Universidade Federal do Rio de Janeiro

Escola Politécnica

Departamento de Eletrônica e de Computação

MILC - Um CODEC de áudio sem perdas

Autor:

Michel Igor de Almeida Ennes

Orientador:

Prof. Luiz Wagner Pereira Biscainho, D.Sc.

Examinador:

Prof[a] Mariane Rembold Petraglia, Ph.D.

Examinador:

Eng. Leonardo de Oliveira Nunes, M.Sc.

DEL

Março de 2010

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

Dedico este trabalho a todos que cooperaram para o meu ingresso na universidade e também àqueles que me apoiaram a continuar na luta pelo sonho de ser engenheiro, especialmente minha família e minha noiva, que estiveram junto a mim ao longo de toda essa jornada e permanecerão continuamente presentes por todo o resto de minha vida.

AGRADECIMENTO

Agradeço, primeiramente, a Deus, pois foi ele quem me deu forças ao longo desta árdua batalha. É importante também agradecer à minha família, à minha noiva, aos meus amigos, ao meu orientador e aos professores, uma vez que, seja de uma forma ou de outra, ajudaram-me a manter o foco, que sempre foi a graduação em engenharia.

RESUMO

Este trabalho aborda a implementação do MILC (Michel Igor Lossless CODEC), que é um CODEC de áudio sem perdas compatível com o padrão MPEG-4 ALS. O MILC é baseado em alguns princípios básicos de compressão tais como segmentação, predição linear *forward*-adaptativa e codificação por entropia. Primeiramente, o leitor encontrará uma pesquisa de diversos CODECs disponíveis na literatura e facilmente acessíveis na literatura. Em seguida, descrevem-se as ferramentas básicas e outras adicionais do MPEG-4 ALS, e os resultados de simulações com oito arquivos de áudio. Por fim, esta documentação descreve como o MILC foi implementado e que ferramentas o compõem, e faz uma comparação das codificações do MILC e do MPEG-4 ALS.

Palavras-chave: MILC, MPEG-4 ALS, sem perdas, codificação, decodificação.

ABSTRACT

This work describes the implementation of the MILC (Michel Igor Lossless CODEC), which is a lossless audio CODEC compatible with the MPEG-4 ALS standard. The MILC is based on some basic principles of compression such as framing, forward-adaptative linear prediction and entropy coding. First of all, the reader will find a survey of the various CODECs available in the literature and easily accessible. After met, an explanation of the basic and some additional sophisticated tools of the MPEG-4 ALS and results of simulations with eight audio files. Finally, this documentation describes how the MILC was implemented and which tools are included in it, and makes a comparison of the MILC and the MPEG-4 ALS encodings.

Key-words: MILC, MPEG-4 ALS, lossless, encoding, decoding.

SIGLAS

MILC - Michel Igor Lossless CODEC

CODEC - Codificador/Decodificador

MPEG - Moving Picture Experts Group

ALS - Audio Lossless Coding

FLAC - Free Lossless Audio CODEC

ALAC - Apple Lossless Audio CODEC

TAK - Tom's Lossless Audio Kompressor

LA - Lossless Audio

TTA - True Audio

LPAC - Lossless Predictive Audio Coder

WMA - Windows Media Audio

BGMC - Block Gilbert-Moore Code

LPC - Linear Predictive Coding

NLC - New Lossless Coding

TUB - Technical University of Berlin

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Tema

O trabalho trata essencialmente de codificação de áudio sem perdas, diretamente atrelado à grande área de "Processamento de Sinais"; envolvendo o conhecimento de tópicos como quantização, predição linear e codificação por entropia.

Sua finalidade é comprimir ao máximo, por codificação, os arquivos de áudio em geral utilizando-se essas técnicas como base, permitindo obter um arquivo idêntico ao original após sua decodificação.

1.2 Delimitação

Entre as pessoas que gostam de música, provavelmente, há aquelas que têm preferência por música clássica, outras preferem MPB, samba etc. É difícil encontrar alguém que não gosta de gênero algum. A música é passatempo para uns, terapia para outros, profissão para muitos e arte para poucos.

Diante dos fatos levantados acima, e com o avanço da tecnologia de informação, é muito comum pessoas acessarem a *web* para baixar músicas. Neste processo, é interessante notar que os arquivos postados na *web* (neste caso músicas), em sua grande maioria, sofrem algum tipo de compressão por motivos pecuniários[1] e de

[1]Excetuando-se condições específicas, quanto menor o tamanho do arquivo a ser postado na internet, menos se paga por este serviço.

desempenho[2].

Há diversas maneiras de compactar arquivos de áudio. Dentre elas, podem-se destacar as codificações com e sem perdas. Esta última é uma codificação sem qualquer perda em relação ao arquivo original, na medida em que somente as redundâncias são excluídas, e portanto o procedimento é inversível por decodificação. Já a primeira elimina, além de redundâncias, informações que não seriam perceptíveis pelo ouvido humano; sua meta é a "transparência"[3] na codificação, sem ser necessariamente inversível na decodificação.

As pessoas que efetuam a compressão destes arquivos optam por uma das modalidades de codificação. Se elas desejarem uma cópia fiel do arquivo original, devem escolher um arquivo que tenha sido codificado sem perdas de informação. As implicações desta opção são arquivos não tão comprimidos, porém com 100% de fidelidade em relação ao arquivo original. Em contrapartida, as pessoas que desejarem arquivos muito menores devem optar pela modalidade com perdas; esta não fornece uma cópia fiel do original, pois, como já visto, há perdas, mas tem capacidade de compressão aumentada em relação à outra modalidade discutida, permitindo guardar uma quantidade maior de arquivos nos dispositivos de armazenamento.

1.3 Justificativa

Por que se deve codificar e comprimir os arquivos, em geral? A compressão de dados possibilita melhorar a eficiência em termos de transmissão e armazenamento dos mesmos.

Em se tratando de dados de vídeo, por exemplo, se hoje a TV digital está se iniciando em nosso país, muito se deve aos algoritmos de codificação desenvolvidos,

[2]Quanto menor o tamanho do arquivo postado na *web*, mais rápidas são realizadas as operações de *upload* e *download* e mais arquivos podem ser armazenados dentro de um limite de dados pré-estabelecido.

[3]Transparência quer dizer que as pessoas conseguem ouvir todos os detalhes de um determinado sinal de áudio com qualidade.

como o MPEG-2, posteriormente o MPEG-4 e evoluções deste último. Estes algoritmos possibilitaram uma redução na taxa de *bits* transmitidos, mantendo-se a mesma qualidade da imagem para uma mesma resolução.

Já para arquivos de áudio, considerando dois *GBytes* a capacidade de armazenamento de um dispositivo de reprodução de áudio, provavelmente cerca de 50 músicas com extensão ".wav" (200 minutos) poderiam ser armazenadas nele. Porém, se estes mesmos arquivos fossem codificados, com ou sem perda de informação, a quantidade de músicas armazenadas dobraria, na média, para esta última modalidade de codificação, e seria possível armazenar até dez vezes mais músicas para a primeira modalidade abordada.

1.4 Objetivos

Este trabalho tem por finalidade: comparar os principais codificadores sem perdas existentes na literatura segundo parâmetros tais como taxa de codificação, tempo necessário para a codificação, flexibilidade e sistemas operacionais compatíveis; e implementar um CODEC de áudio sem perdas compatível com o MPEG-4 ALS com as ferramentas básicas como predição linear, *framing* e codificação por entropia, e outras mais sofisticadas como codificação de canais realizada em conjunto e ordem de predição adaptativa.

1.5 Metodologia

Este trabalho seguiu uma linha na qual primeiramente estudaram-se os CODECs de áudio sem perdas existentes na literatura, dentre os quais FLAC [2], WavPack [3], TAK [4], Monkey's [5], OptimFROG, Apple *Lossless*, WMA *Lossless*, Shorten [6], LA [7], TTA [8], LPAC [9], MPEG-4 ALS [10], Real *Lossless*. Posteriormente, elegeu-se o padrão MPEG-4 ALS como base para a implementação de um CODEC de áudio (sem perdas) próprio. Os critérios utilizados nessa opção foram: aceitação da tecnologia no mercado, quantidade de informações presentes na literatura e nível de compressão atingido.

1.6 Descrição

No Capítulo 2 serão discutidos os CODECs de áudio sem perdas presentes na literatura, através das particularidades de cada um e dos resultados da pós-codificação de arquivos de áudio pré-determinados.

O Capítulo 3 apresenta os princípios básicos de codificação sem perdas em geral.

No Capítulo 4, aborda-se especificamente o padrão MPEG-4 ALS, bem como seus principais recursos.

O Capítulo 5 trata da descrição do MILC e o modo como foi implementado.

O Capítulo 6 apresenta os testes realizados no MPEG-4 ALS e no MILC.

Por fim, no Capítulo 7, será apresentada a conclusão deste trabalho.

Capítulo 2

Comparação de CODECs sem Perdas

Neste capítulo, são realizadas comparações entre diversos CODECs de áudio sem perdas disponíveis na literatura. Há uma descrição completa da metodologia empregada nesse processo que engloba arquivos de áudio utilizados, *softwares* específicos e tabelas que ilustram os resultados de todos os CODECs testados.

2.1 Metodologia de Comparação

O primeiro passo deste trabalho foi procurar na literatura [15] e na internet [14] informações a respeito de codificadores sem perdas existentes no mercado. Com isso, obteve-se a lista de codificadores mostrada na Tabela 2.1. Acredita-se que estes codificadores representam o estado da arte sem perdas, ao menos dentre os disponíveis e com acessibilidade a qualquer usuário.

Em seguida, teve-se de definir quais tipos de arquivo seriam utilizados no trabalho. Eles estão definidos na Tabela 2.2. Os sete primeiros arquivos desta tabela são naturais[1]. Já os dois últimos arquivos são artificiais e foram gerados no MATLAB®, sendo o primeiro deles um sinal composto por uma soma de dez senóides com amplitudes unitárias e cujas frequências angulares Ω variam de 1 até 10 rad/amostra com passo unitário, e o segundo, um sinal composto pela mesma soma de senóides

[1]Não sintéticos originados a partir de instrumento e/ou voz.

CODECs	**Criadores**	**Desenvolvedores**
FLAC [2]	Xiph.Org Foundation	Josh Coalson
Monkey's [5]	Matthew T. Ashland	Matthew T. Ashland
OptimFrog	Florin Ghido	Florin Ghido
LPAC [9]	TUB	Tilman Liebchen
WavPack [3]	David Bryant	David Bryant
LA [7]	Michael Bevin	Michael Bevin
ALAC	Apple Inc.®	Apple Inc.®
Shorten [6]	Soft Sound	Tony Robinson
TTA [8]	Alexander Djourik	Alexander Djourik
Real Lossless	Real Networks®	Real Networks®
WMA Lossless	Microsoft Corp.®	Microsoft Corp.®
TAK [4]	Thomas Becker	Thomas Becker
MPEG-4 ALS [11] [12] [13] [10]	TUB	TUB

Tabela 2.1: Criadores e desenvolvedores dos CODECs sem perdas. Fonte: [14].

somado com ruído branco gaussiano de forma a se obter uma SNR de 2dB. Vale ressaltar que a freqüência de amostragem para todos os arquivos é de 44,1 kHz.

Depois, foi necessário efetuar *downloads* de uma série de *softwares* para os respectivos codificadores que podem ser visualizados na Tabela 2.3. Estes programas são responsáveis pelas codificações dos arquivos de entrada (originais) e para gerar os arquivos de saída (arquivos codificados).

O próximo passo foi escolher em que modo os arquivos seriam codificados. Inicialmente, a metodologia usada nas codificações foi exaustiva, porque alguns codificadores tais como o FLAC, o Monkey's e o MPEG-4 ALS permitem diversos modos de codificação. Resumidamente, estes modos levam em consideração alguns fatores básicos como tempo de codificação e tamanho do arquivo de saída. Por isso, foi necessário codificar todos os arquivos originais em todos os modos para estes codificadores e levar em consideração algum critério para a escolha de apenas um

Arquivos	**Tamanho em Mbytes**	**Duração (s)**	**Descrição**
Berimbau	4,210	25,0	som de um berimbau
BluesInG-MJQ	3,700	22,0	show ao vivo
Cravo	2,350	14,0	música rápida - instrumentos
Dubal	2,370	14,1	voz
HaGente Aqui-MJ	3,570	21,2	música rápida - instrumentos e voz
Largo-Schuls-hv	2,140	12,7	frequências agudas bem definidas
Valsa-Chopin-Godowsky-1916	2,720	16,2	gravação antiga e ruidosa
Senoides	2,003	22,7	frequências baixas bem definidas
Senoides-Ruido-Branco	2,003	22,7	áudio muito ruidoso

Tabela 2.2: Arquivos de áudio usados.

modo. Apesar disso, optou-se pela comparação entre os modos *default* dos CODECs, pois alguns CODECs como TTA, Shorten e Apple *Lossless*, que foram codificados utilizando-se o *software* dBpoweramp, permitiam somente um modo de simulação.

O método de análise dos resultados depende da aplicação final do CODEC sob teste. Para uso pessoal, os resultados baseiam-se em comparações entre as taxas de codificação dos codificadores. Em transmissões, especificamente tratando do exemplo apresentado na Seção 2.2.2, o principal parâmetro a ser comparado é o tempo de codificação. E para implementações de codificadores, deve-se atentar para parâmetros como flexibilidade e ser *open-source* ou não.

Software	**Arquivos codificados**
dBpoweramp Music Converter™ [16]	ALAC, WMA Lossless, Real Lossless, Shorten, TTA, FLAC, Monkey's, OptimFROG e WavPack
La [7]	LA
LPACarq [9]	LPAC
mp4alsRM21 [17]	MPEG-4 ALS
TAK [4]	TAK

Tabela 2.3: *Softwares* utilizados nas codificações.

2.2 Resultados

Comparando-se os resultados obtidos com as simulações, os quais podem ser vistos nas Figuras 2.1 e 2.3, e as informações contidas na Tabela 2.4 é possível apontar os melhores codificadores para cada tipo de aplicação. Pode-se ainda adotar um compromisso entre tempo de codificação e taxa de codificação, além de considerar outros parâmetros como sistemas operacionais compatíveis com cada um deles, ser *open-source* ou não e a quantidade de parâmetros no processo de codificação.

Vale lembrar que nos testes realizados, somente os arquivos de origem natural foram considerados, uma vez que o teste com o sinal sintético composto apenas de senóides (veja Seção 2.1) foi codificado eficientemente, i.e. com baixíssima taxa de codificação e o sinal composto pelas senóides e pelo ruído branco apresentou um resultado muito ruim, i.e. com elevadas taxas de codificação (próximas ou superiores a 100 %).

Parâmetros	**FLAC**	**Wav Pack**	**TAK**	**Monkey's**	**Optim FROG**	**ALAC**	**WAV**	**Shorten**	**LA**	**TTA**	**LPAC**	**MPEG-4 ALS**	**Real Lossless**
Velocidade de Codificação	alta	**muito alta**	**muito alta**	alta	baixa	mode-rada	mode-rada	**muito alta**	baixa	**muito alta**	moderada	moderada	baixa
Velocidade de Decodificação	**muito alta**	**muito alta**	**muito alta**	moderada	moderada	alta	mode-rada	**muito alta**	baixa	alta	alta	alta	baixa
Flexibilidade	**muito alta**	**muito alta**	**muito alta**	**muito alta**	**muito alta**	baixa	baixa	baixa	baixa	baixa	baixa	**muito alta**	baixa
Suporte a *Hardware*	**muito bom**	limitado	não há	limitado	não há	bom	limitado	limitado	não há	limitado	não há	não há	não há
Suporte a *Software*	**muito bom**	bom	médio	bom	médio	ruim	bom	**muito bom**	ruim	médio	médio	ruim	ruim
Open-Source	**sim**	**sim**	não	**sim**	não	não	não	**sim**	não	**sim**	não	**sim**	não
Multicanal	**sim**	**sim**	não	não	não	**sim**	**sim**	não	não	**sim**	não	**sim**	não
Alta Resolução	**sim**	**sim**	**sim**	**sim**	**sim**	**sim**	**sim**	não	não	**sim**	**sim**	**sim**	não
Sistema Operacional	**todos**	**todos**	Win/ Linux	**todos**	Win/Mac/ Linux	Win/ Mac	Win/ Mac	**todos**	Win/ Linux	**todos**	Win/Sol/ Linux	**todos**	Win/Mac/ Linux

Tabela 2.4: Análise de parâmetros. Adaptada de [14]. Significados: alta – o CODEC é muito flexível ou muito rápido; médio – moderado; baixa – o CODEC é pouco flexível ou lento; muito bom – é bem "aceito" por tecnologias diferentes; bom – é "aceito" por algumas tecnologias; ruim – tem sérias restrições para serem usados

2.2.1 Aplicação 1: Uso Pessoal

Caso se deseje apenas codificar músicas para fins de armazenamento em dispositivos de reprodução de áudio para uso pessoal, deve-se levar em consideração a taxa de codificação média para cada um dos CODECs (Figura 2.3) e o desvio padrão desta mesma informação (Figura 2.4), uma vez que é razoável que não haja muita exigência para o tempo total no processo de codificação. Portanto, escolheu-se como critério, para este caso, o menor tamanho do arquivo de saída em *bytes* e o menor desvio padrão para esse indicador.

Considerando-se as Figuras 2.3 e 2.4, os CODECs OptimFROG, LA, Monkey's, TAK e WMA Lossless são os melhores segundo o parâmetro taxa de codificação média (Figura 2.3). Analisando-se o desvio padrão da taxa de codificação, percebe-se que esse parâmetro indica uma robustez de alguns CODECs tais como Monkey's, WMA Lossless, TAK e Apple Lossless. Logo, é possível apontar os melhores codificadores, segundo o parâmetro taxa de compressão como sendo: Monkey's, TAK e WMA Lossless.

2.2.2 Aplicação 2: Transmissão de Arquivos de Áudio

Suponha-se o exemplo hipotético ilustrado na Figura 2.5 no qual deseja-se transmitir um sinal de voz, *frame* por *frame*, para um receptor. Para simplificar este exemplo, algumas considerações devem ser explicitadas: o tempo de codificação de um *frame* é bem menor que o seu tempo de decodificação e seu tempo de transmissão é desprezível se comparado aos tempos de codificação e decodificação; há um *buffer* no receptor que armazena os *frames* recebidos, os quais são decodificados e ouvidos tão logo estejam disponíveis no receptor, na ordem em que foram transmitidos; o *buffer* tem espaço para armazenamento suficiente para que não ocorra *overflow*; o tempo de codificação de qualquer quadro é aproximadamente t_{total}/N, onde t_{total} indica o tempo de codificação total e N significa o tamanho de um quadro; e não há perda de quadros.

