UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E COMPUTAÇÃO
DEPARTAMENTO DE COMUNICAÇÕES

Dissertação de Mestrado

ALGORITMOS DE BUSCA EM CODIFICADORES ACELP

Lucas de Melo Jorge Barbosa

Banca Examinadora:

- Luís Geraldo Pedroso Meloni - UNICAMP (Orientador)
- Dalton Soares Arantes - UNICAMP
- Miguel Arjona Ramírez - USP

Campinas, novembro de 2002

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

B234a

Barbosa, Lucas de Melo Jorge
Algoritmos de busca em codificadores ACELP / Lucas de Melo Jorge Barbosa.--Campinas, SP: [s.n.], 2002.

Orientador: Luís Geraldo Pedroso Meloni.
Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Codificador de voz. 2. Sistemas de processamento da fala. 3. Microprocessadores. I. Meloni, Luís Geraldo Pedroso. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

AGRADECIMENTOS

Gostaria de agradecer inicialmente ao CNPq e à Ericsson Telecomunicações S.A. pelo suporte dado à este trabalho.

Agradeço ainda ao meu orientador, Prof. Luís G. P. Meloni, por toda a ajuda e a confiança depositada em meu trabalho, assim como ao Prof. Miguel A. Ramírez, pela colaboração dada com a implementação do algoritmo JPAS.

Agradeço também aos colegas do Laboratório de Processamento de Sinais de Multimídia em Tempo Real pelo apoio e colaboração.

Por fim, gostaria de agradecer principalmente à minha família, à qual dedico esse trabalho, por todo o amor, apoio e incentivo.

RESUMO

Esse trabalho apresenta uma revisão do processo de codificação de voz baseado em predição linear com excitação por códigos (CELP), assim como um estudo sistemático e aprofundado dos algoritmos de busca utilizados em dicionários de multi-pulsos algébricos, que caracterizam os codificadores ACELP. Além do estudo de diversos tipos de algoritmos de busca existentes, propõe-se aqui um novo método, denominado de busca seqüencial de posições.

Os algoritmos de busca são descritos com uma notação homogênea e representados através de pseudo-códigos. Além disso, são feitas análises dos algoritmos no que diz respeito à complexidade e à qualidade de voz através da inserção dos mesmos nos codificadores G.729 e GSM-AMR, que pertencem respectivamente aos organismos de padronização ITU-T e ETSI. As medidas levantadas mostram que, quando comparado com os algoritmos de busca adotados pelas padronizações atuais de codificação de voz, a busca seqüencial de posições requer um esforço computacional consideravelmente menor, ao custo apenas de uma pequena degradação na qualidade perceptual da voz codificada.

Ainda no âmbito da redução do tempo gasto no processo de busca nos dicionários algébricos, este trabalho também apresenta uma análise do impacto da implementação otimizada dos algoritmos de busca no desempenho do codificador como um todo, questão que surge com a utilização de processadores digitais de sinais modernos com alto nível de paralelismo.

ABSTRACT

This work presents a review of the code-excited linear predictive (CELP) voice coding scheme, and a detailed and structured study of the search algorithms for algebraic multipulse codebooks (ACELP). In addition to the study of the current algorithms, this work also proposes a new search method, named as position-sequential search.

Throughout this work the search algorithms are described using a common notation and represented by pseudo-codes. The computational complexity and perceptual quality of the search algorithms have been measured by the use of the ITU-T G.729 and the ETSI GSM-AMR speech coding standards. The implementations showed that in comparison to standard search procedures the position-sequential search causes only a slight degradation in voice quality, whereas providing a significant reduction in computational complexity.

In the sense of reducing the time spent in the search process, this work also analyzes the impact of the optimized implementation of search algorithms on the global performance of the coding process. This question arises in algorithm implementations in modern digital signal processors with a high level of parallelism.

Conteúdo

Lista de Tabelas

Lista de Figuras

Capítulo 1

INTRODUÇÃO

As pesquisas na área de codificação de voz são voltadas para a obtenção de representações digitais compactas para os sinais de fala. O objetivo principal é comprimir o sinal o máximo possível, ou seja, representá-lo com o menor número possível de bits, procurando por outro lado manter a qualidade perceptual da voz. Com isso, a codificação ou compressão de voz procura tornar mais eficientes tanto a transmissão como o armazenamento de sinais de voz.

Pode-se dizer que as principais características de um codificador de voz são:

- A taxa de compressão, medida geralmente em kbps (kilo-bits por segundo), e que expressa o quanto o codificador pode comprimir o sinal de voz;
- A qualidade de voz, uma medida subjetiva extraída da comparação do sinal de voz original com o sinal decodificado;
- A complexidade computacional do processo de codificação, ou seja, o esforço computacional necessário para a execução de todas as tarefas envolvidas no processo de codificação;
- O atraso inserido nos processos de codificação e decodificação. Um atraso muito grande pode atrapalhar a conversação, tornando-a desconfortável e pouco natural. Nos casos onde a compressão do sinal de voz é feita apenas com o objetivo de um armazenamento mais eficiente do sinal o atraso não é importante.

Vale ressaltar que a taxa de compressão e a qualidade da voz são dois fatores que estão bastante atrelados e que geralmente se opõem, ou seja, quanto maior a compressão,

maior a perda na qualidade de voz. Por outro lado, boas relações de compressão e qualidade são geralmente obtidas com o preço de um alto esforço computacional, o que por vezes impossibilita a implementação prática de tais esquemas de codificação.

