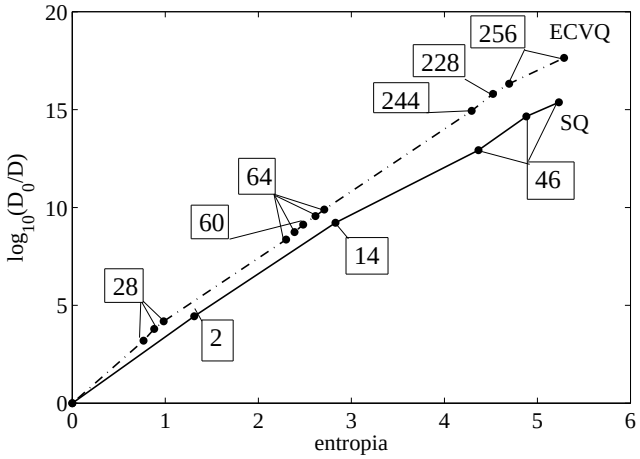


Figura 4.15: Complexidades dos pontos (H, D) dos quantizadores tradicionais.

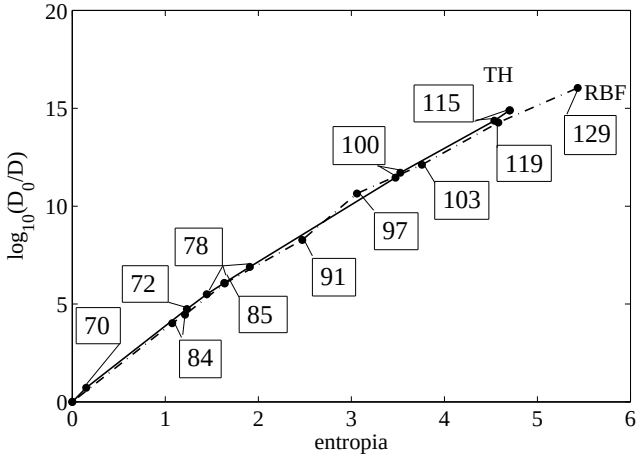


Figura 4.16: Complexidades dos pontos (H, D) dos VQs baseados em funções Kernel.

4.4 Comparação dos Resultados

As Figuras 4.15 e 4.16, juntamente com os resultados obtidos na Seção 4.2 concluem as comparações dos quantizadores em função dos parâmetros entropia, distorção e complexidade.

Para entropia em torno de $H = 4$, o VQ baseado em função Kernel tangente hiperbólica consegue apresentar distorção D até 1 dB menor que o SQ, enquanto o VQ baseado em função Kernel RBF apresenta distorção aproximadamente 0,7 dB menor que o SQ. Para a mesma região, segundo os resultados da seção 4.3, os VQs baseados em funções Kernel são implementados com complexidade estimada duas vezes menor que a complexidade necessária para implementação de um VQ com restrição de entropia.

Capítulo 5

Conclusões

Este trabalho permitiu que fossem postos lado a lado alguns dos métodos de quantização mais comuns aplicados em diferentes áreas da engenharia e a introdução a um método mais recente, que é a VQ baseada em Kernel PCA. Tais métodos foram detalhados desde sua teoria, amplamente estudada, até a complexidade de suas possíveis implementações. O resultado é um conjunto de informações sobre estes métodos, de forma que é comparar suas propriedades e obter relações entre as técnicas.

Para a quantização vetorial baseada em funções Kernel, foram obtidos resultados que mostram que sua implementação pode ser competitiva perante os outros métodos de quantização. Para isso, duas funções Kernel foram abordadas, sendo elas a função tangente hiperbólica e a função de base radial (RBF) gaussiana. Entre elas, melhores desempenhos foram obtidos para a tangente hiperbólica, cuja curva distorção $\times$ entropia mais se aproximou dos resultados obtidos para uma VQ com restrição de entropia, embora tenha sido consideravelmente mais difícil entender o funcionamento dos parâmetros desta função Kernel.

Verificado o desempenho competitivo do VQ baseado em funções Kernel, o próximo passo é adaptar o método para sua aplicação a técnicas de compressão de imagem, fazendo a análise para mais de duas dimensões, e estudar a possibilidade de sua implementação prática, em circuitos integrados. Além disso, outros estudos possíveis envolvem a otimização destes métodos de quantização, desde a variação de seus parâmetros à escolha da função Kernel ideal para cada aplicação.