Uma vez entendido este exemplo, pode-se então analisá-lo sob a ótica da codificação do sinal de voz emitido. Neste caso, é preciso levar em consideração,

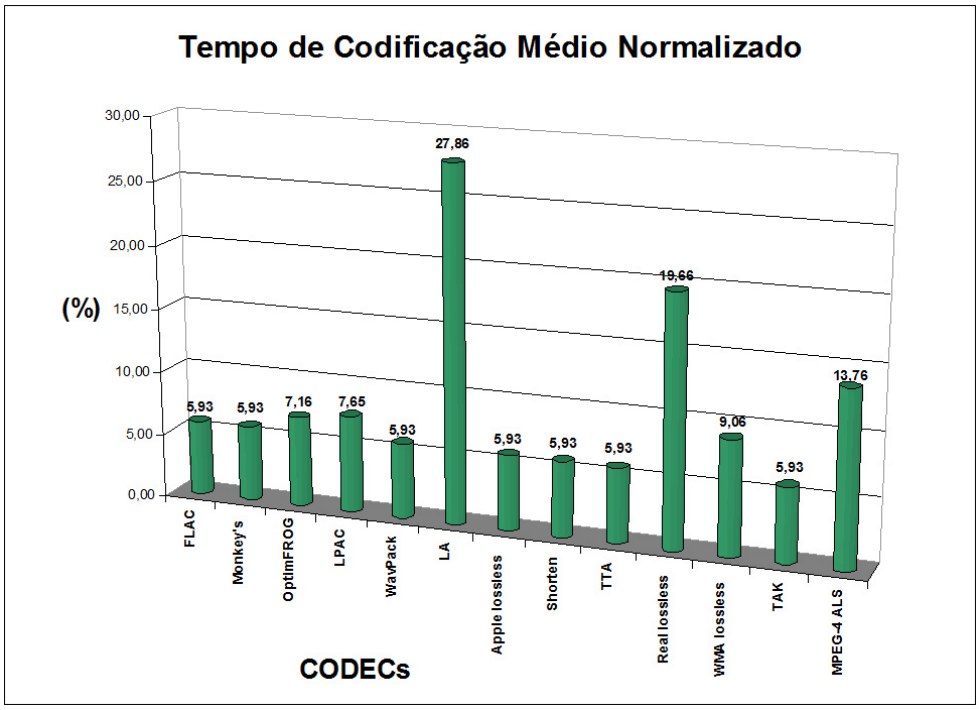

Figura 2.1: Tempo de codificação médio normalizado por CODEC.

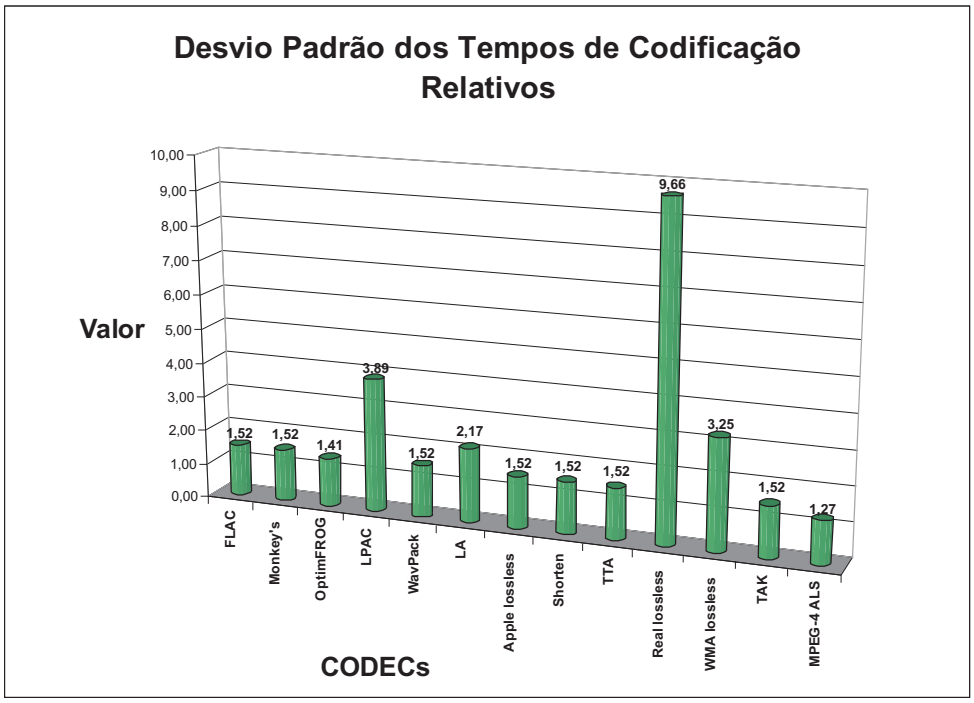

Figura 2.2: Desvio padrão do tempo de codificação normalizado por CODEC.

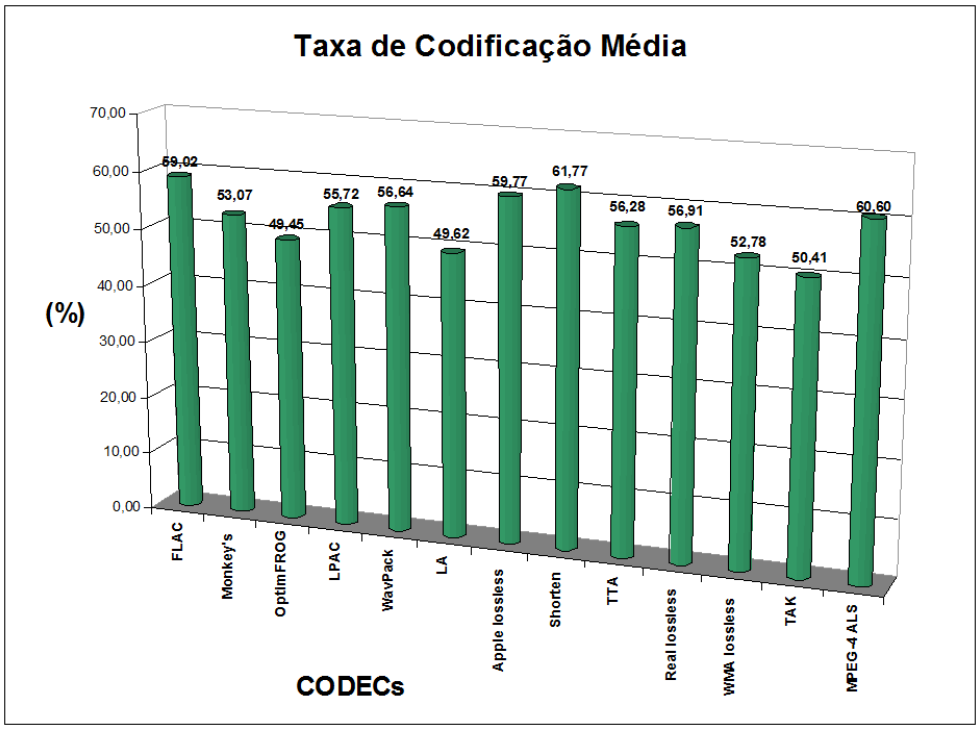

Figura 2.3: Taxa de codificação média por CODEC.

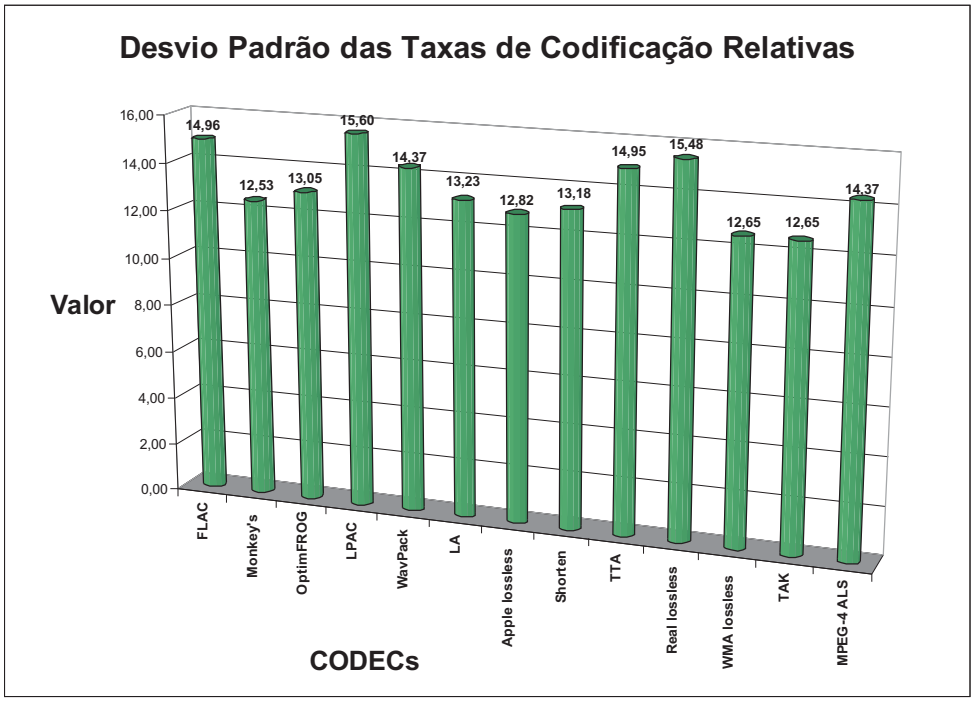

Figura 2.4: Desvio padrão da taxa de codificação por CODEC.

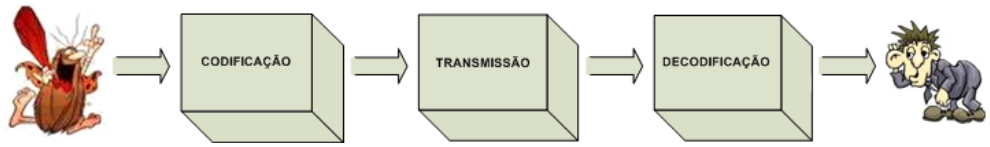

Figura 2.5: Exemplo de aplicação.

principalmente, o tempo de codificação do arquivo em questão (Figuras 2.1 e 2.2). Afinal, conforme as considerações feitas inicialmente, o maior tempo na cadeia de emissão/recepção do sinal de voz é o de codificação.

Assim sendo, os codificadores LA, Real lossless e MPEG-4 ALS tiveram os piores desempenhos, considerando-se o tempo médio normalizado (em relação ao tempo de reprodução dos respectivos sinais), pois este parâmetro foi superior ao dobro da média dos demais CODECs.

Em se tratando dao desvio padrão do tempo de codificação normalizado, percebe-se que os CODECs LPAC, WMA Lossless e, principalmente, Real Lossless apresentaram os maiores valores para esse indicador, o que significa que os tempos de simulação normalizados deles divergiram muito em função das características dos sinais pré-codificados, i.e. esses CODECs são os menos robustos.

Portanto, os CODECs FLAC, Monkey's, WavPack, Apple Lossless, Shorten, TTA e TAK são os mais adequados para esta aplicação, pois apresentam os menores tempos de codificação.

2.2.3 Outras Aplicações: Estudo

Outro exemplo de aplicação poderia ser uma pessoa que deseja estudar mais aprofundadamente um ou outro codificador. Este trabalho está nessa situação, pois trata do estudo do MPEG-4 ALS e da implementação de um novo CODEC compatível com esse padrão. Neste tipo de exemplo, deve-se atentar para algumas características dos codificadores, tais como flexibilidade, sistemas operacionais compatíveis com a tecnologia, os códigos-fonte serem ou não abertos, aceitação no mercado etc.

2.2.4 Considerações Finais

Em suma, a escolha de um codificador para uma determinada aplicação de compressão, envolve muitos parâmetros. Por isso, é necessário, primeiramente, definir bem a aplicação na qual o codificador irá atuar e, em seguida, identificar qual atenderá melhor as especificações dessa aplicação, segundo seus principais parâmetros.

No próximo capítulo, serão estudados alguns conceitos teóricos fundamentais para se trabalhar com codificação/decodificacão sem perdas, tais como predição linear e codificação de entropia.

Capítulo 3

Princípios Básicos de Codificação de Áudio sem Perdas

Neste capítulo, aborda-se-ão alguns conceitos fundamentais na codificação de áudio sem perdas.

A Figura 3.1 é uma representação em diagrama de blocos das operações básicas envolvidas na codificação/compressão de um canal simples de áudio dos CODECs citados neste documento, inclusive o MPEG-4 ALS.

No processo de compressão, é importante, em primeiro lugar, explorar características do sinal de áudio tais como a peridiocidade e a forte correlação entre trechos adjacentes visando a uma representação mais compacta (que necessita de um menor número de *bits*) para o sinal. Nas próximas seções, serão detalhados todos os blocos da Figura 3.1.

3.1 Segmentação

Consiste em dividir o sinal de áudio digital em quadros *(frames)* com comprimentos iguais para serem processados. O objetivo desta etapa é permitir que a similaridade entre amostras temporalmente próximas do sinal de áudio seja explorada.

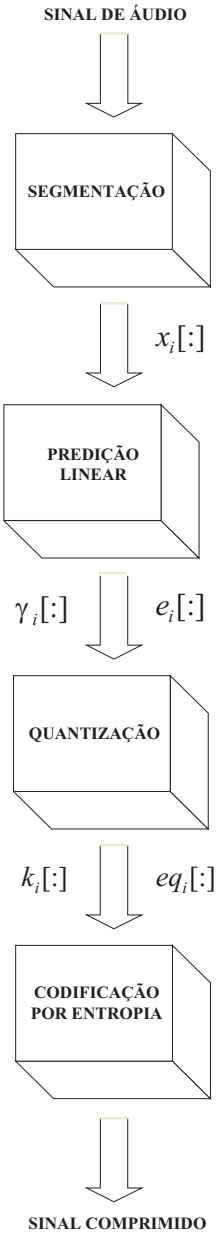

Figura 3.1: Operações básicas nos algoritmos de compressão sem perdas.

Segmentos muito curtos não possuem dados suficientes para que a similaridade seja explorada, enquanto que segmentos muito longos podem conter trechos dissimilares de áudio.

Segundo a Figura 3.1, após a etapa de segmentação, o sinal de áudio é dividido em quadros $x_i[n]$, onde i é o índice do quadro e n é o índice da amostra dentro do quadro.

3.2 Predição Linear

Neste bloco, é obtido um modelo linear para um segmento do sinal de áudio. Tal modelo é obtido através da predição da amostra n do i-ésimo quadro a partir de uma combinação linear das p amostras passadas do quadro, como descrito pela equação

$$\hat{x}_i[n] = \sum_{j=1}^{p} a_j x_i[n-j], \tag{3.1}$$

onde a_j é o j-ésimo coeficiente de predição e $\hat{x}_i[n]$ é a n-ésima amostra da estimativa do i-ésimo quadro.

O erro cometido por essa representação do sinal pode ser escrito como

$$e_i[n] = x_i[n] - \hat{x}_i[n] \tag{3.2}$$

e, juntamente com os coeficientes de predição permite a obtenção do sinal original $x_i[n]$.

Tendo em vista as ressonâncias bem definidas dos sinais de áudio, o modelo adotado nesta fase (modelo autorregressivo-AR) [18] é o mais adequado para compressão desses tipos de sinais, na medida em que ressonâncias indicam a existência de pólos, e o modelo AR pode ser visto como um filtro IIR que só possui pólos em sua função de transferência.

O porquê dessa operação está ligado ao fato de o erro de predição (resíduo) possuir, tipicamente, menores amplitudes em relação ao sinal original, o que implica a utilização de menos *bits* para a codificação do resíduo que o próprio sinal original. Portanto, a aplicação da predição linear está atrelada à redução da quantidade de informação.

A seguir será feita uma breve explicação sobre como os coeficientes de predição podem ser obtidos.

3.2.1 Obtenção dos Coeficientes de Predição

Os coeficientes de predição são obtidos através da minimização do erro quadrático médio do resíduo ($e_i^2[n]$).

O algoritmo de *Levinson-Durbin* é uma solução computacionalmente eficiente para a referida operação. No entanto, ele não fornece os coeficientes de predição diretamente, mas sim os coeficientes de reflexão. Apesar disso, é possível converter estes últimos em coeficientes de predição e vice-versa [18], porque há uma equivalência entre eles. Através da equação

$$a_j[l] = a_j[l-1] + k[l]a_{l-j}[l-1] \tag{3.3}$$

onde $k[l]$ é o l-ésimo coeficiente de reflexão e $a_j[l]$ é o j-ésimo coeficiente de predição de um conjunto de coeficientes da ordem anterior, é possível obter os coeficientes de prediçao a partir dos coeficientes de reflexão.

Uma vantagem da utilização dos coeficientes de reflexão em detrimento dos coeficientes de predição propriamente ditos está no fato de os primeiros possuírem melhores características na operação de quantização (Seção 3.3) tais como menor sensibilidade, por estarem restritos ao intervalo $[-1, 1]$. Há ainda outra vantagem na utilização dos coeficientes de reflexão que, na verdade, diz respeito ao algoritmo utilizado na sua obtenção. Incrementando-se a ordem de predição em uma unidade (de p para $p+1$), por exemplo, os coeficientes da ordem corrente $(p+1)$ podem

reaproveitar todos os p coeficientes calculados da ordem anterior, o que implica uma redução da complexidade computacional.

3.2.2 Ordem de Predição

Em se tratando da ordem de predição, pode-se destacar duas possibilidades: fixa e adaptativa. A diferença entre elas está no fato de o estágio de predição linear apresentar diferentes ordens de predição para quadros distintos quando se opta pela modalidade adaptativa e iguais ordens de predição para todos os quadros quando se escolhe a modalidade fixa.

No método adaptativo, têm-se as vantagens de adequar a ordem de predição às variações estatísticas do sinal original e minimizar a quantidade de informações secundárias gastas para transmissão do conjunto de coeficientes.

Aumentando-se a ordem do preditor, reduz-se a variância do erro de predição, o que leva a umm menor taxa de *bits* R_e para o resíduo codificado. Por outro lado, a taxa de *bits* R_c para os coeficientes de predição aumenta com o número de coeficientes a serem transmitidos. Assim, a meta é encontrar a ordem ótima que minimiza a taxa de *bits* total de um quadro, que pode ser expressa pela Equação (3.4), onde K é a ordem de predição. Ressalta-se que R_e decresce com K, uma vez que a predição torna-se melhor à medida que sua ordem aumenta. Em contrapartida, R_c aumenta monotonicamente com K, porque há um aumento no número de coeficientes a serem transmitidos.

$$R_T(K) = R_e(K) + R_c(K) \tag{3.4}$$

Portanto, para a análise da ordem de predição ótima, é importante o cálculo da taxa de *bits* global R_T em cada iteração (i.e. cada ordem de predição testada K).

Dispondo de R_T, a ordem de predição ótima é obtida pela comparação entre os *bit rates* totais de cada iteração; logo, a ordem de predição ótima será aquela que possuir a menor taxa de *bits* global e que estiver entre 1 e a ordem máxima p.

3.3 Quantização dos Coeficientes de Reflexão

Este processo tem uma perda de informação intrínseca, porque a quantização trunca valores reais em uma representação com número finito de bits. Sua aproximação pode ser tão maior quanto se queira. Para isso, é necessário aumentar a quantidade de *bits* da representação. Em geral, os CODECs trabalham com 16, 24 ou 32 *bits* de precisão.

Quanto à quantização dos l coeficientes de reflexão $k_i[:]$ da Figura 3.1, há um erro de quantização implícito. Entretanto, tem-se o total controle sobre ele, pois além dos coeficientes de reflexão quantizados para o decodificador via *bitstream* são transmitidos os n erros de predição $\mathrm{eq}_i[:]$. Estes, calculados após a quantização dos coeficientes de reflexão, proporcionam a perfeita reconstrução do sinal original no decodificador, pois os coeficientes de predição calculados a partir dos coeficientes de reflexão, tanto na codificação quanto na decodificação, são idênticos.