1.1 Motivação

Atualmente, os métodos mais eficientes de codificação do sinal de fala utilizam uma modelagem matemática do sistema humano de produção da voz. No modelo assume-se que a voz é gerada através da passagem de um sinal de excitação por um filtro só-polos variante no tempo, que tem seus parâmetros extraídos periodicamente através da técnica de predição linear. Entre os métodos de codificação desse tipo o mais conhecido é a chamada codificação baseada em predição linear com excitação por códigos, ou simplesmente codificação CELP (*Code-Excited Linear Prediction*), onde a excitação é representada por uma das entradas de um dicionário de códigos.

O sucesso da codificação CELP deve-se ao fato de que ela possibilita a transmissão de voz com boa qualidade a taxas de cerca de 8kbps. Entretanto, apesar de apresentarem um bom desempenho, os codificadores CELP também apresentam uma alta complexidade computacional, sendo grande parte dessa complexidade devida à tarefa da busca no dicionário, ou seja, a determinação de qual entrada do dicionário deve ser escolhida para representar o sinal de excitação.

Para reduzir o tempo gasto no processo de busca os chamados codificadores ACELP (*Algebraic-CELP*), ou simplesmente codificadores algébricos, utilizam dicionários esparsos que simplificam os cálculos necessários para a determinação do melhor sinal de excitação. Porém, a complexidade inerente ao processo de busca é tamanha que mesmo em codificadores algébricos a adoção de um procedimento ótimo de busca no dicionário seria impraticável considerando-se os processadores digitais de sinais atuais. Tal fato levou ao desenvolvimento de algoritmos de busca sub-ótimos, que executam o processo de busca de maneira simplificada, introduzindo por outro lado uma certa degradação na qualidade de voz resultante.

Pode-se dizer que é graças à redução de complexidade decorrente da adoção de algoritmos sub-ótimos que o método de codificação ACELP pode ser utilizado na prática e implementado em tempo-real. O desenvolvimento de algoritmos de busca sub-ótimos ainda é importante porque, além de acelerar o próprio processo de codificação, a execução mais rápida da busca também permite que sejam adotados dicionários mais complexos.

Nesse sentido, esta tese apresenta uma análise dos diferentes tipos de algoritmos de busca existentes, e propõe um procedimento de busca de baixa complexidade denominado de busca seqüencial de posições, sugerido como uma alternativa para a redução da complexidade da busca em dicionários algébricos.

No âmbito da redução do tempo gasto no processo de busca também é analisado o impacto da implementação otimizada dos algoritmos de busca. Uma implementação otimizada de um algoritmo para um certo processador é uma implementação eficiente e construída especificamente para o processador considerado, onde procura-se tirar a maior vantagem possível dos recursos de hardware disponíveis. Como a busca no dicionário é uma das tarefas mais custosas da codificação CELP, sua implementação otimizada traz grandes ganhos no que diz respeito ao desempenho do codificador como um todo.

1.2 Organização

Inicialmente, no Capítulo 2, é apresentado o sistema humano de produção da fala e suas relações com as características do sinal de voz. A compreensão dos mecanismos de produção da voz é vital para o entendimento das técnicas de compressão da voz utilizadas na maioria dos codificadores atuais.

No Capítulo 3 é apresentada a predição linear, uma das técnicas mais poderosas de extração de características e modelagem do sinal de voz. Inicialmente é apresentada de maneira generalizada a predição linear de seqüências temporais, e em seguida procura-se destacar em particular a aplicação da predição linear na codificação da voz.

O método de codificação CELP é descrito no Capítulo 4, sendo dada maior ênfase ao processo de busca. Inicialmente são apresentados os dicionários de códigos, e o método denominado análise-por-síntese, utilizado para a escolha do melhor sinal de excitação. Por fim é apresentado o processo básico de busca no dicionário, ponto de partida para o desenvolvimento de algoritmos de busca.

No Capítulo 5 são apresentados os dicionários de multi-pulsos algébricos, que caracterizam os codificadores ACELP, e são descritos alguns tipos de algoritmos de busca utilizados em padrões de codificação ACELP atuais. Por último são apresentados dois algoritmos de busca seqüenciais de baixa complexidade, a busca conjunta de posições e amplitudes, e a busca seqüencial de posições, proposta nesse trabalho.

O Capítulo 6 apresenta alguns resultados de medidas de complexidade computacio-

nal e qualidade de voz referentes aos diversos tipos de algoritmos de busca. Nas análises procura-se dar maior ênfase ao desempenho dos algoritmos de busca seqüenciais, destacando principalmente a redução de complexidade decorrente da adoção desse tipo de algoritmo.

No Capítulo 7 é apresentada rapidamente a implementação otimizada de algoritmos, sendo descritas a otimização gerada por compiladores e a otimização manual. Por fim é considerada uma implementação otimizada do codificador G.729A e são analisados os ganhos no que diz respeito à redução de complexidade decorrente da otimização do algoritmo de busca.

As conclusões deste trabalho estão expostas no Capítulo 8, e em seguida ainda constam dois apêndices. O Apêndice A apresenta, dando ênfase aos algoritmos de busca adotados, os padrões de codificação G.729 e GSM-AMR, utilizados para a realização de medidas de qualidade de voz dos algoritmos de busca. O Apêndice B apresenta o processador digital de sinais de ponto-fixo TMS320C6202 da Texas Instruments^{TM}, utilizado para a descrição do processo de otimização de códigos e para a extração de medidas de complexidade dos algoritmos de busca.

Capítulo 2

PRODUÇÃO DA VOZ

Este capítulo procura apresentar de maneira rápida o sistema de produção da voz, aproveitando para destacar também as suas relações com as características temporais e espectrais do sinal de fala.