Referências Bibliográficas

[1] OLIVEIRA, F., HAAS, H., GOMES, J., *et al.*, "CMOS Imager with FocalPlane Analog Image Compression Combining DPCM and VQ", *IEEE Trans. Circuits and Systems I: Regular Papers*, v. 60, pp. 1331 – 1344, Maio 2013.

[2] GERSHO, A., GRAY, R. M., *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, 1991.

[3] HUFFMAN, D., "A Method for the Construction of Minimum-Redundancy Codes", *Proceedings of the IRE*, v. 40, pp. 1098 – 1101, Setembro 1952.

[4] LLOYD, S., "Least squares quantization in PCM", Bell Telephone Laboratories Paper, 1957.

[5] MAX, J., "Quantizing for minimum distortion", *IRE Transactions, Information Theory*, v. 6, pp. 7 – 12, Março 1960.

[6] LINDE, Y., BUZO, A., GRAY, R., "An Algorithm for Vector Quantizer Design", *IEEE Trans. Communications*, v. 28, pp. 84 – 95, Janeiro 1980.

[7] SCHÖLKOPF, B., SMOLA, A., MÜLLER, K.-R., "Nonlinear component analysis as a kernel eigenvalue problem", *Neural Comput.*, v. 10, n. 5, pp. 1299–1319, Jul. 1998.

[8] CHOU, P., LOOKABAUGH, T., GRAY, R., "Entropy-Constrained Vector Quantization", *IEEE Trans. Acoustics, Speech and Signal Processing*, v. 37, pp. 31 – 42, Janeiro 1989.

[9] GOMES, J. G. R. C., MITRA, S. K., "A Comparative Study of the Complexities of Neural Network Based Focal-Plane Image Compression Schemes", *IEICE*

Transactions on Fundamentals of Electronics, Communications, and Computer Sciences, v. J88-A, pp. 1185 – 1196, Novembro 2005.

[10] SEDRA, A. S., ROBERTS, G. W., GOHH, F., "The current conveyor: history, progress and new results", *IEE Proceedings G. Circuits, Devices and Systems*, v. 137, pp. 78 – 87, Abril 1990.

[11] ANDREOU, A., BOAHEN, K., POULIQUEN, P., *et al.*, "Current-mode subthreshold MOS circuits for analog VLSI neural systems", *IEEE Trans Neural Networks*, v. 2, pp. 205 – 213, Março 1991.

[12] BULT, K., WALLINGA, H., "A class of analog CMOS circuits based on the square-law characteristic of an MOS transistor in saturation", *IEEE Journal of Solid-State Circuits*, v. 22, pp. 357 – 365, Junho 1987.

Apêndice A

Codificação de Huffman

A *codificação de Huffman* é um algoritmo que gera códigos de prefixo e tem como idéia básica a minimização da redundância existente nas palavras binárias utilizadas para representação de mensagens de uma determinada fonte. O código de Huffman gerado tem comprimento médio $\overline{L}$ das palavras que se aproxima do limite $H(\varphi)$, definido como a entropia de uma onte discreta sem memória, respeitando o primeiro teorema de Shannon. Portanto:

$$\overline{L} \geq H(\varphi) \tag{A.1}$$

Dado um conjunto de mensagens e suas probabilidades, algoritmo de codificação de Huffman pode ser descrito pelos seguintes passos:

1. Ordenar as mensagens de acordo com as probabilidades

2. Agrupar as duas mensagens de menor probabilidade em uma nova mensagem, cuja probabilidade é a soma das duas mensagens originais. Um novo conjunto de mensagens é formado a partir das mensagens que restaram e desta nova mensagem. Cada passo gera um novo nó de uma árvore binária.

3. Repetir estes passos até que restem apenas duas mensagens.

Ao final dos passos acima, atribui-se 0 e 1 às duas mensagens finais. Para se obter a palavra binária correspondente a cada mensagem, basta fazer o caminho reverso da árvore, atribuindo 0 e 1 para os dois caminhos possíveis a partir de cada nó. Quando se alcançar a mensagem, a palavra corresponderá ao caminho percorrido.