3.4 Codificação por Entropia

A codificação por entropia procura reduzir o número de bits necessários para representar o sinal de áudio explorando redundâncias do *bitstream*. Para isso, sequências de *bits* frequentes são associadas a códigos menores, enquanto sequências mais "raras" recebem códigos maiores. Códigos de *Huffman* [19] [20], códigos de *Rice* [10] e GBMC [10] são alguns exemplos conhecidos deste processo. O *Rice coding* será detalhado na próxima seção.

3.4.1 Códigos de *Rice*

O *Rice coding* é uma técnica de codificação de entropia que codifica "símbolos" em duas partes: prefixo e sub-código. Esta última tem tamanho fixo e a primeira possui tamanho variável. Os prefixos são formados de modo que símbolos com menores e maiores magnitudes possuam, respectivamente, menos e mais *bits*.

Nota-se que isso é uma grande vantagem, pois o erro de predição possui, geralmente, amplitudes baixas; logo, tende a ser codificado com o menor número de *bits*

possível.

Um código de *Rice*[10] é definido pelo parâmetro $s \geq 0$. Para um determinado valor de s, cada palavra-código consiste de um prefixo (com m *bits*) e um sub-código (contendo s *bits*). Para se formar o prefixo, usam-se $m-1$ *bits* 1 e um *bit* 0, onde m depende do valor codificado. Para um sinal de valor x e $s > 0$, $m-1$ é calculado segundo o conjunto de Equações (3.5) [10], onde $\div$ denota divisão inteira.

$$m-1 = \begin{cases} x \div 2^{s-1} & , x \geq 0 \\ (-x-1) \div 2^{s-1} & , x < 0 \end{cases} \tag{3.5}$$

Para $s = 0$ usa-se um cálculo modificado conforme ilustra o conjunto de Equações (3.6) [10].

$$m-1 = \begin{cases} 2x & , x \geq 0 \\ -2x-1 & , x < 0 \end{cases} \tag{3.6}$$

O sub-código para $s > 0$ é calculado utilizando-se o conjunto de Equações (3.7) [10].

$$sub = \begin{cases} x - 2^{s-1}(m-1) + 2^{s-1} & , x \geq 0 \\ (-x-1) - 2^s - 1(m-1) & , x < 0 \end{cases} \tag{3.7}$$

Para $s = 0$ não há sub-código, mas somente o prefixo. Logo, a palavra-código é idêntica ao prefixo. Os valores permitidos para s pertencem ao intervalo $[0, 15]$, se a resolução for menor ou igual a 16 *bits*/amostra ou ao intevalo $[0, 31]$, se a resolução for maior que 16 *bits*/amostra.

As Tabelas 3.1 e 3.2 ilustram exemplos de códigos de *Rice* com $s = 4$ e $s = 0$, respectivamente, enquanto que a Tabela 3.3 mostra os sub-códigos com $s = 4$ para alguns valores de m.

Valores	m	**Prefixo**	**Palavra-código**
[-8, +7]	1	0	0xxxx
[-16, -9]; [+8, +15]	2	10	10xxxx
[-24, -17]; [+16, +23]	3	110	110xxxx
[-32, -25]; [+24,+31]	4	1110	1110xxxx
[-40, -33]; [+32, +39]	5	11110	11110xxxx

Tabela 3.1: Códigos de *Rice* com $s = 4$. Os *bits* xxxx contêm o sub-código de 4 *bits sub*. Fonte:[10].

Valores	m	**Prefixo**	**Palavra-código**
0	1	0	0
-1	2	10	10
+1	3	110	110
-2	4	1110	1110
+2	5	11110	11110

Tabela 3.2: Códigos de *Rice* especial para $s = 0$. Fonte:[10].

3.5 A Influência da Ordem de Predição, do Tamanho do Quadro e do Resíduo na Codificação

Quando se têm tamanhos de quadro elevados (maiores que 2048, por exemplo) com a ordem de predição pré-fixada, menos informações de coeficientes e outros parâmetros são necessárias. Por outro lado, pode ocorrer, em geral, um ligeiro aumento na amplitude do resíduo, uma vez que os coeficientes de predição calculados para o segmento podem não dar conta da descrição das variações sofridas pelo sinal no trecho.

Quanto à ordem de predição, aumentando-se seu valor e mantendo-se constante o tamanho quadro, aumenta-se também a quantidade de informação para descrever os coeficientes e reduz-se a amplitude do resíduo[1], o que acarreta menos informações

[1]Os coeficientes modelam melhor um determinado segmento de sinal de áudio à medida que se aumenta o seu número.

Valores ($m = 1$)	**Valores** ($m = 2$)	**Valores** ($m = 3$)	**sub-código (xxxx)**
-8	-16	-24	0111
-7	-15	-23	0110
-6	-14	-22	0101
-5	-13	-21	0100
-4	-12	-20	0011
-3	-11	-19	0010
-2	-10	-18	0001
-1	-9	-17	0000
0	8	16	1000
1	9	17	1001
2	10	18	1010
3	11	19	1011
4	12	20	1100
5	13	21	1101
6	14	22	1110
7	15	23	1111

Tabela 3.3: Sub-códigos do código de *Rice* com $s = 4$ para os primeiros três prefixos. Fonte:[10].

para a descrição do resíduo.

Capítulo 4

O Padrão MPEG-4 ALS

Este capítulo descreve o CODEC MPEG-4 ALS, conforme descrito na norma [10].

A escolha desse codificador como referência para uma nova implementação está ligada a fins didáticos e há grande quantidade de informações suas disponibilizadas na literatura ([10] [13] [12] são alguns exemplos de referências consultadas). Além disso, seu *software* de referência é totalmente aberto [17].

4.1 Codificador do Padrão MPEG-4 ALS

Nesta seção, será descrito o codificador MPEG-4 ALS [10]. A Figura 4.1 exibe os principais blocos do codificador. Nas seções seguintes, serão detalhados cada um desses blocos.

4.1.1 *Bitstream*

Antes da discussão dos diversos estágios do codificador do padrão MPEG-4 ALS, é necessário apresentar a disposição do *bitstream*, i.e. sinal codificado, na medida em que algumas etapas obrigatórias (como segmentação) e opcionais (como *block switching*) demandam um entendimento prévio dessa estrutura.

A Figura 4.2 ilustra a estrutura geral do *bitstream* de um arquivo de áudio comprimido de M canais.

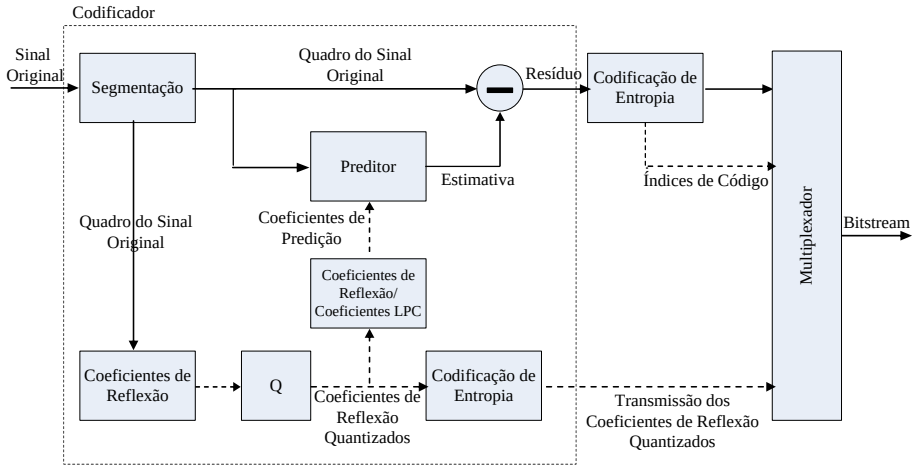

Figura 4.1: Estrutura do Codificador.

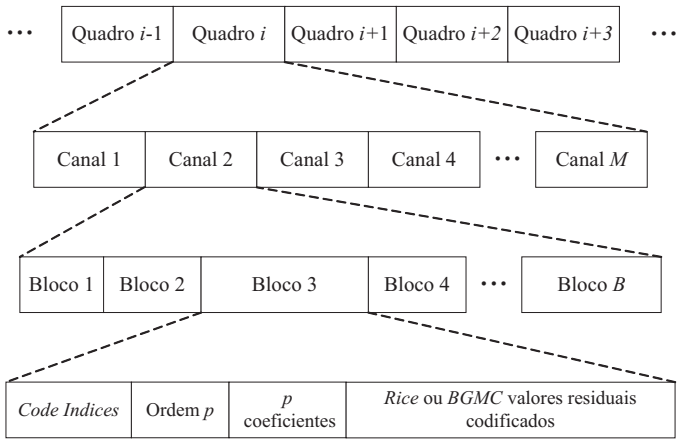

Figura 4.2: Estrutura do *bitstream*. Fonte:[10]

Cada quadro possui de 1 até 32 blocos por canal (Seção 4.1.6.3). O penúltimo nível da Figura 4.2 mostra os blocos que são efetivamente processados pelo padrão MPEG-4 ALS, o último nível representa os diversos parâmetros do sinal codificado.

Por simplicidade, e sempre que for possível[1], tratar-se-á o *bitstream* básico, no qual o *frame* corresponde a um bloco e o conjunto de todos *frames* determina um canal apenas. Isso faz com que a Figura 4.2 se resuma ao primeiro e ao último níveis. Para isso, basta a ferramenta "divisão em blocos" ou *block switching* (Seção 4.1.6.3) estar desabilitada e o sinal de entrada ter somente um canal.

4.1.2 Algoritmo de Segmentação

Como já visto na Seção 3.1, a etapa de segmentação divide o sinal de entrada em quadros. No MPEG-4 ALS, utilizam-se as seguintes equações:

$$\mathrm{nq} = \left\lceil \frac{\mathrm{na}}{N+1} \right\rceil, \tag{4.1}$$

cujo objetivo é calcular o número de quadros nq a ser processado no codificador a partir do número de amostras do sinal de entrada na, do tamanho do quadro N e do operador *ceil* $\lceil . \rceil$;

$$\mathrm{resto} = \mathrm{na} - \mathrm{nq} \cdot (N+1), \tag{4.2}$$

que calcula o tamanho do último quadro; e

$$x[h,n] = \mathrm{sinal}[Nh+n], \tag{4.3}$$

que obtém a amostra n do h-ésimo quadro a partir do sinal.

Vale lembrar que se a variável resto for diferente de zero, então é necessário incrementar o número de quadros nq em uma unidade.

[1]Só não será possível considerar o *bitstream* básico em alguns exemplos da ferramenta *block switching* e quando o sinal de entrada for multi-canal.

4.1.3 Predição Linear[1]

A operação de predição linear realizada no padrão MPEG-4 ALS segue os princípios apontados na Seção 3.2. Sua aplicação, no codificador de referência, compreende diversas etapas: geração dos coeficientes de reflexão; sua quantização e escalamento; e conversão dos mesmos em coeficientes de predição. Somente após esses passos é que se obtém efetivamente a estimativa de um segmento do sinal original.

4.1.3.1 Geração dos Coeficientes de Reflexão

O MPEG-4 ALS utiliza o mesmo princípio para geração dos coeficientes de reflexão que foi apontado na Seção 3.2.1, i.e. o algoritmo de *Levinson-Durbin*. A seguir, tem-se o Algoritmo 1 que ilustra, passo a passo, como o codificador de referência obtêm esses coeficientes.

A função janelamento do Algoritmo 1, como o próprio nome diz, executa a operação de janelamento [21] de um quadro do sinal de entrada. Embora haja quatro tipos de janelas implementadas (Hanning, retangular, Hamming e Blackman [21]), apenas a primeira está disponível para a codificação [11].

Quanto à função *Autocorrelação* do Algoritmo 1, ela diz respeito ao cálculo da autocorrelação determinística de um quadro janelado e, por conveniência, é descrita pelo Algoritmo 6 que está no Apêndice A.

Já a função *Levinson Durbin* calcula efetivamente os coeficientes de reflexão de um quadro do sinal original.

4.1.3.2 Quantização dos Coeficientes de Reflexão

Como já observado na Seção 3.3, no processo de quantização há um erro implícito. Neste caso, esse erro não é corrigido, mas propagado pelos estágios subsequentes do codificador e é informado ao decodificador, transmitido juntamente com os coeficientes de reflexão (Seção 4.1.5.1).

O processo de quantização dos coeficientes de reflexão do padrão MPEG-4 ALS é realizado utilizando-se funções específicas. Para os dois primeiros coeficientes γ_1 e

Algoritmo 1 Algoritmo de *Levinson Durbin* usado com ordem fixa [11].

quadro_janelado ⇐ Janelamento(quadro,tam_quadro) {*possiveis janelas: retangular; hamming; hanning; blackman*}

rxx ⇐ Autocorrelacao(x_janelado, tam_quadro, ordem_pred)

parCof = LevinsonDurbin(rxx, ordem_pred) {Função que gera os coeficientes de reflexão de um quadro do sinal.

Parâmetros de entrada: rxx - vetor de autocorrelação de um quadro do sinal; ordem_pred - ordem de predição.

Parâmetros internos: i, j - contadores; evar - - erro de predição para a n-ésima ordem do filtro; temp e dir - variáveis auxiliares.

Parâmetro de saída: parCof - coeficientes de reflexão.}

```
evar ⇐ rxx[0]
para (i = 1; i ≤ ordem_pred; i++) faça
  parCof[i] ⇐ -rxx[i]
  para (j = 1; j ⇐ i; j++) faça
    parCof[i] ⇐ parCof[i] - dir[j] * rxx[i-j]
  fim para
  parCof[i] ⇐ parCof[i] / evar
  dir[i] ⇐ parCof[i]
  para (j = 1; j ≤ i/2; j++) faça
    temp ⇐ dir[j] + parCof[i] * dir[i-j]
    dir[i-j] ⇐ dir[i-j] + parCof[i] * dir[j]
    dir[j] ⇐ temp
  fim para
  evar ⇐ evar * (1 - parCof[i] * parCof[i])
fim para
```

γ_2 são utilizadas as Equações (4.4) e (4.5) [10], respectivamente, onde k_1 e k_2 são o primeiro e o segundo coeficientes de reflexão quantizados, e $\lfloor . \rfloor$ denota a operação *floor*.

$$k_1 = \lfloor 64(-1 + \sqrt{2}\sqrt{\gamma_1 + 1}) \rfloor; \tag{4.4}$$

$$k_2 = \lfloor 64(-1 + \sqrt{2}\sqrt{-\gamma_2 + 1}) \rfloor. \tag{4.5}$$

A justificativa da utilização de parte dessas equações[2] ($-1 + \sqrt{2}\sqrt{\gamma_1 + 1}$ e $-1 + \sqrt{2}\sqrt{-\gamma_2 + 1}$) se deve ao fato de os valores dos primeiros coeficientes de reflexão γ_1 e γ_2 serem muito próximos a -1 e 1, respectivamente, e nas proximidades desses valores os coeficientes de reflexão possuírem maior sensibilidade na operação de quantização. Já as partes $\lfloor 64(.) \rfloor$ correspondem a quantizadores uniformes (de 7 *bits*).

Para os coeficientes γ_l ($l > 2$), que possuem amplitudes próximas a zero, são usados quantizadores uniformes (de 7 *bits*) cuja expressão é dada pela Equação (4.6) [10], onde k_l é o l-ésimo coeficiente de reflexão quantizado.

$$k_l = \lfloor 64\gamma_l \rfloor \ , \quad l > 2. \tag{4.6}$$

Ressalta-se que a faixa dos possíveis valores de k_l está compreendido entre -64 e 63, pois γ_l está compreendido no intervalo [-1,1] (veja a Seção 3.2.1).

4.1.3.3 Escalamento dos Coeficientes de Reflexão Quantizados

Antes do codificador efetivamente obter os coeficientes para o filtro de predição, é aplicada uma operação de escalamento para a reconstrução dos coeficientes de reflexão, desta vez em representação inteira em vez daqueles originais gerados pelo

[2]Essas funções são chamadas *compander*, i.e. uma combinação de um compressor e um expansor [20].

algoritmo de *Levinson Durbin*, cuja representação era real. Os dois primeiros coeficientes de reflexão escalados *par_1* e *par_2* são obtidos pelo conjunto de Equações (4.7); já os demais, são obtidos pela Equação (4.8), onde 2^Q representa uma constante de escalamento (Q = 20) requerida para representação inteira desses novos coeficientes, e $\Gamma(.)$ é um mapeamento mostrado na Tabela 4.1.

$$\begin{cases} \text{par}_1 = \lfloor \hat{\gamma}_1 2^Q \rfloor = \Gamma(k_1) \\ \text{par}_2 = \lfloor \hat{\gamma}_2 2^Q \rfloor = -\Gamma(k_2) \end{cases} \tag{4.7}$$

$$\text{par}_l = \lfloor \hat{\gamma}_l 2^Q \rfloor = k_l 2^{Q-6} + 2^{Q-7} \ , \quad (l > 2). \tag{4.8}$$

São esses coeficientes de reflexão reconstruídos (escalados) que são efetivamente convertidos em coeficientes de predição para serem empregados na predição linear.

4.1.3.4 Conversão dos Coeficientes de Reflexão em Coeficientes para o Filtro de Predição

A obtenção dos coeficientes para o filtro de predição no padrão MPEG-4 ALS obedece àqueles princípios de conversão tratados na Seção 3.2.1. Esse passo é efetivamente realizado, no codificador de referência, pelo Algoritmo 2 (que é uma implementação da Equação 3.3), que converte os coeficientes de reflexão quantizados nos coeficientes de predição.

4.1.3.5 Ordem de Predição

No MPEG-4 ALS, esse parâmetro segue todos aqueles princípios abordados na Seção 3.2.2. A exceção está no algoritmo de obtenção da ordem de predição ótima (modalidade adaptativa), pois, de acordo com [12], o algoritmo obtém a ordem de predição ótima quando o *bit rate* total que é expresso pela Equação (3.4) não decresce mais. Ou seja, o algoritmo sofre iterações até esse ponto.

Algoritmo 2 Algoritmo de conversão dos coeficientes de reflexão para coeficientes LPC. Fonte:[11][10]

cof = par2cof(par, ordem_pred) {Função de conversão dos coeficientes de reflexão em coeficientes para o filtro de predição.

Parâmetros de entrada: par - vetor contendo todos os coeficientes de reflexão de um quadro; ordem_pred - ordem de predição.

Parâmetros internos: m e i - contadores; temp e temp2 - variáveia auxiliares; Q - constante; fator - constante para escalamento.