O estudo do sistema de produção da voz é importante porque está diretamente relacionado com o desenvolvimento de métodos eficientes de codificação. Os codificadores de voz atuais, por exemplo, utilizam técnicas de processamento de sinais e modelos matemáticos baseados no funcionamento do sistema humano de produção da voz.

2.1 O Sistema Humano de Produção da Voz

O que se entende por som consiste, em última instância, na sensação proporcionada pela vibração de delicadas membranas do ouvido, causada por uma onda de pressão que se propaga geralmente no ar. É importante notar que tal onda se origina em geral da vibração de algum material. De fato, pode-se observar que todos os instrumentos musicais se baseiam nesse princípio para gerar sons. Os instrumentos de cordas geram sons a partir da vibração de cordas esticadas, enquanto que a maioria do instrumentos de percussão se baseiam na vibração de superfícies. A voz humana, por sua vez, é gerada através da vibração das cordas vocais.

O princípio simples de geração da voz não tira dela o papel de maior responsável pelo suprimento da nossa necessidade de comunicação. Apesar dos enormes avanços científicos alcançados nas últimas décadas dependemos exclusivamente dos nossos sentidos mais naturais para nos comunicarmos, e a voz é sem dúvida um dos mais importantes. De fato,

ao fazer um ligação telefônica intercontinental, por exemplo, os dispositivos transmissores e receptadores não serão, em última instância, os aparelhos telefônicos utilizados, mas sim os sistemas auditivos e os sistemas de produção da voz dos indivíduos em questão.

Pode-se considerar que o processo de produção da voz nos seres humanos tem início na mente. Inicialmente, respondendo a algum estímulo externo, ou de acordo com a própria vontade, surge de alguma maneira a vontade de expressar algo, uma idéia, um sentimento, uma reclamação, um pedido, o que quer que seja. Para externar o que se quer pode-se fazer uso de gestos ou de sons. Aqui não é de interesse o primeiro caso, onde é enviada um informação visual, mas o segundo, onde a informação é sonora. Nesse caso a mente trata então de transformar a idéia abstrata em uma frase, montando uma estrutura léxica e sintática para o que se pretende expressar. O próximo passo é fazer uso do sistema mecânico de produção da voz para gerar som, que será então transmitido através do meio para os outros indivíduos.

O processo que ocorre na mente durante a fala ainda não é completamente compreendido pela ciência. De fato, pouco se sabe sobre o funcionamento da mente humana, tendo-se apenas conhecimentos superficiais sobre seu comportamento. Portanto, será exposto a seguir apenas o processo mecânico de produção da voz, que pode ser considerado como composto pelas três etapas seguintes:

- A criação pelos pulmões de um fluxo de ar;
- A passagem desse fluxo de ar pela laringe, onde pode ou não haver a influência das cordas vocais;
- A atuação do trato vocálico.

Baseado nessa divisão e na estrutura física do corpo humano, o sistema mecânico de produção da voz é em geral dividido em três partes: o sistema abaixo da laringe, a laringe e as estruturas que a cercam, e o sistema acima da laringe. A Figura 2.1 mostra o caminho percorrido pelo ar durante a fala, ressaltando a disposição física dos elementos acima e abaixo da laringe.

2.1.1 O Sistema Abaixo da Laringe

Também chamado de sub-laríngeo, esse sistema é o responsável pelo bombeamento de ar para a geração da voz, sendo o "motor" do sistema de produção da fala. O sistema

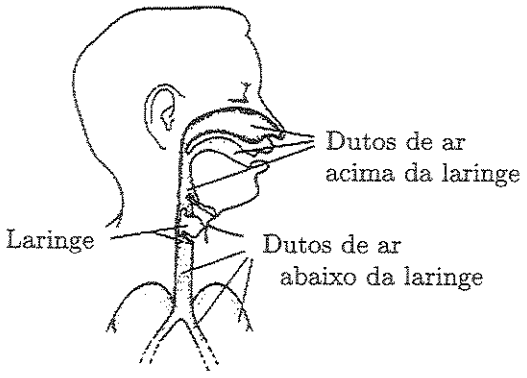


Figura 2.1: Divisão do sistema de produção da voz entre os elementos acima, abaixo e que circundam a laringe.

sub-laríngeo é composto basicamente pelos pulmões e pela traquéia, podendo-se ainda incluir nesse grupo a estrutura muscular que auxilia os pulmões.

Os pulmões, além de fazerem parte do sistema respiratório, atuando na oxigenação do sangue, têm grande importância na produção da fala pelo fato de serem capazes de prover um fluxo de ar aproximadamente constante por um grande período de tempo. Assim, apesar de durante a respiração normal os períodos de inspiração e expiração serem aproximadamente iguais, durante a fala é gasto em média apenas cerca de 10% do tempo na inspiração, sendo o ar liberado lentamente dos pulmões devido a existência de obstáculos no seu fluxo para o meio externo.

Para criar o fluxo de ar necessário para a produção da voz os pulmões sofrem a ação de forças musculares que expandem ou reduzem seu volume, o que causa respectivamente uma redução ou um aumento da pressão interna, forçando o ar então a fluir do meio externo (maior pressão) para o meio interno (menor pressão), ou vice-versa.

2.1.2 A Laringe e as Estruturas que a Cercam

Como exposto na Figura 2.2, o caminho do ar que deixa os pulmões passa pela traquéia, atingindo logo em seguida a laringe, onde o fluxo de ar sofrerá a atuação das cordas vocais. A Figura 2.2 mostra em detalhes os elementos principais do sistema de produção da voz.