Parâmetro de saída: cof - coeficientes para o filtro de predição.}

Q $\Leftarrow$ **20**

fator $\Leftarrow$ **1** << **(Q - 1)** {<< *é uma operação de deslocamento para esquerda de Q - 1 unidades*}

para (m = 1; m $\leq$ **ordem_pred; m++) faça**

  para (i = 1; i $\leq$ **m/2; i++) faça**

    temp $\Leftarrow$ **cof[i] + (par[m] * cof[m-i] + fator)** >> **Q** {>> *é uma operação de deslocamento para direita de Q unidades*}

    temp2 $\Leftarrow$ **cof[m-i] + (par[m] * cof[i] + fator)** >> **Q**

    cof[m-i] $\Leftarrow$ **temp2**

    cof[i] $\Leftarrow$ **temp**

  fim para

  cof[m] $\Leftarrow$ **par[m]**

fim para

l	$\Gamma(l)$	l	$\Gamma(l)$	l	$\Gamma(l)$	l	$\Gamma(l)$
-64	-1048544	-32	-913376	0	-516064	32	143392
-63	-1048288	-31	-904928	1	-499424	33	168224
-62	-1047776	-30	-896224	2	-482528	34	193312
-61	-1047008	-29	-887264	3	-465376	35	218656
-60	-1045984	-28	-878048	4	-447968	36	244256
-59	-1044704	-27	-868576	5	-430304	37	270112
-58	-1043168	-26	-858848	6	-412384	38	296224
-57	-1041376	-25	-848864	7	-394208	39	322592
-56	-1039328	-24	-838624	8	-375776	40	349216
-55	-1037024	-23	-828128	9	-357088	41	376096
-54	-1034464	-22	-817376	10	-338144	42	403232
-53	-1031648	-21	-806368	11	-318944	43	430624
-52	-1028576	-20	-795104	12	-299488	44	458272
-51	-1025248	-19	-783584	13	-279776	45	486176
-50	-1021664	-18	-771808	14	-259808	46	514336
-49	-1017824	-17	-759776	15	-239584	47	542752
-48	-1013728	-16	-747488	16	-219104	48	571424
-47	-1009376	-15	-734944	17	-198368	49	600352
-46	-1004768	-14	-722144	18	-177376	50	629536
-45	-999904	-13	-709088	19	-156128	51	658976
-44	-994784	-12	-695776	20	-134624	52	688672
-43	-989408	-11	-682208	21	-112864	53	718624
-42	-983776	-10	-668384	22	-90848	54	748832
-41	-977888	-9	-654304	23	-68576	55	779296
-40	-971744	-8	-639968	24	-46048	56	810016
-39	-965344	-7	-625376	25	-23264	57	840992
-38	-958688	-6	-610528	26	-224	58	872224
-37	-951776	-5	-595424	27	23072	59	903712
-36	-944608	-4	-580064	28	46624	60	935456
-35	-937184	-3	-564448	29	70432	61	967456
-34	-929504	-2	-548576	30	94496	62	999712
-33	-921568	-1	-532448	31	118816	63	1032224

Tabela 4.1: Coeficientes de reflexão quantizados l e os correspondentes coeficientes de reflexão escalados $\Gamma(l)$. Fonte:[10].

4.1.4 Obtenção do Resíduo

O resíduo é obtido pela comparação entre um quadro original e sua estimativa (gerada a partir da predição linear). No codificador de referência, essa operação

é realizada pelo Algoritmo 3 no MPEG-4 ALS. Nesse algoritmo, ocorre também a operação de predição linear (*loops* aninhados).

Algoritmo 3 Algoritmo de obtenção do resíduo. Fonte:[11][10]

residuo = ObtResiduo(cof, frame, tam_frame, ordem_pred) {Função que executa a obtenção do resíduo.

Parâmetros de entrada: cof - vetor com os coeficientes de predição; frame - vetor que contém um quadro do sinal de entrada; tam_frame - tamanho do frame; ordem_pred - ordem de predição.

Parâmetros internos: n e k - contadores; y - variável auxiliar; Q e fator - constantes.

Parâmetro de saída: resíduo - vetor contendo o resíduo de um quadro.}

Q $\Leftarrow$ **20**

fator $\Leftarrow$ **1** << **(Q - 1)** {<< *é uma operação de deslocamento para esquerda de Q - 1 unidades*}

para (n = 0; n $<$ **tam_frame; n++) faça**

  y $\Leftarrow$ **fator**

  para (k = 1; k $\leq$ **ordem_pred; k++) faça**

    y $\Leftarrow$ **y + (cof[k-1] * frame[n-k])**

  fim para

  residuo[n] $\Leftarrow$ **frame[n] - (y >> Q)** {>> *é uma operação de deslocamento para direita de Q unidades*}

fim para

4.1.5 Codificação por Entropia

No codificador de referência, há duas maneiras para se codificar o resíduo originado da predição linear: códigos de *Rice* (Seção 3.4.1), mais simples, e GBMC *(Block Gilbert-Moore Codes)*, mais complexa e eficiente. Como o codificador básico do MPEG-4 ALS usa os códigos de *Rice*, então apenas ele será tratado neste documento. O leitor pode encontrar informações sobre o método GBMC nas referências [11][12][13][10].

Esse recurso é utilizado tanto para a codificação (transmissão) dos coeficientes de reflexão quanto para o resíduo.

4.1.5.1 Trasmissão ou Armazenamento dos Coeficientes de Reflexão

A transmissão[3] dos coeficientes quantizados k_l deve obedecer algumas etapas. A primeira delas consiste na obtenção de δ_l (coeficientes de reflexão efetivamente trasmitidos) que é realizada pela Equação (4.9) [10], onde k_l é o l-ésimo coeficiente de reflexão quantizado e offset_l é o valor do l-ésimo parâmetro necessário para a recentralização do l-ésimo coeficiente de reflexão em torno do seu valor mais provável [12], ou seja δ_l são os coeficientes de reflexão quantizados e recentralizados.

$$\delta_l = k_l - \text{offset}_l; \tag{4.9}$$

Após isso, os valores δ_l são então codificados, por entropia, pelo uso de códigos de *Rice* (Seção 3.4.1). Os parâmetros essenciais para este último estágio estão dispostos na Tabela 4.2, onde o campo *coef_table*[4] tem influência impactante na codificação por entropia desses coeficientes, pois se seu valor for 11 (em representação binária), ela não é aplicada[5] e os coeficientes quantizados são transmitidos com 7 *bits* cada. Neste caso, o parâmetro offset é sempre -64; logo, os valores da Equação (4.10) ficam restritos ao intervalo $[0, 127]$. Caso contrário, há, de fato, aplicação da codificação por entropia; cada um dos coeficientes quantizados é transmitido com o número de *bits* informados pelas operações do código de *Rice* (Seção 3.4.1), que necessitam do valor de δ_l e das informações da Tabela 4.2, como parâmetros *Rice* (s da Seção 3.4.1) e *offsets*.

$$\delta_l = k_l + 64\ , \quad coef\_table = 11; \tag{4.10}$$

[3]Transmissão neste contexto quer dizer o envio para o decodificador, via *bitstream*.

[4]Advém do *software* do codificador de referência[11]. Seus possíveis valores são: 00, 01, 10 e 11.

[5]Este caso provavelmente é reservado para eventuais testes.

	$coef\_table = 00$		$coef\_table = 01$		$coef\_table = 10$	
Coeficiente #	*Offset*	Parâmetro *Rice*	*Offset*	Parâmetro *Rice*	*Offset*	Parâmetro *Rice*
1	-52	4	-58	3	-59	3
2	-29	5	-42	4	-45	5
3	-31	4	-46	4	-50	4
4	19	4	37	5	38	4
5	-16	4	-36	4	-39	4
6	12	3	29	4	32	4
7	-7	3	-29	4	-30	4
8	9	3	25	4	25	3
9	-5	3	-23	4	-23	3
10	6	3	20	4	20	3
11	-4	3	-17	4	-20	3
12	3	3	16	4	16	3
13	-3	2	-12	4	-13	3
14	3	2	12	3	10	3
15	-2	2	-10	4	-7	3
16	3	2	7	3	3	3
17	-1	2	-4	4	0	3
18	2	2	3	3	-1	3
19	-1	2	-1	3	2	3
20	2	2	1	3	-1	2
$2l-1,\ 10 < l < 65$	0	2	0	2	0	2
$2l,\ 10 < l < 64$	1	2	1	2	1	2
$l > 127$	0	1	0	1	0	1

Tabela 4.2: Parâmetros usados para a codificação dos coeficientes de reflexão. Fonte:[10].

4.1.6 Recursos Adicionais do MPEG-4 ALS

Nesta seção, aborda-se-ão alguns recursos adicionais do padrão MPEG-4 ALS, tais como: os modos de codificação de canais; o rearranjo dos canais; e o *block switching*.

4.1.6.1 Codificação do(s) Canal(is)

O(s) canal(is) pode(m) ser codificado(s) independente ou conjuntamente[6]. Enquanto que a primeira modalidade trata somente das redundâncias existentes em um único canal, a segunda modalidade explora a redundância que pode existir entre

[6]Somente se existir mais de um canal.

dois canais.

O *software* de referência do padrão MPEG-4 ALS[11] dispõe de um *flag* chamado *joint_stereo*, o qual é responsável pelo modo de tratamento dos canais. Se esse *flag* estiver desabilitado, cada canal se comporta de modo independente dos demais; sendo assim, o processamento é feito individualmente. Em caso contrário, o tratamento é feito com pares sucessivos de canais. Considerando ainda este último caso, se o número de canais for ímpar, o último deles é processado de modo independente.

Ressalta-se, ainda, que há dois tipos de codificação conjunta: codificação a diferença e multicodificação[10]. Nas duas estratégias, os canais originais e a diferença entre eles são necessariamente processados; porém, apenas os dois canais com menor *bit rate* são efetivamente transmitidos. Logo, o canal-diferença só é transmitido quando sua taxa de *bits* é menor que pelo menos um dos canais originais.

4.1.6.2 Rearranjo de Canais[10]

Recurso utilizado pelo codificador, somente na multi-codificação, para reorganizar os pares de canal de modo que se tenha maior correlação entre os canais de um par, com a meta de melhorar o desempenho na codificação.

Caso se deseje codificar um sinal de áudio que apresente a configuração *surround 5.1 (L, R, Lr, Rr, C, LFE)*[7], por exemplo, o modo de codificação conjunta é o mais indicado para os pares *L/R* e *Lr/Rr* e o modo independente é o melhor para os canais *C* e *LFE*, uma vez que nos primeiros pares de canal há uma forte correlação, enquanto que no último par citado, há uma fraca correlação.

Caso o *flag joint_coding*, presente no *software* do codificador de referência[11], esteja ativo, o codificador trata somente pares de canal, assim há três pares para o referido exemplo. Se a configuração for *L, R, C, Lr, Rr, LFE* ou *L, Lr, C, Rr, R, LFE*, por exemplo, há pouca redundância entre canais sucessivos, então o modo independente é mais benéfico que o modo conjunto. Contudo, é possível rearranjar os

[7]Cinco canais de som (*L*: esquerdo frontal, *R*: direito frontal, *Lr*: esquerdo traseiro, *Rr*: direito traseiro e *C*: central) e um canal para as frequências graves (*LFE*).

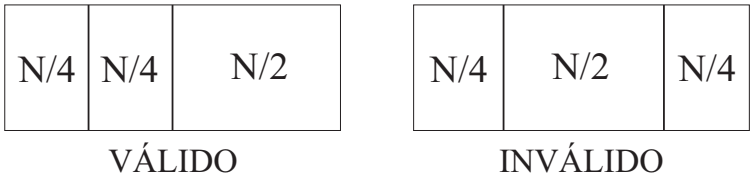

Figura 4.3: Exemplos da utilização de *block_switching*.

canais de modo a formar pares de canal com maior redundância entre si, favorecendo, desta maneira, a codificação conjunta. Vale lembrar que esse rearranjo é feito pelo usuário (as configurações são parâmetros de entrada do codificador) .

4.1.6.3 *Block Switching*

Essa operação consiste em dividir hierarquicamente cada canal de um quadro em até 32 blocos. A grande vantagem do *block switching* é permitir o uso de blocos curtos e longos no mesmo sinal de áudio. Podem-se utilizar estes últimos em segmentos estacionários combinados com preditores de ordem elevada e os primeiros em segmentos transitórios, utilizando-se para isso, preditores de ordem reduzida.

Essa divisão é feita, no *software* do codificador de referência [11], através do *flag block_switching (2 bits)*, desde que ele esteja habilitado. Caso o *flag* esteja desabilitado, cada canal comporta-se como um bloco único.

Na Tabela 4.3, têm-se a relação entre os valores de *block_switching*, o número de níveis e a quantidade máxima de blocos de um canal e na Figura 4.3 estão representados exemplos de divisão em blocos com o *flag* ativo em 01.

Valor (bits)	00	01	10	11
# níveis	*flag* inativo	1 a 3 níveis	4 níveis	5 níveis
# blocos	apenas 1	até 8	16	32

Tabela 4.3: Correspondência entre o *flag block_switching* e a quantidade de blocos para um canal. Fonte:[10].

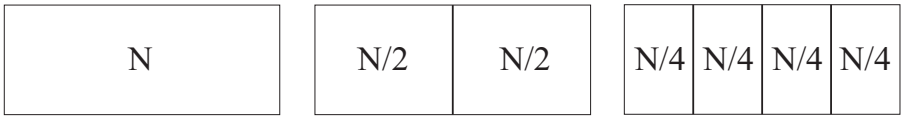

Figura 4.4: Correção do exemplo inválido de *block_switching*.

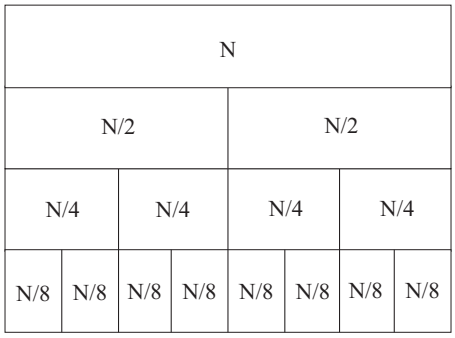

Figura 4.5: Hierarquia de níveis da ferramenta *block_switching*.

O procedimento de subdivisão de blocos só é possível em blocos oriundos de outra subdivisão (exceto para o bloco inicial de comprimento N). Para ilustrar este processo, têm-se dois exemplos mostrados na Figura 4.3. O erro no segundo exemplo está no bloco $N/2$, pois como o primeiro e o último bloco são $N/4$, então necessariamente a parte central do *frame* deveria estar subdividida em duas partes $N/4$, como sugere a Figura 4.4.

Na realidade, no *software* de referência, a decisão de particionar um determinado segmento no codificador de referência obedece a um critério que compara, a partir do último nível da Figura 4.5, a taxa de *bits* de partes (que se correspondem em relação ao tamanho do quadro) de dois níveis adjacentes de modo que sua opção sempre favoreça a parte que possui a menor taxa de *bits*. Suponha-se, por exemplo, que o codificador de referência deva optar por uma das partes hachuradas na Figura 4.6 (note que elas são correspondentes) e que a parte correspondente ao nível mais acima possui a menor taxa de *bits* em relação ao nível mais abaixo. Nesse caso, não haverá a partição do segmento correspondente a parte hachurada do primeiro nível.

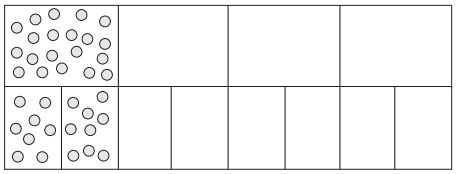

Figura 4.6: Exemplo da decisão de particionar segmentos no codificador de referência.

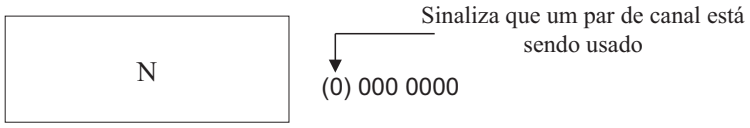

Figura 4.7: Exemplos da utilização de *bs_info* - bloco básico.

Há, ainda, outro campo importante no processo de divisão em blocos: o *bs_info*. A partição atual é sinalizada por esse campo, que depende do número de níveis de *block_switching*, como mostra a Tabela 4.4. A sequência de Figuras 4.7, 4.8, 4.9 e 4.10 ilustra exemplos da utilização do campo *bs_info*.

block_switching (bits)	00	01	10	11
# bits de bs_info	-	8	16	32

Tabela 4.4: Correspondência entre os campos *block_switching* e *bs_info*. Fonte:[10].

Em cada *frame*, os campos *bs_info* são transmitidos para todos os pares de canal e todos os canais simples, habilitando assim, uma divisão independente para diferentes canais. Assim, enquanto que o tamanho do *frame* é idêntico para todos os canais, a divisão em blocos pode ser feita individualmente para cada canal.

Se a codificação a diferença é utilizada, ambos os canais de um par devem utilizar a mesma divisão em blocos, enquanto que outros pares de canal podem usar outras configurações de divisão em blocos.

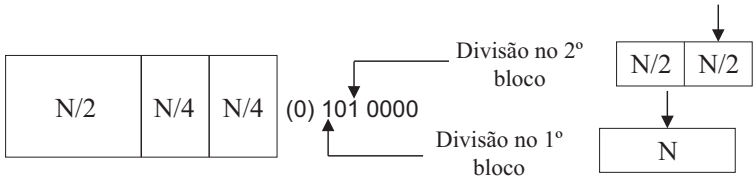

Figura 4.8: Exemplos da utilização de *bs_info* - primeiro caso de divisão em dois níveis.

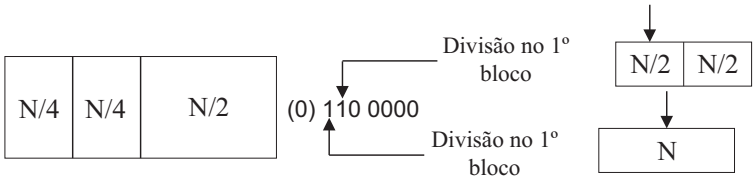

Figura 4.9: Exemplos da utilização de *bs_info* - primeiro caso de divisão em dois níveis.

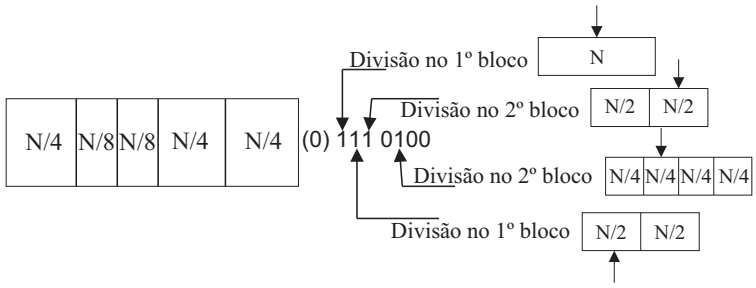

Figura 4.10: Exemplos da utilização de *bs_info* - caso de divisão em três níveis.

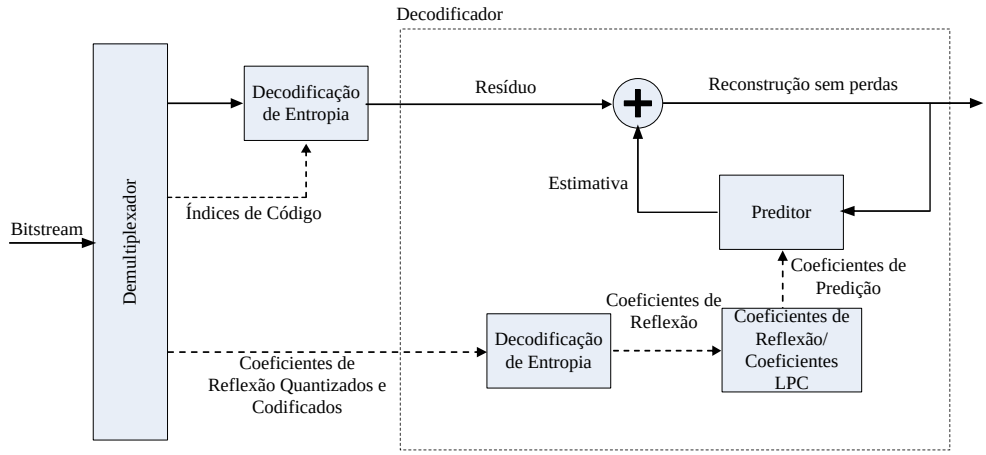

Figura 4.11: Estrutura do decodificador.

A informação associada à aplicação da divisão em blocos a um canal de um par ou ao par de canais como um todo está presente no primeiro *bit* do campo *bs_info*: se o 1º *bit* for (0), *bs_info* corresponde a um par de canais; se o 1º *bit* for (1), o campo corresponde a um único canal, mais precisamente o primeiro de um par. Neste caso, há a necessidade de outro campo *bs_info* para o segundo canal do par. Há alguns exemplos da utilização do campo *bs_info* nas Figuras 4.7, 4.8, 4.9 e 4.10.

4.2 Decodificador do Padrão MPEG-4 ALS

O processo de decodificação consiste na obtenção do sinal original através da predição do sinal codificado. Em seguida, o resíduo é somado ao sinal predito, obtendo-se uma réplica do sinal que foi codificado. A Figura 4.11 ilustra esse processo.

Ressalta-se que o decodificador, em geral, é menos complexo que o codificador, uma vez que as informações necessárias para este processo não precisam ser cal-

culadas novamente. Em suma, o esforço computacional do decodificador depende essencialmente da ordem do preditor utilizada no codificador, pois quanto maior o valor desse parâmetro, mais iterações serão demandadas pelas operações de decodificação.

No estágio de decodificação por entropia, utilizam-se os prefixos e sub-códigos (dicionário) de cada informação do *bitstream* codificado para efetivamente obter o resíduo e os coeficientes de reflexão decodificados a serem empregados na etapa seguinte, que é a predição linear.

Após o estágio de decodificação por entropia dos coeficientes quantizados e transmitidos δ_l (veja Figura 4.11), é necessário realizar a reconstrução dos coeficientes de reflexão. Para esse fim, primeiramente, executa-se a combinação entre δ_l (coeficientes de reflexão quantizados e transmitidos) e os *offsets*, que são ilustrados na Tabela 4.2, para a geração dos coeficientes de reflexão quantizados k_l (que são os mesmo calculados na codificação), segundo a equação

$$k_l = \delta_l - \text{offset}_l. \tag{4.11}$$

Em seguida, são realizadas as mesma operações da Seção 4.1.3.3 para o escalamento dos coeficientes de reflexão. Os dois primeiros coeficientes de reflexão par_1 e par_2 são calculados pelo conjunto de Equações (4.7).

A reconstrução dos coeficientes de 3ª ordem ou ordens superiores é feita pela Equação (4.8).

Após essas etapas, os coeficientes de reflexão são transformados em coeficientes para o filtro de predição, através do Algoritmo 2.

A obtenção do sinal decodificado (idêntico ao sinal original pré-codificado) é executada pelo Algoritmo 4.

4.3 Considerações Finais

Conclui-se que o MPEG-4 ALS, através de seus inúmeros recursos, abordados no corrente documento, possibilita uma codificação sem perda de informação, na medida em que é possível recuperar o sinal original a partir de uma decodificação das amostras pós-processadas pelo codificador. Como já visto, o CODEC dispõe, ainda, de alguns recursos especiais, cujo objetivo principal é melhorar o desempenho: predição de longa duração, ordem de predição adaptativa e divisão em blocos e outros que não foram tratados neste trabalho (suporte a sinais de áudio com representação em ponto flutuante [12][11] e predição sequencial utilizando filtros adaptativos [22], por exemplo). Há uma grande diversidade de informações acerca desse CODEC na literatura, o que facilita o estudo de novas implementações.

Algoritmo 4 Algoritmo de obtenção de um quadro decodificado. Fonte:[11][10]

quadro_original = ObtSinal(cof, residuo, tam_quadro, ordem_pred)
{Parâmetros de entrada: cof - vetor com os coeficientes de predição; residuo - vetor que contém o residuo de um quadro; tam_quadro - tamanho do quadro; ordem_pred - ordem de predição.
Parâmetros internos: n e k - contadores; y - variável auxiliar; Q e fator - constantes.
Parâmetro de saída: quadro_original - vetor contendo um quadro do sinal original.}
Q ⇐ 20
fator ⇐ 1 << (Q - 1) {<< *é uma operação de deslocamento para esquerda de Q - 1 unidades*}
para (n = 0; n < tam_frame; n++) faça
  y ⇐ fator
  para (k = 1; k ≤ ordem_pred; k++) faça
    y ⇐ y + (cof[k-1] * quadro_original[n-k])
  fim para
  quadro_original[n] ⇐ residuo[n] - (y >> Q) {>> *é uma operação de deslocamento para direita de Q unidades*}
fim para

Capítulo 5

MILC

Neste capítulo, será detalhada a implementação de um codificador sem perdas compatível com o padrão MPEG-4 ALS. Tal codificador é chamado MILC (Michel Igor Lossless CODEC) e possui as ferramentas básicas do MPEG-4 ALS descritas no Capítulo 4 e algumas das ferramentas avançadas.

O MILC foi implementado para fins didáticos, ou seja, sua finalidade está relacionada ao estudo de codificadores em geral e à aplicação de conhecimentos teóricos tais como quantização, predição linear e codificaçao por entropia. Essa implementação não se destina a usuários que desejam simplesmente comprimir seus arquivos de áudio pessoais, mas sim aqueles que desejam se familiarizar com os princípios básicos de codificação sem perdas.

5.1 Metodologia da Implementação

É de suma importância se pensar em métodos que simplifiquem ao máximo o desenrolar de um projeto. Tendo isso em vista, a metodologia usada na implementação do MILC seguiu uma sequência na qual primeiramente implementaram-se as ferramentas básicas como segmentação, predição linear e codificação por entropia, uma a uma, tanto na codificação quanto na decodificação, para que fosse possível corrigir os erros de implementação, conforme surgiam. Ressalta-se que os testes de validação tornavam-se mais complicados à medida que os recursos eram agregados ao programa principal.

Excetuando-se algumas funções específicas, tais como codificação por entropia, obtenção dos coeficientes de reflexão e predição e autocorrelação estatística, que foram implementadas em C para melhorar seu desempenho, o MILC foi implementado em MATLAB®. A integração entre essas duas linguagens foi efetuada através de *mex files* [23], que são uma ferramenta do MATLAB.

5.2 Princípios de Funcionamento do MILC

Os diagramas de blocos do codificador e do decodificador são idênticos aos apresentados nas Figuras 4.1 e 4.11, respectivamente, e o funcionamento de ambos segue as descrições mostradas nas Seções 4.1 e 4.2. As ferramentas básicas consistem em segmentação (Seção 4.1.2), predição linear (Seção 4.1.3), codificação por entropia (códigos de *Rice* - Seção 4.1.5) e codificação de arquivos de áudio com apenas um canal (*mono*). Esses recursos são essenciais no processo de codificação/decodificação de áudio sem perdas, o que exige uma atenção especial a suas implementações; os algoritmos utilizados são, de fato, equivalentes aos usados no padrão MPEG-4 ALS.

Posteriormente, incluiram-se as *features*: codificação de arquivos com dois canais estéreo (utilizando-se codificação independente ou conjunta de canais-Seção 4.1.6.1) e ordem de predição adaptativa. A primeira alinha-se ao algoritmo do *software* de referência do padrão estudado, enquanto que a segunda não, pois não está incluído seu código-fonte[1]; estavam disponíveis apenas algoritmos em alto nível, nas referências [12] e [10]. O Algoritmo 5 (que usa uma variação do algoritmo de *Levinson Durbin* que se encontra no Algoritmo 7 do Apêncice A) mostra como foi implementado o referido recurso no MILC.

5.3 Limitações

O MILC é limitado em comparação ao codificador de referência, na medida em que há uma série de restrições para o seu funcionamento e o número de ferramentas utilizadas é bem inferior ao MPEG-4 ALS.

[1]O codificador de referência utiliza arquivos com extensão ".obj" para aplicar o recurso de ordem de predição adaptativa.

Algoritmo 5 Algoritmo de implementação da *feature* ordem de predição adaptativa. Fontes: [12] e [10]

```
frame_janelado ⇐ Janelamento(frame,tam_frame) {Hamming}
indice ⇐ 1 {ordem de predição corrente}
enquanto (indice ≤ ordMax) faça
    rxx ⇐ Autocorrelacao(x_janelado, tam_frame, indice) {rxx tem dimensão igual
    a indice + 1}
    coefs_reflexao ⇐ DurbinAdapt(indice, rxx) {O Algoritmo 7 corresponde a esta
    função que é uma variação da Durbin anteriormente usada.}
    residuo ⇐ GetResidual(frame,indice,coefs) {Esta função segue o Algoritmo 3}
    bitRateResiduo ⇐ RiceEncode(residuo) {Codificação do resíduo}
    bitRateCoefs ⇐ RiceEncode(coefs) {Codificação dos coeficientes}
    bitRateAtual ⇐ bitRateResiduo + bitRateCoefs
    se (indice > 1) então
        se (bitRateAnterior > bitRateAtual) então
            optP ⇐ indice {Ordem de predição ótima}
            bitRateAnt ⇐ bitRateAtual
        fim se
    senão
        optP ⇐ indice
        bitRateAnt ⇐ bitRateAtual
    fim se
    indice ⇐ indice + 1
fim enquanto
```

5.3.1 Limitações Quanto ao Funcionamento

O MILC só pode ser executado em sistemas operacionais Windows (plataformas de 32 *bits*) que contenham o *software* MATLAB (versões 7.0 ou superiores) devidamente instalado.

5.3.2 Limitações Quanto às Ferramentas

Assim como o codificador de referência, o MILC aceita tamanhos de *frame* entre 32 e 65535 e ordem máxima de predição igual a 1023.

Já para frequências de amostragem, aceitam-se valores até 48 kHz (o padrão MPEG-4 ALS trabalha também com frequências mais elevadas). Esse parâmetro impacta na escolha do campo coef_table e, consequentemente, na escolha dos parâmetros s (ou *Rice*) de codificação por entropia dos coeficientes de reflexão (veja a Tabela 4.2). A Tabela 5.1 mostra a correspondência entre a frequência de amostragem e o campo coef_table.

frequência de amostragem (kHz)	**coef_table (representação binária)**
$]0, 48]$	00
$]48, 96]$	01
$]96, 192]$	10

Tabela 5.1: Correspondência entre a frequência de amostragem e o campo coef_table.

As resoluções compatíveis com a nova implementação são 16, 24 e 32, ao passo que o codificador de referência trabalha com resoluções de 8, 16, 24 e 32 *bits*/amostra. Os testes executados com ambos os codificadores utilizaram sinais com 16 *bits*/amostra de resolução.

Em se tratando dos recursos implementados no MILC, podem-se citar: codificação de sinais com um e dois canais (neste último caso, a codificação pode ser efetuada independente ou conjuntamente, conforme se viu na Seção 4.1.6.1), ordem de predição fixa e adaptativa; codificação por entropia através de códigos de *Rice*.

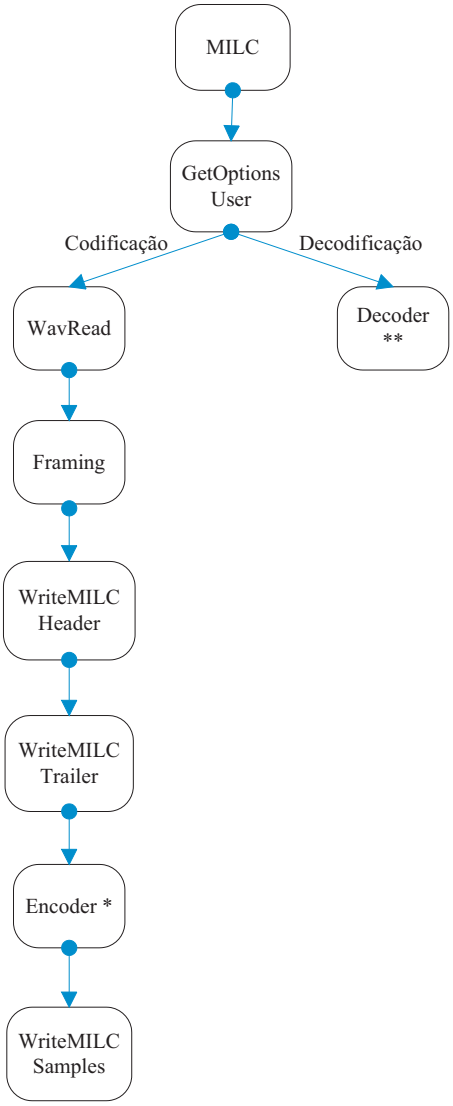

Figura 5.1: Fluxograma do *software* MILC.

5.4 Organização do *Software* MILC

Esta seção trata do modo como o MILC foi implementado. A seguir, têm-se as principais rotinas que o compõem.

5.4.1 Rotina Principal

A parte principal do *software* foi implementada segundo o fluxograma da Figura 5.1.

As funções dos blocos apresentados na referida figura são:

- `MILC`: Função principal cujo parâmetro de entrada é uma *string* que corresponde às opções do usuário. A utilização desta função é abordada na Seção 5.5.

- `GetOptionsUser`: Função que trata as opções de codifição ou decodificação do usuário. Seu parâmetro de entrada é a *string* passada pelo usuário na função principal (MILC) e seus parâmetros de saída são: as opções propriamente ditas (devidamente separadas), o nome do arquivo de entrada, o nome do arquivo de saída e a extensão do arquivo de entrada.

- `WavRead`: Função que lê as amostras originais (".wav") do arquivo de entrada. Esta é uma função do próprio MATLAB.

- `Framing`: Parte da função "MILC" (na codificação) e "ReadSamples" (na decodificação) que particiona o sinal em segmentos para ser processado.

- `WriteMILCHeader`: Função que escreve o cabeçalho do arquivo de saída (".nlc"). Seus parâmetros de entrada são: o número de amostras, a resolução, uma estrutura contendo as opções do usuário, o tamanho das informações secundárias (artista, compositor, álbum, ano etc.) e o cabeçalho original. Já os seus parâmetros de saída são, respectivamente, o *bitstream* do cabeçalho (que na verdade é um vetor com as informações a serem escritas no cabeçalho) e o número de bits de cada informação contida no cabeçalho.

- `WriteMILTrailer`: Esta função apenas salva informações secundárias tais como nome, gênero, compositor etc.

- `Encoder`: Função que codifica, segmento por segmento, as amostra originais do arquivo de entrada. Seus parâmetros de entrada são o *frame*, a estrutura com as opções de codificação do usuário, a frequência de amostragem usada, a resolução, o nome do arquivo de saída, e um conjunto de amostras do *frame* anterior necessárias no estágio de predição linear. Sua saída é composta do *bitstream* com as amostras codificadas do *frame* e o erro de predição. Existe um fluxograma próprio para explicar esta função que pode ser visto na Figura 5.2.

- `WriteMILCSamples`: Função que escreve as amostras codificadas no arquivo de saída. Esta função é implementada a partir de comandos "fwrite" do MATLAB.

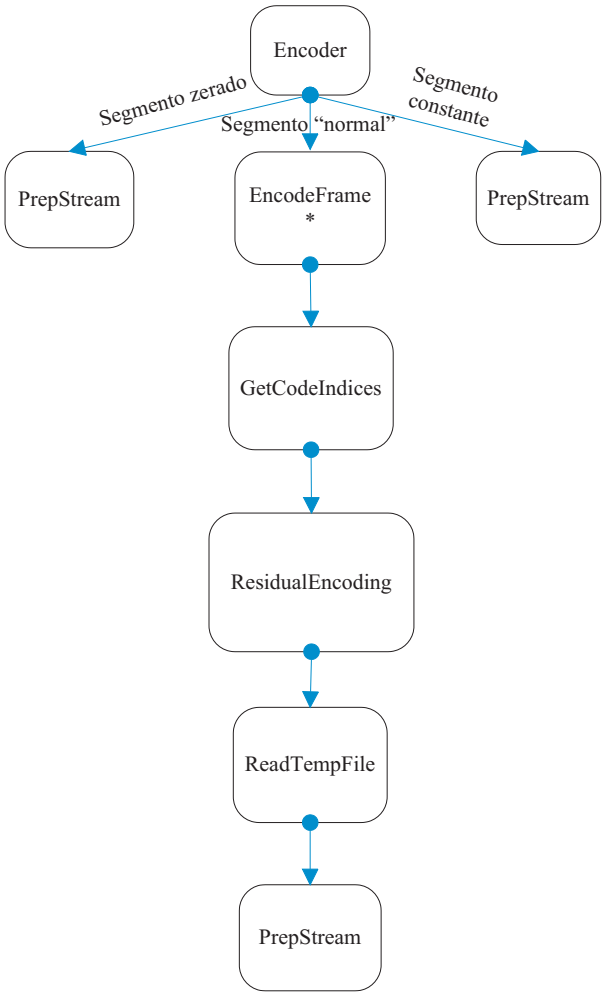

Figura 5.2: Fluxograma do codificador do *software* MILC.

- `Decoder`: Função que decodifica arquivos ".nlc" (MILC) em arquivos do tipo ".wav". Existem só dois parâmetros de entrada: o nome do arquivo de entrada e o nome do arquivo de saída. Devido à grande complexidade desta função, há um fluxograma exclusivo que ilustra a sua implementação (Figura 5.4).

5.4.2 Rotina do Codificador

A Figura 5.2 ilustra a implementação do codificador do MILC.

Nessa figura há diversos blocos cujas funções são:

- `EncodeFrame`: Função que calcula os coeficientes de predição e executa o estágio de predição linear. Seus parâmetros de entrada são: o quadro, a es-

trutura com as opções do usuário e as n últimas amostras do quadro anterior (onde n é a ordem de predição). Dentre os parâmetros de saída, têm-se: o resíduo, os coeficientes de predição e o número de *bits* necessários para cada um deles e a ordem de predição. Os detalhes desta função podem ser vistos na Figura 5.3.

- `GetCodeIndices`: Na realidade isto não é propriamente uma função, mas um trecho da função Encoder que executa o cálculo dos índices de código (parâmetros Seção 3.4.1), que são parâmetros de entrada no estágio de codificação por entropia.

- `ResidualEncoding`: Função que realiza a codificação por entropia do resíduo calculado na predição linear. Existem somente três parâmetros de entrada: o resíduo, o tamanho do resíduo e os índices de código *(code indices)*. Ao invés de apresentar parâmetros de saída, esta função executa a escrita de um arquivo temporário contendo o *bitstream* das amostras codificadas.

- `PrepStream`: Função que ajusta o *bitstream* das amostras codificadas. Ela possui como parâmetros de entrada o total de *bytes* necessários para o referido ajuste, o *bitstream* lido em "ReadTempFile", e a quantidade de *bits* necessários para a codificação de cada uma das amostras. Existe apenas um parâmetro de saída, que corresponde ao *bitstream* ajustado e pronto para ser escrito no arquivo de saída. A disposição das amostras codificadas no *bitstream* pode ser vista na Figura 4.2.