Na produção da voz a laringe assume basicamente o papel de mecanismo fonatório, transformando o fluxo de ar contínuo proveniente dos pulmões em um onda de pulsos de

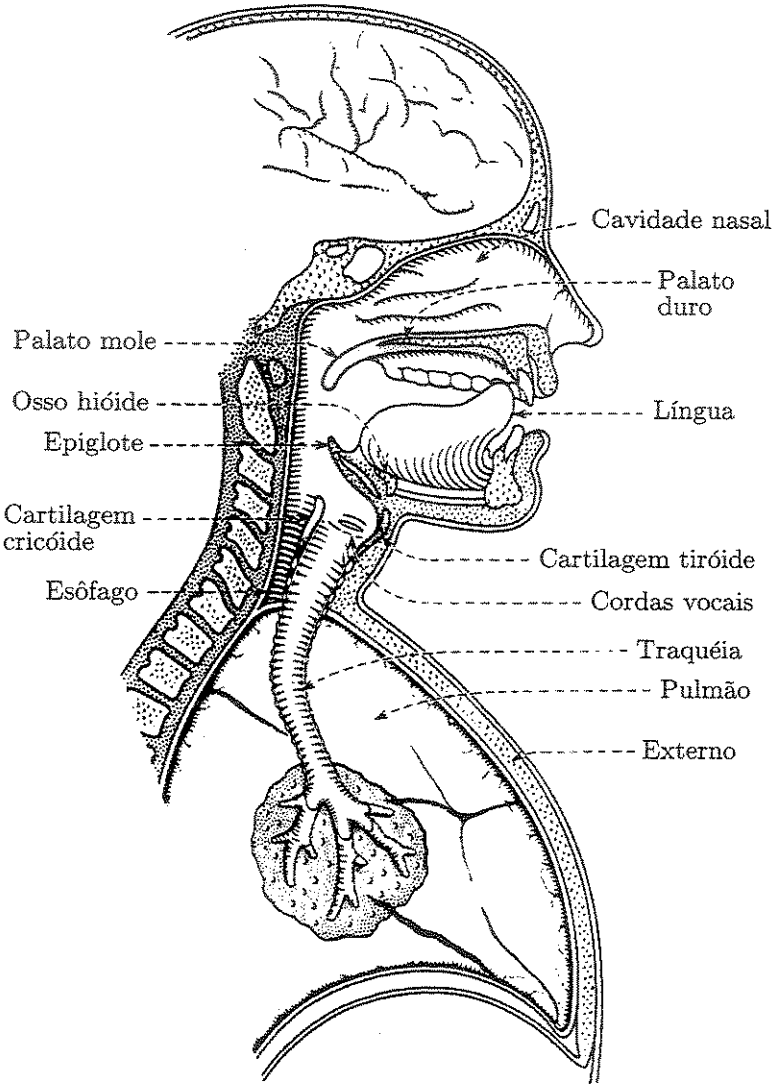


Figura 2.2: Detalhamento das principais estruturas do sistema de produção da voz.

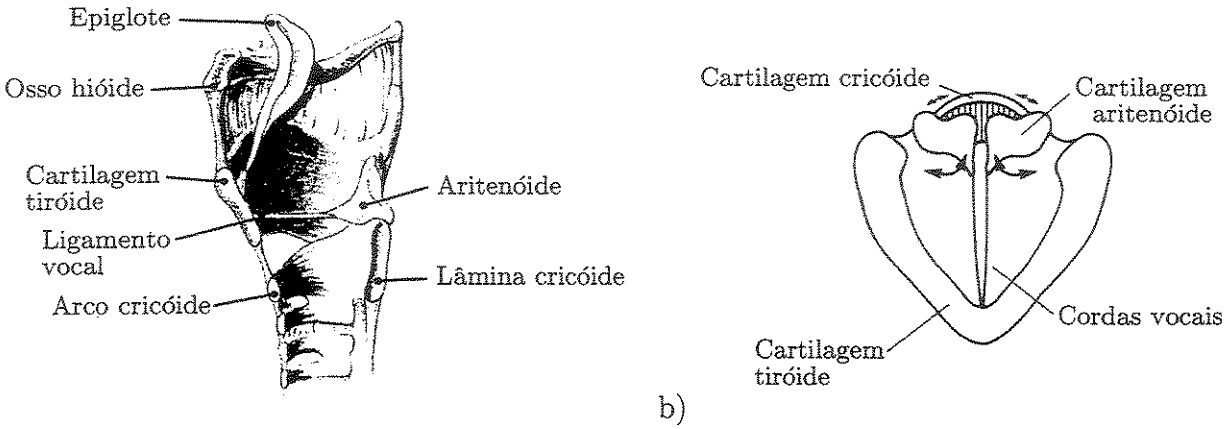


Figura 2.3: Posicionamento das principais estruturas que compõem a laringe (De Dickson e Maue-Dickson, 1982).

ar. Modificando a posição e a tensão das cordas vocais a laringe ainda pode controlar a freqüência de vibração das mesmas, alterando profundamente as características do sinal de voz gerado.

Durante a produção da voz o fluxo de ar gerado pelos pulmões pode ou não passar sem alterações pela laringe. Esse fato faz com que os fonemas, menores unidades com significado na fonologia de linguagem, sejam geralmente classificados quanto ao seu modo de produção como:

1. Surdos - As cordas vocais ficam bem separadas e o fluxo de ar proveniente dos pulmões passa pela glote[1] sem maiores interferências. Nesse caso o fluxo de ar será modelado apenas no trato vocal, onde uma constrição formada pela língua, dentes ou lábios produzirá uma turbulência no fluxo de ar.