5.4.3 Codificação dos Quadros

O fluxograma mostrado na Figura 5.3 mostra como é implementada a codificação dos quadros.

As funções das partes integrantes dessa rotina encontram-se a seguir.

- `GetCofOptOrder`: Função correspondente ao Algoritmo 5 que calcula a ordem de predição ótima, os coeficientes de reflexão e o resíduo quando se escolhe o modo de predição linear adaptativa. Seus parâmetros de entrada são: o quadro, o tamanho do quadro, a ordem de predição máxima (n) e as n últimas

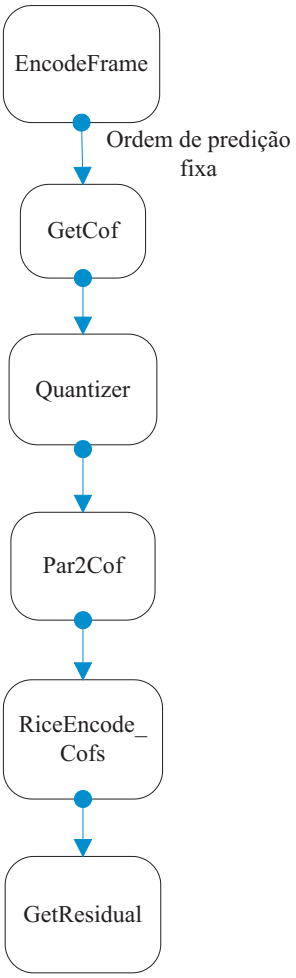

Figura 5.3: Fluxograma de codificação do quadro do *software* MILC.

amostras do frame anterior. Os parâmetros de saída correspondem à ordem de predição ótima, os coeficientes de reflexão e o resíduo.

- `GetCof`: Função de obtenção dos coeficientes de reflexão cujos parâmetros de entrada correspondem ao quadro, ao seu tamanho e à ordem de predição, e cujo único parâmetro de saída é um vetor contendo os coeficientes de reflexão.

- `Quantizer`: Função que quantiza os coeficientes de reflexão. Ela recebe os coeficientes não quantizados e os retorna quantizados.

- `Par2Cof`: Função de conversão dos coeficientes de reflexão em coeficientes LPC próprios para o filtro de predição. Possui como parâmetros de entrada os coeficientes de reflexão e a ordem de predição, e o único parâmetro de saída é um vetor com os coeficientes LPC. O Algoritmo 2 ilustra a implementação desta função.

- `RiceEncode_cofs`: Função que realiza a codificação por entropia dos coeficientes de reflexão para transmissão. Ela recebe os coeficientes de reflexão e os índices de código dos mesmos coeficientes (veja a Seção 3.4.1) e retorna os coeficientes de reflexão codificados.

- `GetResidual`: Função de obtenção do resíduo, ou seja, que realiza a predição do sinal utilizando os coeficientes quantizados. Seus parâmetros de entrada são o quadro, o seu tamanho, a ordem de predição n, os coeficientes LPC e as n últimas amostras do *frame* anterior. O parâmetro de saída é o resíduo. O Algoritmo 3 ilustra a implementação desta função.

5.4.4 Rotina do Decodificador

A decodificação no MILC foi implementada segundo o fluxograma apresentado na Figura 5.4. Nessa figura, estão representados alguns blocos, cujas funções são apresentadas abaixo.

- `ReadHeader`: Função que lê o cabeçalho do arquivo codificado (arquivo de entrada com extensão ".nlc"). Seu único parâmetro de entrada é o *bitstream* codificado, e seu parâmetro de saída é uma estrutura com as informações de codificação.

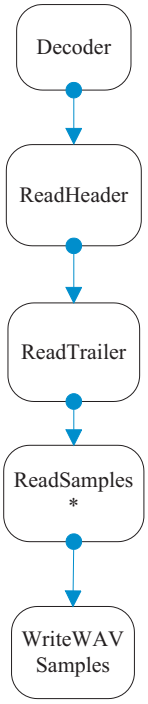

Figura 5.4: Fluxograma do decodificador do *software* MILC.

- `ReadTrailer`: Trecho da função "Decoder" que lê as informações secundárias (nome do arquivo, gênero etc.) do arquivo codificado.
- `ReadSamples`: Função que lê as amostras codificadas. Ela recebe o *bitstream* codificado e a estrutura com as informações de codificação e retorna as amostras decodificadas idênticas às pré-codificadas. A Figura 5.5 ilustra o fluxograma desta função.
- `WriteWAVSamples`: Parte da função "Decoder" que realiza a escrita do arquivo decodificado ".wav".

5.4.5 Rotina de Leitura das Amostras Codificadas

A leitura das amostras, no decodificador, é feita tal como ilustra o fluxograma da Figura 5.5.

A seguir, há a função de cada uma das partes que integram essa rotina.

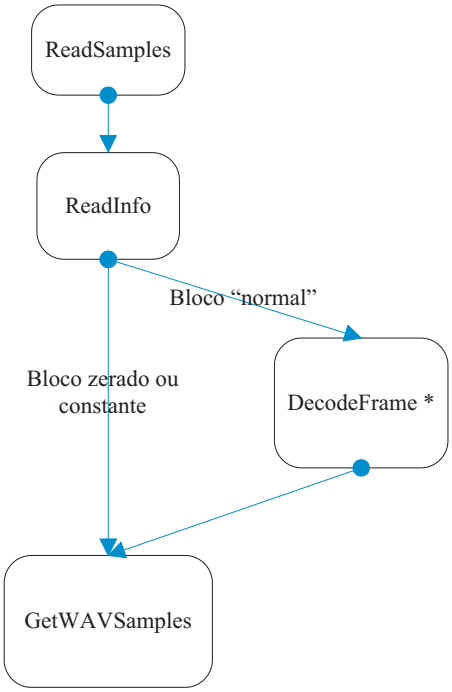

Figura 5.5: Fluxograma de leitura das amostras codificadas do *software* MILC.

- `ReadInfo`: Função que decodifica o resíduo e separa as informações *(bits)* dos coeficientes de reflexão. Esta função recebe uma série de parâmetros de entrada, tais como o *bitstream*, o índice atual no *bitstream* e a estrutura com opções de codificação; existem também muitos parâmetros de saída, sendo o principal deles o resíduo.

- `DecodeFrame`: Função que obtém as amostras originais através do processo de predição linear. Em resumo, esta função recebe o resíduo e os coeficientes de predição e retorna o quadro com amostras decodificadas. A Figura 5.6 mostra o fluxograma desta função.

- `GetWAVSamples`: Trecho da função "ReadSamples" para obtenção do *bitstream* ".wav".

5.4.6 Rotina de Decodificação de Quadros

Esta rotina foi implementada conforme a Figura 5.6. Abaixo, têm-se as funções correspondentes dos blocos ilustrados na figura.

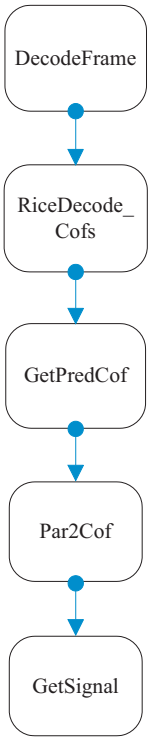

Figura 5.6: Fluxograma de decodificação do quadro do *software* MILC.

- `RiceDecode_Cofs`: Função que realiza efetivamente a decodificação dos coeficientes de reflexão. Seus parâmetros de entrada são o resíduo codificado e os índices de código, e seu parâmetro de saída é o resíduo decodificado.

- `GetPredCof`: Função que obtém os coeficientes de reflexão quantizados. Ela recebe os coeficientes de reflexão quantizados com o *offset* de transmissão e retorna os coeficientes de reflexão quantizados sem o *offset*.

- `GetSignal`: Função de obtenção do quadro decodificado. Esta função recebe o resíduo, o tamanho do resíduo, a ordem de predição (n) e as n últimas amostras decodificadas do quadro anterior e retorna o quadro decodificado. Há mais detalhes desta função no Algoritmo 4.

5.5 Utilização

O uso do MILC é feito pela linha de comando do MATLAB. Para sua execução, basta digitar: "MILC [opcoes de codificação ou decodificação] [caminho completo

do arquivo de entrada] [caminho completo do arquivo de saída]". Como opções de codificação ou decodificação, têm-se:

- "-n#" para a opção do tamanho do *frame*, onde "#" indica o seu tamanho propriamente dito (*default*: # = 2028);
- "-o#" para a opção de ordem de predição, onde "#" significa a ordem escolhida (*default*: # = 10);
- "-a" para a opção de ordem de predição adaptativa (*default*: ordem de predição fixa);
- "-i" para a opção de codificação na modalidade independente (*default*: modo conjunto);
- "-x" para a opção de decodificação (*default*: codificação).

Vale lembrar que todas as opções são de codificação, exceto "-x", que é opção de decodificação, e o ocultamento do caminho do arquivo de saída implica um nome equivalente ao arquivo de entrada, exceto pela extensão que passa a ser ".nlc".

Capítulo 6

Testes

Este capítulo apresenta os testes realizados no *software* do codificador de referência do padrão MPEG-4 ALS [11] e do MILC em um Intel® Pentium® Dual T2390 de 1,86 GHz e 2 GB de memória RAM.

Esses testes têm por objetivo avaliar as taxas e os tempos de codificação para variações de parâmetros tais como tamanho do quadro, ordem de predição, utilização ou não de *block switching* (no caso do MPEG-4 ALS) e codificação de canais de modo independente.

No caso do codificador de referência, a intenção desses testes é analisar o impacto de algumas ferramentas nos tempos e taxas de codificação.

Em se tratando do MILC, a meta é avaliar se, de fato, o MILC é compatível com o codificador de referência.

6.1 Testes do MPEG-4 ALS

Esta seção corresponde aos testes realizados no MPEG-4 ALS que serão explicitados nas seções adiante.

6.1.1 Análise do Tamanho do Quadro

Neste teste, variou-se o tamanho dos quadros conforme a legenda das Figuras 6.2 e 6.1 e mantiveram-se constantes ou desabilitados os demais parâmetros relevantes

na codificação (Tabelas 6.1 e A.1).

parâmetro	**valor ou *status***
tamanho do quadro	variável conforme a Figura 6.2
ordem de predição	10
block switching	desabilitado
ordem adaptativa	desabilitado
codificação independente	desabilitado

Tabela 6.1: Parâmetros utilizados no teste do tamanho do quadro.

Na Figura 6.1, percebe-se que, em geral, quanto maior o tamanho do quadro, menor é o tempo de codificação. A justifcativa para isso é a redução de iterações na simulação, que, por sua vez, ocorre devido à redução do número de coeficientes calculados para o filtro de predição, dos índices de código (parâmetros s) e de outros parâmetros do codificador do padrão MPEG-4 ALS.

Comparando-se as difereças entre os arquivos, nota-se que apenas o arquivo "cravo-mono" apresenta um tempo significativamente inferior aos demais. O porquê desse comportamento está no fato de esse arquivo possuir apenas um canal.

A Figura 6.2 ilustra um gráfico contendo as taxas de codificação para cada arquivo utilizando-se os respectivos tamanhos de quadro. Analisando-se arquivo por arquivo, há uma semelhança notável entre os resultados para um mesmo arquivo codificado, pois as taxas são muito próximas para os valores de tamanho de quadro utilizados. Portanto, o parâmetro tamanho do quadro não impacta tanto na taxa de codificação, o que só corrobora a discussão abordada na Seção 3.5, que trata da influência do tamanho do quadro na codificação.

Considerando-se, agora, o gráfico da Figura 6.2 como um todo, percebe-se que há grandes diferenças entre as taxas de codificação dos arquivos. A melhor taxa é a do arquivo "Dubal", enquanto que a pior é a do arquivo "Valsa_Chopin-Godowsky_-1916". O porquê para essa discrepância se deve às características e especificidades desses dois arquivos. "Dubal" é um arquivo de voz, portanto mais fácil de se codificar

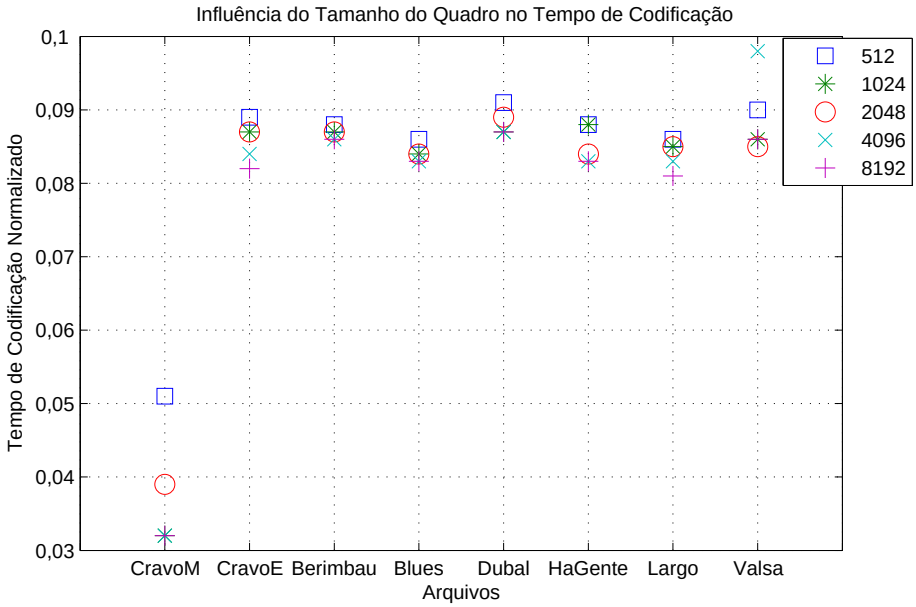

Figura 6.1: Tempo de codificação para variações no tamanho do quadro.

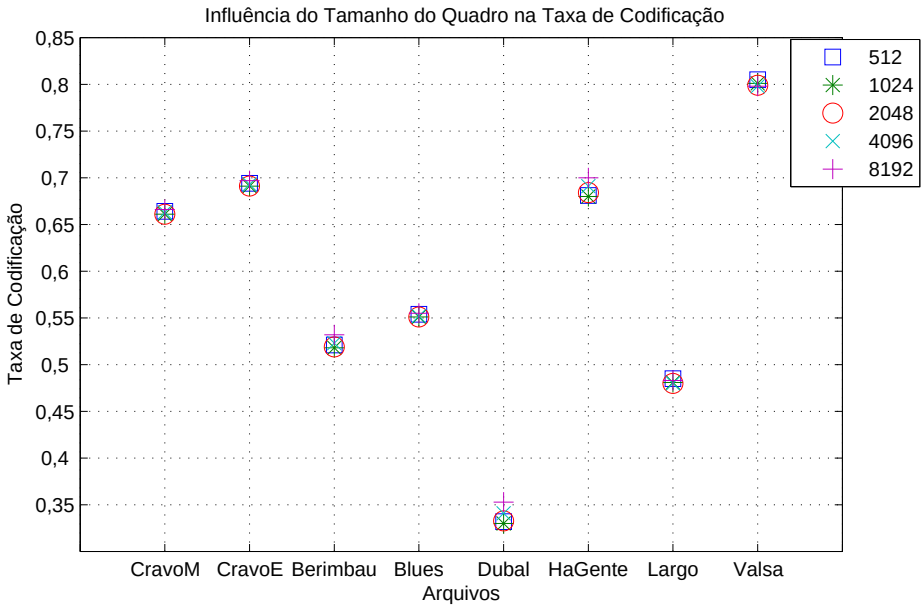

Figura 6.2: Taxa de codificação para variações no tamanho do quadro.

e “Valsa_Chopin-Godowsky_1916” é uma gravação muito antiga e ruidosa, o que implica maior dificuldade em sua codificação.

6.1.2 Análise da Ordem de Predição

Esta seção trata do teste da ordem de predição. Para cada arquivo de áudio, selecionou-se o tamanho de quadro correspondente à melhor taxa de codificação do teste da Seção 6.1.1 (Tabela 6.2), variou-se a ordem de predição segundo os valores mostrados na legenda das Figuras 6.4 e 6.3 e usaram-se os demais parâmetros conforme ilustram as Tabelas 6.3 e A.1.

arquivo	**tamanho do quadro**
Cravo-mono	1024
Cravo-estereo	2048
Berimbau	1024
BluesInG-MJQ	2048
Dubal	1024
HaGenteAqui-MJ	1024
Largo-Schuls-hv	2048
Valsa-Chopin-Godowsky-1916	8192

Tabela 6.2: Tamanhos de quadro usados no teste da ordem de predição.

parâmetro	**valor ou *status***
tamanho do quadro	variável conforme a Tabela 6.2
ordem de predição	variável conforme a legenda da Figura 6.4
block switching	desabilitado
ordem adaptativa	desabilitado com ordens 10, 30, 50 e 100 e habilitado com ordem 127
codificação independente	desabilitado

Tabela 6.3: Parâmetros utilizados no teste da ordem de predição.

Ressalta-se que 200 foi o valor inicialmente escolhido como limite máximo de ordem de predição adaptativa. Porém, como há um mecanismo de adequação da

ordem de predição máxima (Pmax) em relação ao tamanho de um determinado segmento de áudio no *software* do codificador de referência:

$$\text{Pmax} = N/8 - 1\ , \quad se\ \ \text{ord} \geq N/8, \tag{6.1}$$

onde N e ord são respectivamente o tamanho do quadro e a ordem de predição informada pelo usuário na codificação. Assim sendo, alguns arquivos tais como "Cravo-mono", "Berimbau", "Dubal" e "HaGenteAqui-MJ", que foram codificados com tamanho de quadro igual a 1024, não puderam ser codificados com o referido limite (200), uma vez que aquele mecanismo limita a ordem de predição em 127 para esses arquivos com o tamanho de quadro utilizado (segundo a Equação (6.1)). Portanto, codificaram-se todos os arquivos, na modalidade adaptativa de ordem de predição, com o valor 200 ou com o valor resultante da Equação (6.1).

A Figura 6.3 mostra o tempo de codificação para os arquivos de áudio. Nela, nota-se que para ordem de predição fixa, quanto maior o seu valor, maior é o tempo de codificação para um determinado arquivo. Quanto à ordem de predição adaptativa, para cada arquivo observa-se que o tempo de simulação é bem próximo do valor obtido com a ordem de predição correspondente à melhor taxa de codificação (Figura 6.4). Isso indica que, na modalidade adaptativa, a ordem média é próxima da melhor ordem fixa.

Quanto à discussão das diferenças dos tempos de codificação entre os arquivos, dentre os que são estéreo, não há diferenças significativas quando se comparam os arquivos com as respectivas ordens de predição fixa. Somente quando a ordem de predição é adaptativa é que são evidenciadas alguma diferenças. A justificativa para esse padrão se encontra na Seção 6.1.4.