2. Sonoros - As cordas vocais são posicionadas próximas uma da outra, de forma a vibrarem com a passagem do ar. Nesse processo a glote abre e fecha repetidamente, transformando o fluxo contínuo de ar em um onda de pulsos de ar.

O período de vibração das cordas vocais define a chamada *freqüência fundamental* das cordas vocais. Tal freqüência varia de acordo com o indivíduo, sendo ainda sujeita a tensão aplicada na cordas vocais e a pressão exercida pelos pulmões. Em geral, a freqüência

[1] O termo abertura glótica, fenda glótica, ou simplesmente glote, se refere ao espaço compreendido entre as cordas vocais.

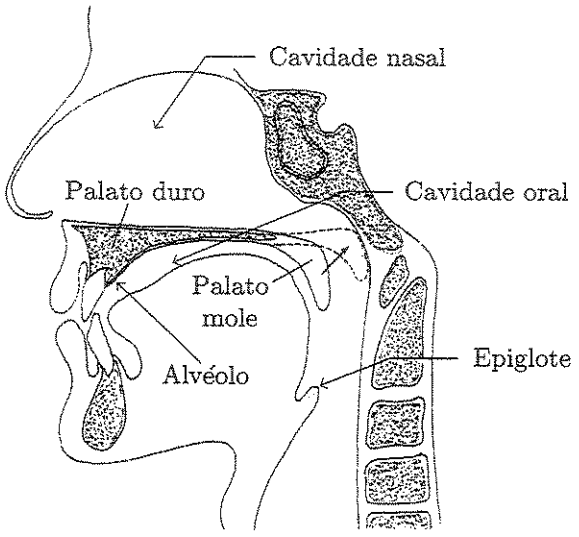


Figura 2.4: Corte esquemático do aparelho fonatório humano. (Adaptado de Dickson e Maue-Dickson, 1982).

fundamental é maior nas mulheres (média de 250Hz) do que nos homens (média de 125Hz), sendo esta a principal razão da diferença entre a voz masculina e a feminina.

2.1.3 O Sistema Acima da Laringe

Também chamado de supra-laríngeo, o sistema acima da laringe é composto basicamente pela faringe e pelo trato vocal. O papel desse sistema na produção da voz é a modelagem, a filtragem, do som gerado pela ação da laringe e do sistema sub-laríngeo. É importante observar que vários elementos da faringe e do trato vocal sofrem controle voluntário, permitindo que sejam feitas grandes modificações aos sons produzidos pelas cordas vocais, ou mesmo que sejam agregadas partes distintivas ao mesmos, podendo haver inclusive a produção de sons próprios do sistema supra-laríngeo.

A faringe conecta a laringe ao trato vocal e às fossas nasais. Na produção da voz ela atua como um tubo acústico, cuja forma possui influência fundamental na voz gerada. Vale ressaltar ainda que vários músculos constritores atuam sobre a faringe possibilitando uma modificação considerável de sua forma.

O trato vocal atua na produção da voz, juntamente com a faringe, compondo uma cavidade acústica de ressonância, onde o fluxo de ar é transformado em sons inteligíveis através da articulação conjunta de seus elementos. A Figura 2.4 mostra os elementos mais

importantes do trato vocal do ponto de vista da produção da voz.

A riqueza da linguagem humana está sem dúvida associada a diversidade dos sons que podem ser gerados pelo sistema de produção da voz. Tal diversidade, por sua vez, deve-se principalmente aos diferentes movimentos e formas do trato vocal durante a fala. São variadas as maneiras das quais o trato vocal pode atuar durante a produção de um determinado fonema: a cavidade nasal pode ou não estar acoplada com a cavidade oral; pode ou não haver uma constrição no trato vocal; pode haver constrições momentâneas, para que a liberação do ar preso gere os chamados sons oclusivos. Para lidar com a grande variedade de fonemas, a fonética articulatória, além de classificá-los como sonoros ou surdos, ainda os classifica quanto a zona e ao modo de articulação.

Quanto a zona de articulação os fonemas são classificados basicamente como labiais, dentais (ex. **t**odo), alveolares (ex. **n**ata), palatais (ex. ho**j**e), velares (ex. ca**rr**o), faríngeos ou glóticos. Vale ressaltar que os dois últimos tipos não existem em português, e que os fonemas labiais se dividem ainda entre os bilabiais (ex. **p**ato) e os labio-dentais (ex. **f**aca). Já quanto ao modo de articulação, os fonemas podem ser classificados basicamente como:

- Vogais - O ar flui diretamente através da faringe e do trato vocal sem encontrar qualquer constrição que cause turbulência. Esse tipo de som é sempre sonoro, pois sempre há a vibração das cordas vocais. Os elementos acima da laringe atuam apenas como ressonadores, modificando o fluxo pulsado de ar através da inserção de algumas freqüências de ressonância, que dependem basicamente do posicionamento dos elementos do trato vocal e da faringe. Ex. Casa, mesa;

- Líquidos (ou laterais) - São similares as vogais, mas nesse caso a língua é utilizada para gerar uma leve obstrução no trato vocal. Ex. Ca**lh**a, **l**ata;

- Nasais - Também são semelhante às vogais, mas nesse caso o véu palatino está abaixado, permitindo a passagem de ar para a cavidade nasal. Ex. Que**n**te, ta**n**to;

- Oclusivos (ou plosivos) - Envolvem o fechamento completo e subseqüente liberação do trato vocal. O som é gerado através de uma "explosão"do ar que estava previamente recluso na região supra-laríngea. São sons extremamente rápidos e transitórios, onde pode ou não haver a vibração prévia das cordas vocais. Ex. **B**ola, **t**odo, **d**ata;

- Fricativos - Uma constrição do trato vocal, na faringe ou na glote, produz uma turbulência no fluxo de ar. Pode ou não haver a vibração das cordas vocais. Caso não

haja, o sinal resultante se aparenta a um ruído. Ex. Acho, vaca, sapo;

É importante ressaltar que as freqüências de ressonância envolvidas no processo de produção da voz, que correspondem as ressonâncias do trato vocal com o possível acoplamento da cavidade nasal, são chamadas de freqüências de formantes ou simplesmente *formantes*. Usualmente a freqüência fundamental é representada por $F0$, enquanto que os formantes são denominados por $F1, F2, F3, ...$, por ordem crescente de freqüência.

2.2 Características do Sinal de Voz

Independentemente da língua falada, as características dos sinais de voz são aproximadamente as mesmas, já que intrinsicamente a voz é gerada em sistemas de anatomia semelhante. Como visto nas seções anteriores, a voz humana é basicamente uma onda de pressão acústica radiada quando o ar é expelido pelos pulmões, passando em seguida pela laringe e pelo trato vocal. Nesse percurso pode ou não haver a vibração das cordas vocais, e podem ou não haver constrições no trato vocal. A laringe e o trato vocal atuam ainda como ressonadores, modelando o fluxo de ar com as freqüências dos formantes. O resultado final é uma onda sonora com maior parte de sua energia na faixa de freqüências de 300 a 3500 Hz, possuindo ainda um decaimento médio de cerca de 6dB/oitava.

Apesar da sensação de que de todos os elementos do trato vocal se movimentam rapidamente durante a produção da voz, do ponto de vista da produção dos fonemas, porém, pode-se dizer que o trato vocal movimenta-se de maneira extremamente lenta, chegando-se ao ponto de podermos considerá-lo aproximadamente imóvel, caso a janela de tempo considerada seja pequena o suficiente (cerca de 30ms ou menos). Assim, apesar das características do sinal da fala serem variantes no tempo (não-estacionárias), pode-se assumir sem grande erro que num pequeno segmento de tempo o sinal de voz é quasi-estacionário (localmente estacionário).

Talvez a mais importante característica de um trecho de sinal de voz, do ponto de vista da codificação, seja a sua classificação quanto à sonoridade. Sons sonoros possuem maior energia, apresentando formantes bem definidos e um forma de onda quasi-periódica, o que se deve ao fato dos sons sonoros serem gerados a partir da vibração das cordas vocais, e de não haver constrições no trato vocal. Esse tipo de sinal contem sempre uma forma de onda mais suave, apresentando uma grande correlação entre as amostras. Já os sons surdos possuem energia bem menor e freqüências mais altas, tendo forma semelhante a de

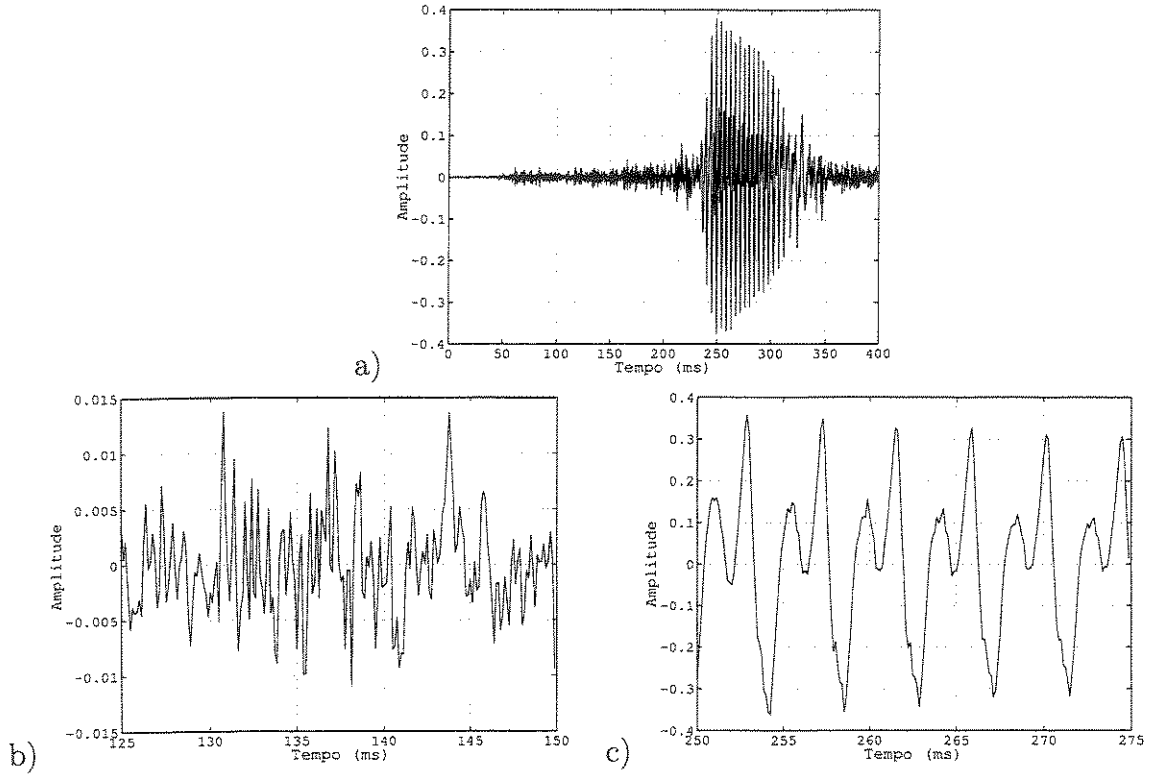


Figura 2.5: (a) Trecho inicial da palavra "sussurro". (b) Detalhe do trecho de som surdo correspondente ao fonema fricativo /s/. (c) Detalhe do trecho de som sonoro correspondente à vogal /u/. Nessa janela de tempo o sinal pode ser considerado quasi-periódico.