Na Figura 6.4, apresenta-se um gráfico que contém as taxas de codificação para os arquivos de áudio. Nesse gráfico, as taxas de codificação são bem próximas para cada arquivo, o que acarreta em pouca influência da ordem de predição. A discussão abordada na Seção 3.5 ajuda a entender essa proximidade, pois há uma compensação entre o resíduo e a ordem de predição, i.e. aumentando-se a ordem de

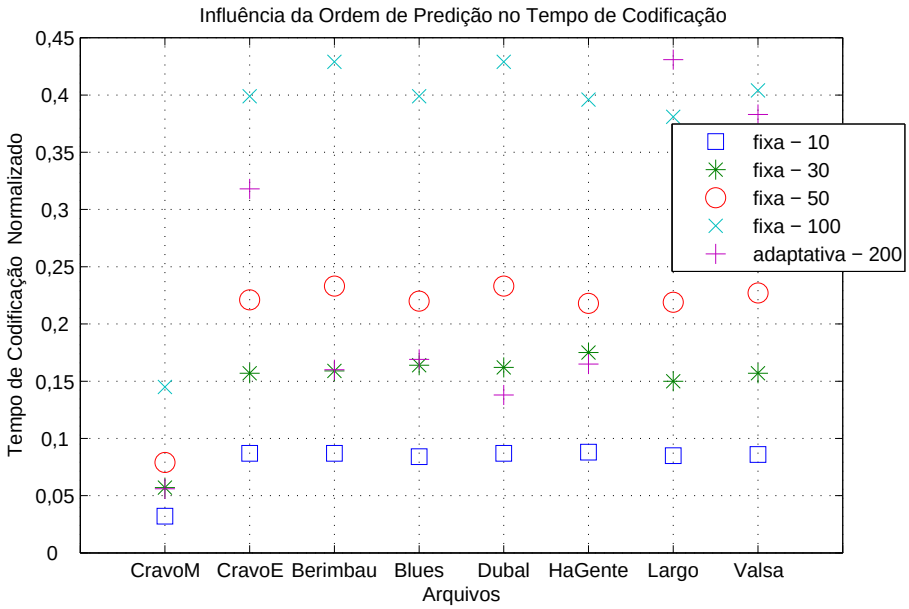

Figura 6.3: Tempo de codificação para variações na ordem de predição.

predição, aumentam-se os coeficientes a serem codificados e reduz-se a quantidade de informação do resíduo a ser codificada.

Analisando-se o gráfico da Figura 6.4 por inteiro, em conjunto com a explicação abordada na Seção 6.1.1, pode-se dizer que as diferenças encontradas entre os arquivos se devem às características dos sinais (veja a Tabela 2.2). O sinal "Valsa_-Chopin-Godowsky_1916" apresentou o pior resultado (áudio muito ruidoso) e o sinal "Dubal" apresentou o melhor resultado (sinal de voz).

Antes de encerrar esta seção, é importante mencionar que nos testes subsequentes optar-se-á pela ordem de predição fixa correspondente à melhor taxa de codificação desta rodada de testes, uma vez que, no MILC (veja Seção 6.2), a ordem de predição adaptativa apresenta maior complexidade e um dos objetivos deste capítulo é exatamente comparar os resultados obtidos com ambos os codificadores.

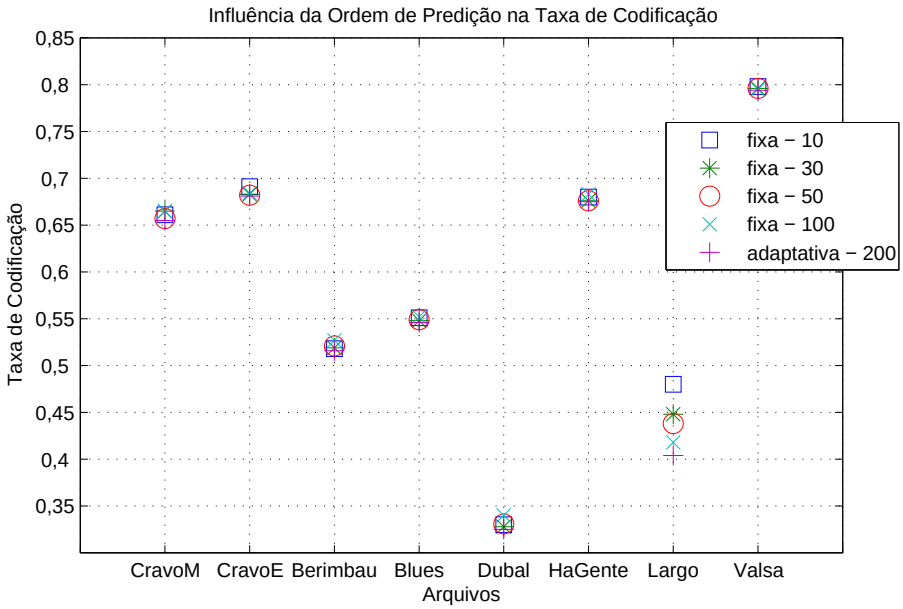

Figura 6.4: Taxa de codificação para variações na ordem de predição.

6.1.3 Análise da Utilização de *Block Switching*

Esta seção aborda o *block switching*. As Tabelas 6.2 e 6.4 mostram os parâmetros tamanho do quadro e ordem de predição usadas para cada arquivo nesta rodada de testes. As Tabelas 6.5 e A.1 ilustram os demais parâmetros utilizados.

É importante mencionar, ainda, que aquele mecanismo de adequação da ordem de predição máxima citado na Seção 6.1.2 tem grande influência na ferramenta *block switching*, na medida em que pode haver blocos com tamanho tal que satisfaça a Equação (6.1). Assim, existe uma limitação na ordem de predição máxima pela equação. A seguir, a Tabela 6.6 mostra os valores da ordem de predição utilizados pelo *software* do codificador de referência para os respectivos tamanhos de quadro (OP sistema), ordens de predição indicadas pelo usuário (OP usuário) e níveis de *block switching*.

arquivo	**ordem de predição**
Cravo-mono	30
Cravo-estereo	50
Berimbau	10
BluesInG-MJQ	30
Dubal	30
HaGenteAqui-MJ	30
Largo-Schuls-hv	100
Valsa-Chopin-Godowsky-1916	100

Tabela 6.4: Ordens de predição usadas no teste do *block switching*.

parâmetro	**valor ou *status***
tamanho do quadro	variável conforme a Tabela 6.2
ordem de predição	variável conforme a Tabela 6.4
block switching	variável conforme a legenda da Figura 6.6
ordem adaptativa	desabilitado
codificação independente	desabilitado

Tabela 6.5: Parâmetros utilizados no teste de *block switching*.

			Divisão em blocos *(Block switching)*				
			2 níveis	**4 níveis**	**8 níveis**	**16 níveis**	**32 níveis**
Arquivo	**Tamanho do *frame***	**OP usuário**	**OP sistema**	**OP sistema**	**OP sistema**	**OP sistema**	**OP sistema**
Cravo- Mono	1024	30	30	30	15	7	3
Cravo- Estereo	2048	50	50	50	31	15	7
Berimbau	1024	10	10	10	10	7	3
BluesInG_MJQ	2048	30	30	30	30	15	7
Dubal	1024	30	30	30	15	7	3
HaGenteAqui-MJ	1024	30	30	30	15	7	3
Largo_Schulz-hv	2048	100	100	63	31	15	7
Valsa_Chopin-Godowsky	8192	100	100	100	100	63	31

Tabela 6.6: Ordens de predição utilizadas pelo *software* do codificador de referência no teste de *block switching*.

A Figura 6.5 mostra um gráfico com os tempos de simulação dos arquivos de áudio usados neste teste. Nele, percebe-se que quanto mais se segmenta o sinal, maior é o tempo de codificação por arquivo de áudio. Isso é coerente, na medida em que são necessários mais cálculos de coeficientes e demais parâmetros.

Analisando-se a diferença de tempo entre os arquivos, "Valsa_Chopin-Godowsky_-1916", "Largo_Schulz-hv" e "Cravo-estereo" apresentaram os maiores tempos de codificação por nível de *block switching*. Isso se deve ao fato desses arquivos utilizarem um maior número de coeficientes por segmento (Tabela 6.6), o que implica mais iterações na codificação.

Já a Figura 6.6 mostra as taxas de codificação correspondentes aos arquivos de áudio utilizados. É notório que os valores das taxas, para cada arquivo, são praticamente os mesmos para os diversos níveis de *block switching* adotados. Isso se deve aos mecanismos de divisão desta ferramenta e aquele (adequação da ordem de predição máxima) abordado na Seção 6.1.2, uma vez que quanto mais se segmenta o sinal de áudio,mais tende a se reduzir o número de coeficientes por segmento. Desse modo, há uma compensação entre o tamanho do bloco processado e a ordem de predição considerada, a qual impacta também a complexidade, de modo a reduzi-la.

6.1.4 Análise da Codificação Independente

Aqui, a intenção é analisar o modo de codificação independente. Para isso, utilizaram-se como tamanho de quadro e ordem de predição, para os arquivos de áudio, os valores mostrados nas Tabelas 6.2 e 6.4, respectivamente. Usaram-se, ainda, os valores ilustrados nas Tabelas 6.7 e A.1 para os demais parâmetros.

Nessa rodada de testes, há somente arquivos de áudio estéreo, pois o arquivo "Cravo-mono" já foi codificado no modo independente na Seção 6.1.2. Não é possível codificá-lo na modalidade conjunta, porque há somente um canal no referido arquivo.

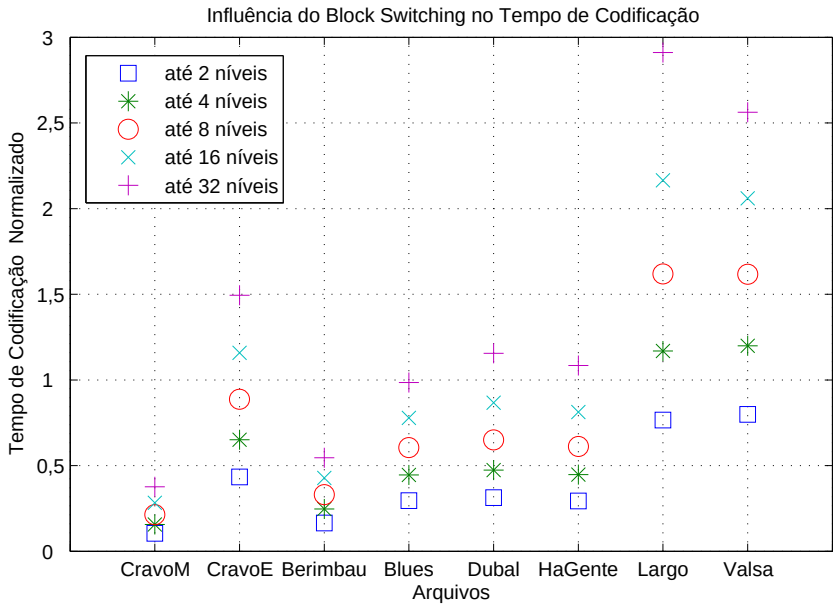

Figura 6.5: Tempo de codificação levando-se em consideração o *block switching*.

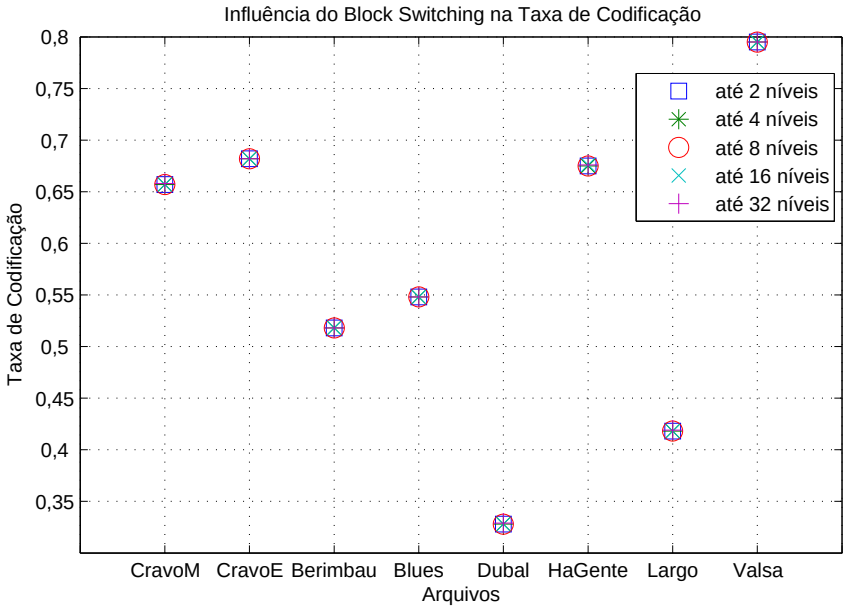

Figura 6.6: Taxa de codificação levando-se em consideração o *block switching*.

parâmetro	**valor ou *status***
tamanho do quadro	variável conforme a Tabela 6.2
ordem de predição	variável conforme a legenda da Figura 6.8 e a Tabela 6.4
block switching	desabilitado
ordem adaptativa	variável de acordo com a legenda da Figura 6.8
codificação independente	habilitado

Tabela 6.7: Parâmetros utilizados no teste da codificação independente de canais.

A Figura 6.7 mostra os tempos de codificação para cada arquivo de áudio para ordens de predição fixa (Tabela 6.4) e adaptativa. Nela, nota-se que os tempos correspondentes à ordem de predição adaptativa foi menor para uns arquivos e maior em outros em relação à modalidade fixa de ordem de predição. A justificativa para esse comportamento se deve ao fato de a ordem média no modo adaptativo ser, respectivamente, menor para alguns arquivos e maior para outros. E esse comportamento, por sua vez, está ligado a forma de implementação da ordem de predição adaptativa no codificador de referência (veja a Seção 4.1.3.5).

Quanto à taxa de codificação, a Figura 6.8 mostra seus valores para cada arquivo de áudio considerado. O comportamento segue os testes anteriores, ou seja, não há grandes discrepâncias entre as taxas de codificação para cada arquivo. Novamente, pode-se voltar à explicação dada na Seção 3.5.

Agora, comparando-se as modalidades conjunta e independente de codificação de canais, tendo como base as Figuras 6.3, 6.4, 6.7 e 6.8, pode-se apontar a melhor modalidade, considerando-se a taxa e o tempo de codificação.

Quanto à taxa de codificação, a modalidade conjunta é melhor que a modalidade independente, uma vez que seus valores para ordem de predição adaptativa e fixa (correspondentes à Tabela 6.4, que são os mesmos valores usados no teste da codificação independente) são inferiores ou equivalentes para os diversos arquivos codificados em relação ao modo independente de codificação de canais. Entretanto, seus valores são bem próximos, o que implica um ganho baixo (cerca de 0,1 %) da codificação conjunta em relação à codificação independente.

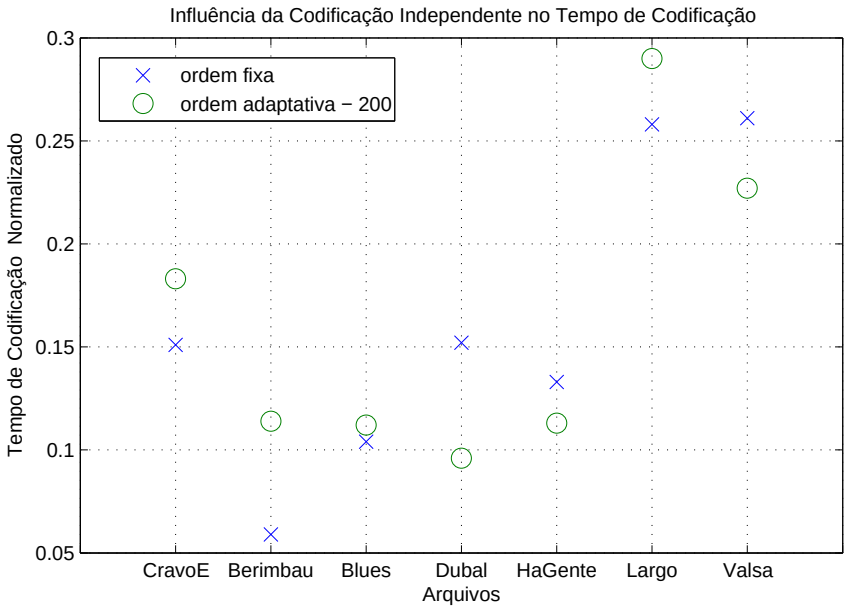

Figura 6.7: Tempo para codificação independente de canais.

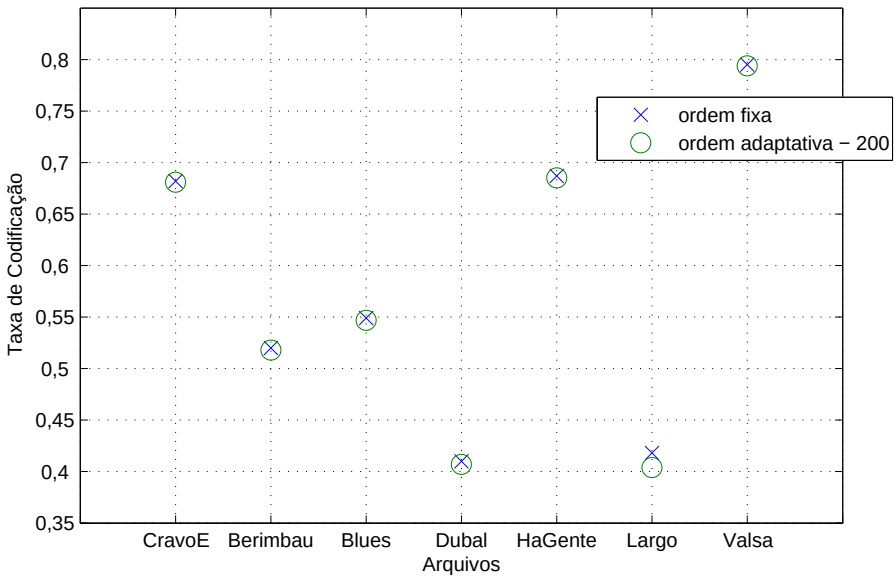

Figura 6.8: Taxa para variações codificação independente e canais.

No que diz respeito ao tempo de codificação, o modo conjunto é pior em relação ao modo fixo (veja as Figuras 6.3 e 6.7), pois na modalidade conjunta são simulados os dois canais independentemente e a diferença entre eles, e apenas os dois canais com menos informações são efetivamente codificados. Assim, há uma coerência entre os resultados apresentados nas Figuras 6.3 e 6.7, na medida em que se demanda mais tempo na codificação conjunta de canais.

6.2 Testes do MILC

Os testes de codificação no MILC seguem, basicamente, as mesmas condições dos testes realizados no MPEG-4 ALS. A exceção é o parâmetro *block switching*, que não foi implementado no MILC.

Ressalta-se que as taxas de codificação obtidas nos testes do MILC são equivalentes às taxas obtidas no codificador de referência (para ordem de predição fixa), por isso as informações pertinentes a esse indicador não serão ilustradas nesta seção.

Os tempos de codificação do MILC em geral, mostraram-se bem superiores aos tempos de codificação do padrão MPEG-4 ALS. Isso pode ser comprovado visualizando-se as Figuras 6.1, 6.3, 6.7, 6.9, 6.11 e 6.13. A justificativa para esse fato se deve às linguagens empregadas na implementação do MILC, que são MATLAB e C (Seção 5.1). O MPEG-4 ALS é integralmente implementado em C++, por isso é mais rápido que o MILC.

A seguir há seções correspondentes aos testes realizados no MILC.