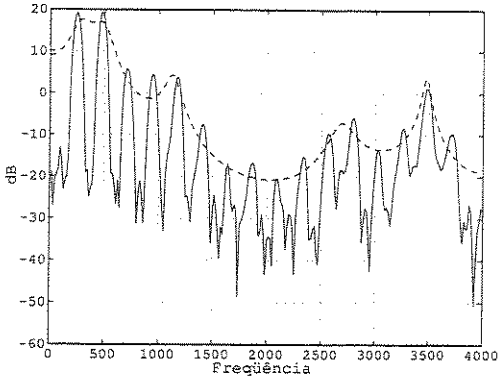


Figura 2.6: Representação em freqüência de trecho do fonema /u/, extraído da início da palavra sussurro. Pode-se notar a presença dos 4 primeiros formantes, nas freqüências de aproximadamente 400, 1200, 2700 e 3500 Hz. A linha pontilhada representa a envoltória espectral do sinal e foi obtida através de uma análise LPC com 14 coeficientes.

um ruído, colorido pelas ressonâncias do trato vocal e pela radiação labial. Esse tipo de som é geralmente produzido pela turbulência gerada quando o ar é forçado através de uma constrição no trato vocal. A Figura 2.5 mostra o sinal correspondente ao início da palavra /sussurro/, compreendendo o fonema surdo /s/ e o fonema sonoro /u/. Em 2.5(c) nota-se que o trecho sonoro possui natureza claramente periódica, sendo possível perceber, inclusive, que o período da oscilação, ou *período de pitch*, é de cerca de 5 ms. Em 2.5(b) pode-se observar a característica ruidosa do trecho surdo, que no caso é gerado pela turbulência na constrição entre a língua e o alvéolo. Deve-se salientar entretanto que alguns sons não se encaixam perfeitamente em nenhuma das duas classificações, sendo na verdade uma mistura de ambos. É o caso por exemplo das fricativas sonoras, onde existe vibração das cordas vocais e constrição no trato vocal (ex. vaca).

A presença dos formantes nos trechos de som sonoro é outra importante característica do sinal de fala. A Figura 2.6 mostra a representação em freqüência de um trecho de som sonoro correspondente ao fonema /u/. Como evidenciado na figura, o espectro de um trecho sonoro apresenta diversas harmônicas da freqüência fundamental, que no caso é de cerca de 250Hz. É interessante observar que as harmônicas assumem o formato de uma envoltória espectral onde se destacam como picos os formantes. A importância dos formantes fica clara dado o fato de que eles são os grandes responsáveis pela diferenciação dos diversos sons sonoros. Mais especificamente, apenas a presença dos três primeiros formantes já é suficiente para que a compreensão de um fonema seja possível sem a contribuição do contexto em que

foi pronunciado. As formantes correspondem às freqüências de ressonância da composição acústica formada pelo trato vocal e pela faringe, sendo determinadas basicamente pelo posicionamento desses elementos durante a produção do som. Assim, durante a produção do fonema /a/, por exemplo, o ar encontra na faringe um duto bem estreito, enquanto que o trato vocal está bastante aberto. No caso do fonema /i/ ocorre o inverso: o trato vocal comprime a passagem do ar, enquanto que a faringe está expandida. É interessante lembrar que o posicionamento dos elementos do trato vocal também é importante no caso dos fonemas surdos, onde o local e a forma da constrição são os fatores determinantes para as características espectrais do som gerado.

É possível calcular de maneira aproximada as freqüências de ressonância do sistema acima da laringe através da utilização de modelos de tubos acústicos concatenados que procuram reproduzir o formato da faringe e do trato vocal. Através de tais modelos pode-se mostrar ainda que a envoltória espectral composta pelas formantes pode ser representada pela resposta em freqüência de um filtro digital só-pólos [29, 8], onde cada par de pólos corresponde a uma freqüência de formante.

Capítulo 3

PRINCÍPIOS DE CODIFICAÇÃO LPC

Esse capítulo procura apresentar os fundamentos dos métodos de codificação de voz baseados em predição linear (LPC - *Linear Predictive Coding*).

A predição linear é uma técnica matemática que, embora não esteja limitada ao processamento de sinais de fala, possui inúmeras e importantes aplicações nesta área, constituindo-se como um dos métodos mais poderosos de modelagem e extração de características do sinal de voz [26, 29]. Dessa forma, a predição linear é utilizada tanto na síntese de voz, quanto no reconhecimento e na codificação. A aplicação da predição linear ao processamento de sinais de voz foi sugerida no final da década de sessenta, por Itakura e Saito [14], e Atal e Schroeder [3]. A aplicação das técnicas de predição linear na codificação trouxe ganhos tanto de qualidade quanto referentes à redução da taxa de transmissão.