6.2.1 Teste Considerando-se o Tamanho do Quadro

Este teste consiste na obtenção do tempo de codificação dos arquivos de áudio utilizados, levando-se em consideração o parâmetro tamanho de quadro. A Figura 6.9 ilustra os valores de tempo de codificação obtidos nesse teste. Como já observado, nela, nota-se que os tempos de codificação obtidos no MILC são sempre superiores aos encontrados no MPEG-4 ALS.

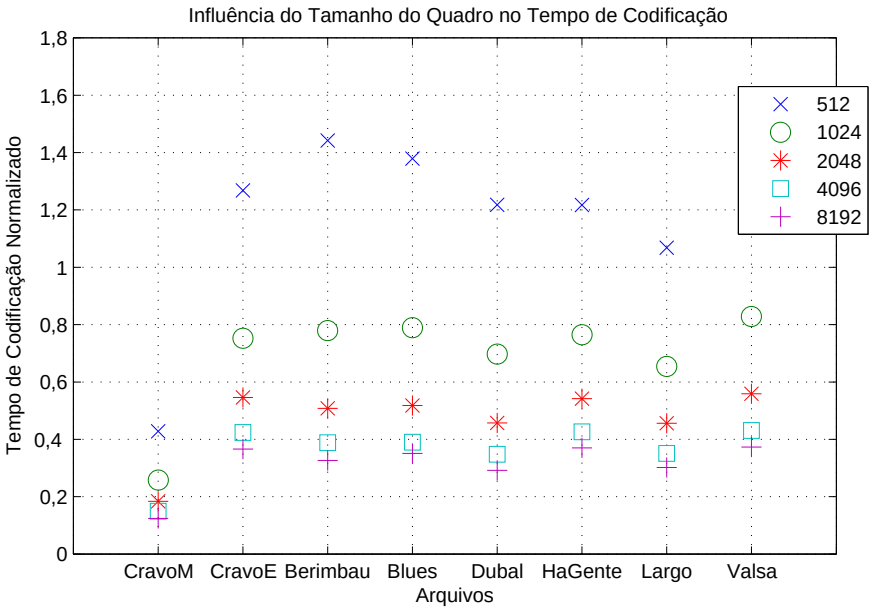

Figura 6.9: Tempo de codificação para variações no tamanho do quadro no MILC.

6.2.2 Teste da Ordem de Predição

Esta seção diz respeito ao teste do parâmetro ordem de predição. Nesse teste, obtêm-se o tempo de codificação a partir de alguns valores de ordem de predição fixa e um valor de ordem de predição adaptativa (veja as Figuras 6.11 e 6.10) e a taxa de codificação apenas para ordem de predição adaptativa.

Em se tratando da taxa de codificaçao, a Figura 6.10 mostra as taxas de codificação obtidas a partir da ordem de predição adaptativa. Comparando-a com a parte correspondente da Figura 6.4, constata-se que há pequenas diferenças que são justificadas pelas diferentes implementações do recurso ordem de predição adaptativa no MILC em relação ao MPEG-4 ALS (veja as Seções 4.1.3.5 e 5.2).

Levando-se em consideração o tempo de codificação, a Figura 6.11 ilustra os tempos de codificação obtidos através das respectivas ordens de predição utilizadas. Para ordem de predição fixa, os pontos dessa figura são próximos dos pontos correspondentes da Figura 6.3. Já para ordem de predição adaptativa, nota-se que os

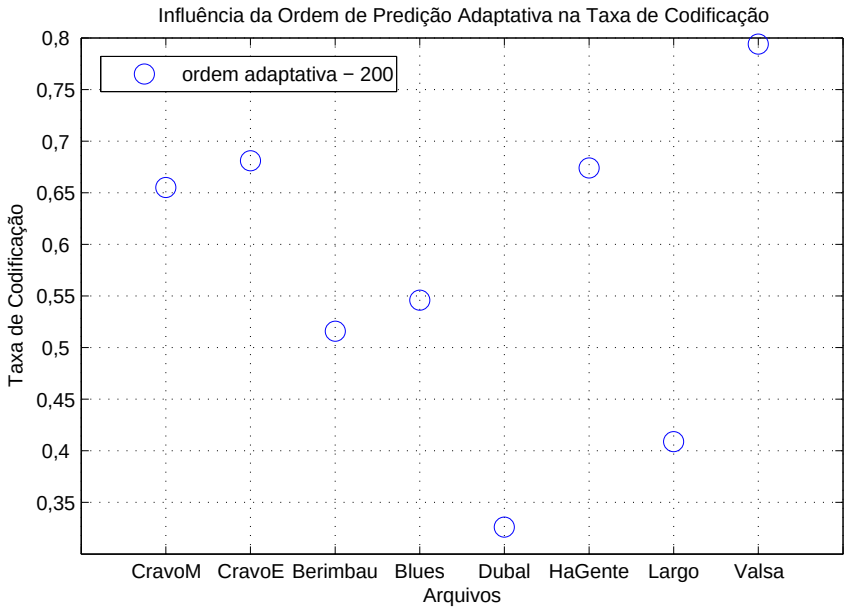

Figura 6.10: Taxa de codificação para variações na ordem de predição no MILC.

tempos de codificação no MILC são muito maiores que no MPEG-4 ALS. Isso se deve tanto às linguagens empregadas em sua implementação, quanto aos diferentes algoritmos usados na implementação desse recurso.

6.2.3 Teste da Codificação Independente

Esta seção corresponde ao modo de codificação independente de canais. Obtêm-se, com o teste aqui realizado, a taxa para ordem e predição adaptativa (Figura 6.13) e o tempo (Figura 6.13) de codificação para ordens de predição fixa e adaptativa.

Quanto à taxa de codificação, nota-se uma proximidade muito grande entre o MILC e o MPEG-4 ALS, embora este último apresente os melhores resultados.

Já quanto ao tempo de codificação, como já observado nas Seções 4.1.3.5 e 5.2, as grandes diferenças observadas se devem às respectivas implementações de ordem de predição adaptativa.

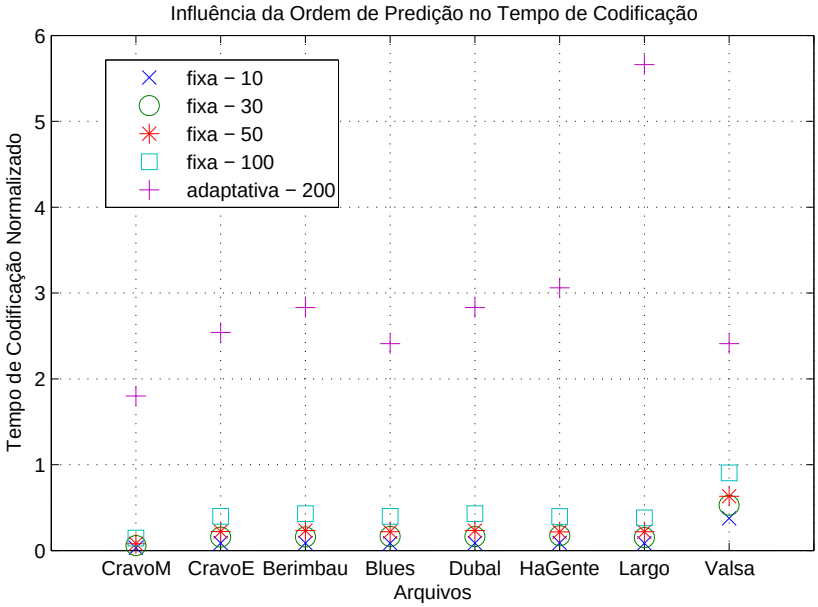

Figura 6.11: Tempo de codificação para variações na ordem de predição no MILC.

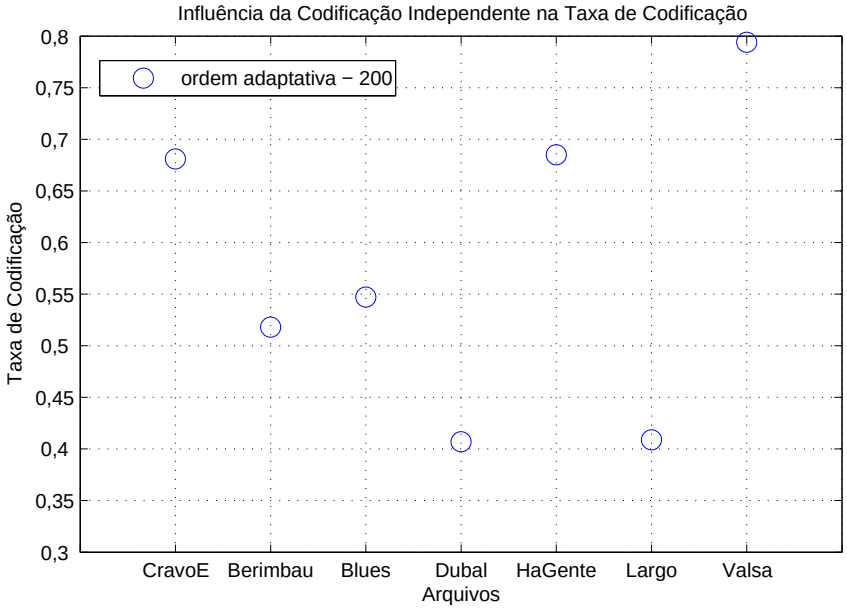

Figura 6.12: Taxa de codificação considerando-se a codificação independente no MILC.

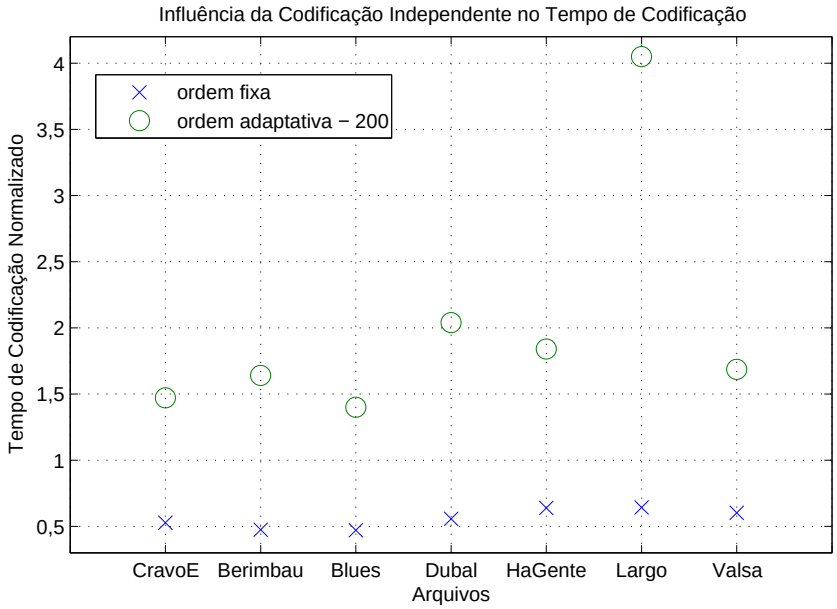

Figura 6.13: Tempo de codificação considerando-se a codificação independente no MILC.

Capítulo 7

Conclusão

Conclui-se que com este trabalho foi possível conhecer assuntos tais como codificação e compressão de arquivos em geral, especialmente arquivos de áudio, na medida em que se implementou um CODEC de áudio sem perdas (MILC). Em linhas mais específicas, essa implementação possibilitou a aplicação prática de alguns conhecimentos ligados à área de processamento de sinais tais como quantização, predição linear e codificação por entropia.

O MILC, através de suas ferramentas e diante da pluralidade dos sinais de áudio testados, proporcionou bons resultados de codificação em comparação com o codificador de referência do MPEG-4 ALS, uma vez que os valores de taxas de codificação ficaram iguais (ordem de predição fixa) ou bem próximos (ordem de predição adaptativa) aos valores do codificador de referência. No entanto, em se tratando dos tempos necessários para as codificações, o MILC levou muito tempo em codificações com ordem de predição adaptativa e um tempo próximo ao do codificador de referência em codificações com ordem de predição fixa, o que indica uma dificuldade do MILC para trabalhar com ordem de predição adaptativa.

Quanto às dificuldades na implementação do MILC, considera-se justamente a implementação da ordem de predição adaptativa como a parte mais difícil, pois não há informações em baixo nível no *software* do codificador de referência. A questão da integração das linguagens MATLAB e C para melhorar o desempenho do MILC configura outra dificuldade encontrada, pois foram necessários conhecimentos avançados da ferramenta *mex files* contida no MATLAB.

Por fim, dentre as propostas para melhorar o desempenho e a manipulação do MILC, enumeram-se: mudar a linguagem usada na sua implementação para agilizar o processamento; melhorar os algoritmos que trabalham com ordem de predição adaptativa; e criar uma interface para o usuário.

Referências Bibliográficas

[1] BOSI, M., GOLDBERG, R. E., *Introduction to Digital Audio Coding and Standards*, chapter Representation of Audio Signals, Boston/Dordrecht/London, Kluwer Academic Publishers, pp. 63–67, 2003.

[2] COALSON, J., "FLAC - Free Lossless Audio Codec", http://flac.sourceforge.net., 2008, (Acesso em julho 2008).

[3] BRYANT, D., "WavPack", http://www.wavpack.com/, 2008, (Acesso em julho 2008).

[4] BECKER, T., "MPEG-4 ALS Reference Software", http://thbeck.de/Tak/Tak.html, 2008, (Acesso em julho 2008).

[5] ASHLAND, M. T., "Monkey's", http://www.monkeysaudio.com, 2008, (Acesso em julho 2008).

[6] ROBINSON, T., "Shorten", http://www.etree.org/shnutils/shorten/, 2008, (Acesso em julho 2008).

[7] BEVIN, M., "LA", http://www.lossless-audio.com/, 2008, (Acesso em julho 2008).

[8] DJOURIK, A., "TTA", http://www.true-audio.com, 2008, (Acesso em julho 2008).

[9] LIEBCHEN, T., "LPAC", http://www.nue.tu-berlin.de/wer/liebchen/lpac.html, 2008, (Acesso em julho 2008).

[10] ISO/IEC-14496-3:2005/AMD.2:2006(E), *Information technology, Coding of audio and visual objects, part 3: Audio*. International Standard, 2006.

[11] TUB, "MPEG-4 ALS", http://www.nue.tu-berlin.de/forschung/projekte/lossless/mp4als.html, 2008, (Acesso em julho 2008).

[12] HARADA, N., KAMAMOTO, Y., LIEBCHEN, T., *et al.*, "The MPEG-4 Audio Lossless Coding (ALS) Standard-Tecnology and Applications", *119 th AES Convention*, v. 3, 2005.

[13] LIEBCHEN, T., REZNIK, Y., "Improved Forward-Adaptive Prediction for MPEG-4 Audio Lossless Coding", *118 th AES Convention*, v. 3, pp. 142–151, 2005.

[14] HIDROGENAUDIO, "Lossless Comparison", http://wiki.hydrogenaudio.org/index.
php?title=Lossless_comparison, 2008, (Acesso em julho 2008).

[15] HANS, M., SCHAFER, R. W., "Lossless Compression of Digital Audio", *IEEE Sinal Processing Magazine*, v. 18, pp. 21–32, 2001.

[16] DBPOWERAMP, "dBpoweramp", http://superdownloads.uol.com.br/
download/112/dbpoweramp-music-converter/, 2008, (Acesso em julho 2008).

[17] TUB, "MPEG-4 ALS Reference Software", ftp://ftlabsrv.nue.tu-berlin.de/mp4lossless/, 2008, (Acesso em julho 2008).

[18] MAKHOUL, J., "Linear prediction: a tutorial review", *Proceedings of the IEEE*, v. 63:4, 561-80, 1975.

[19] BOSI, M., GOLDBERG, R. E., *Introduction to Digital Audio Coding and Standards*, chapter Quantization, Boston/Dordrecht/London, Kluwer Academic Publishers, pp. 38–43, 2003.

[20] HAYKIN, S., *Sistemas de Comunicação Analógicas e Digitais*, chapter Limites Fundamentais da Teoria da Informação, Boston/Dordrecht/London, Bookman, pp. 606–608, 2004.

[21] HARRIS, J. F., "On the Use of Windows for Harmonic Analisys with the Discrete Fourier Transform", *Proceedings of the IEEE*, v. 66:1, 1978.

[22] DINIZ P. E SILVA, E. E NETTO, S., *Processamento Digital de Sinais*. Bookman, 2004.

[23] MATHWORKS, "Mex Files", http://www.mathworks.com/support/technotes/1600/1605.html, 2009, (Acesso em outubro 2009).

[24] VISWANATHAN, R., MAKHOUL, J., "Quantization properties of transmission parameters in linear predictive systems", *IEEE Trans. on Acoustics, Speech and Signal Processing*, v. 23:3, 309-21, 1975.

Apêndice A

Informações Específicas

Algoritmo 6 Algoritmo de autocorrelação[10][11].

rxx = Autocorrelacao(ordem, frame_janelado, tam_frame) {Função que gera um vetor de autocorrelação de um segmento de um determinado sinal.
Parâmetros de entrada: ordem - ordem de predição; frame_janelado: segmento do sinal janelado; tam_frame: tamaho do segmendo do sinal.
Parâmetros internos: indice e cont - contadores;
Parâmetro de saída: rxx - vetor de autocorrelação do segmento do sinal.}

```
para (indice = 0; indice ≤ ordem_pred; indice++) faça
  rxx[indice] ⇐ 0.0
  para (cont = indice; cont ⇐ tam_frame; cont++) faça
    rxx[indice] ⇐ rxx[indice] + frame_janelado[cont] * frame_janelado[cont-indice]
  fim para
fim para
```

Algoritmo 7 Algoritmo de *Levinson Durbin* usado com ordem adaptativa [11].

DurbinAdapt(rxx, ordem_pred, parCof, evar) {Função que gera os coeficientes de reflexão, quando a ordem de predição é adaptativa.

Parâmetros de entrada: rxx - vetor de autocorrelação de um quadro do sinal; ordem_pred - ordem de predição.

Parâmetros internos: i, j - contadores; temp e dir - variáveis auxiliares.

Parâmetro de entrada e saída (passagem por referência): parCof - coeficientes de reflexão; evar - erro de predição para a n-ésima ordem do filtro.}

```
se indice == 1 então
  evar ⇐ rxx[0]
fim se
parCof[ord] ⇐ -rxx[ord]
para (j = 1; j ⇐ ord; j++) faça
  parCof[ord] ⇐ parCof[ord] - dir[ord] * rxx[ord-j]
fim para
parCof[ord] ⇐ parCof[ord] / evar
dir[ord] ⇐ parCof[ord]
para (j = 1; j ≤ i/2; j++) faça
  temp ⇐ dir[j] + parCof[ord] * dir[ord-j]
  dir[ord-j] ⇐ dir[ord-j] + parCof[ord] * dir[j]
  dir[j] ⇐ temp
  evar ⇐ evar * (1 - parCof[ord] * parCof[ord])
fim para
```

parâmetro	**valor ou *status***
frequencia de amostragem	44,1 kHz
resolução	16 bits/amostra
representação	inteira
GBMC	desabilitado
RLSLMS	desabilitado
LPC	desabilitado
demais parâmetros	desabilitados

Tabela A.1: Parâmetros gerais usados nos testes do MPEG-4 ALS.