3.1 Predição Linear

A predição linear é uma operação matemática onde valores futuros de um sinal digital são estimados como função linear de amostras passadas. Dessa forma, caso o valor futuro tenha alguma relação com os valores passados, e mais especificamente, caso essa relação seja linear, a estimação do valor futuro com base nos valores passados poderá ser feita de maneira eficaz. A idéia principal inerente à predição linear é a extração de redundância do sinal considerado, baseando-se num procedimento de subtração entre o valor predito e o valor real. O resultado dessa diferença, chamado de erro de predição ou resíduo, tende a

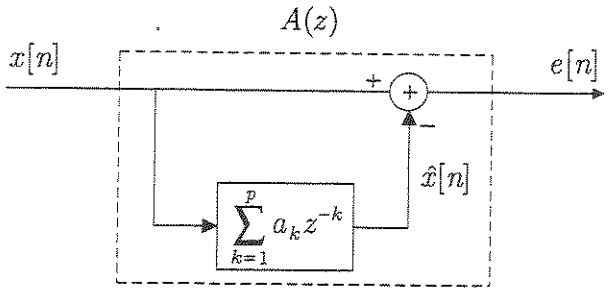


Figura 3.1: Diagrama representando a predição linear do sinal $x[n]$. O filtro FIR $A(z)$ é chamado filtro de erro de predição.

possuir um espectro branco.

Considerando um sinal representado por $x[n]$, o valor predito da n-ésima amostra desse sinal em função de p valores passados é dado por:

$$\hat{x}[n] = \sum_{k=1}^{p} a_k x[n-k], \tag{3.1}$$

onde a_k são os coeficientes da combinação linear, e $\hat{x}[n]$ é o valor estimado para $x[n]$.

O erro de predição $e[n]$ é então dado pela diferença entre o valor real e o valor estimado pelo processo linear, ou seja,

$$e[n] = x[n] - \hat{x}[n], \tag{3.2}$$

$$e[n] = x[n] - \sum_{k=1}^{p} a_k x[n-k]. \tag{3.3}$$

Aplicando a predição linear às amostras do sinal em questão, tem-se que $\hat{x}[n]$ constitui uma seqüência de valores preditos, enquanto que $e[n]$ constitui uma seqüência com os erros de predição, resultando no processo descrito no diagrama de blocos da Figura 3.1.

Do cálculo da transformada z dos termos da equação (3.3), resulta que:

$$E(z) = X(z)\left(1 - \sum_{k=1}^{p} a_k z^{-k}\right). \tag{3.4}$$

Dessa forma, existe um sistema linear que transforma o sinal $x[n]$ no erro de predição $e[n]$. Tal sistema corresponde a um filtro FIR chamado filtro de erro de predição (PEF -

Prediction Error Filter), cuja transformada A(z) é dada por:

$$A(z) = 1 - \sum_{k=1}^{p} a_k z^{-k}. \tag{3.5}$$

3.1.1 Cálculo dos Coeficientes de Predição Linear

Para determinar os coeficientes de predição linear a_k é geralmente utilizada como critério a minimização do erro quadrático médio (EQM). Ou seja, é escolhido o conjunto de coeficientes que minimize a energia do erro de predição no intervalo considerado.

Abordagem estatística

Para tratar o problema será considerado inicialmente, ao invés de um sinal de voz, um processo estocástico $\mathbf{x}[n]$, real, estacionário e de média nula. O erro de predição e os valores preditos serão representados respectivamente por $\mathbf{e}[n]$ e $\hat{\mathbf{x}}[n]$.

Desse modo, o erro quadrático médio E_m é dado por:

$$E_m = E\left\{\mathbf{e}^2[n]\right\}, \tag{3.6}$$

onde $E\{.\}$ é o operador de esperança matemática.

A escolha dos coeficientes a_j é feita de forma a minimizar E_m, ou seja, pretende-se resolver o seguinte problema:

$$\min_{\{a_j\}} E\left\{\mathbf{e}^2[n]\right\} \qquad j = 1, 2, \ldots, p. \tag{3.7}$$

O problema de minimização representado pela equação (3.7) pode ser resolvido igualando-se a zero as derivadas do erro quadrático médio em relação a cada um dos p coeficientes procurados, o que resulta em

$$\frac{\partial(E_m)}{\partial(a_j)} = \frac{\partial}{\partial(a_j)} E\left\{\mathbf{e}^2[n]\right\} = 0. \tag{3.8}$$

$$\Rightarrow E\left\{2\mathbf{e}[n]\frac{\partial(\mathbf{e}[n])}{\partial(a_j)}\right\} = 0 \tag{3.9}$$

Com base na equação (3.3) pode-se substituir o termo $\mathbf{e}[n]$ dentro da derivada:

$$E\left\{\mathbf{e}[n]\frac{\partial}{\partial(a_j)}\left(\mathbf{x}[n]-\sum_{k=1}^{p}a_k\mathbf{x}[n-k]\right)\right\}=0 \tag{3.10}$$

$$E\left\{\mathbf{e}[n]\mathbf{x}[n-j]\right\}=0 \tag{3.11}$$

A equação (3.11) representa a condição de ortogonalidade: para que o erro quadrático médio seja minimizado, a correlação cruzada entre o sinal e o erro de predição deve ser nula, ou seja, o erro deve ser perpendicular ao sinal.

A partir de (3.11), e utilizando novamente a equação (3.3) para substituir o valor de $\mathbf{e}[n]$, tem-se:

$$E\left\{\left(\mathbf{x}[n]-\sum_{k=1}^{p}a_k\mathbf{x}[n-k]\right)\mathbf{x}[n-j]\right\}=0 \tag{3.12}$$

E, como o $\mathbf{x}[n]$ é estacionário, chega-se à chamadas equações de Wiener-Hopf:

$$\sum_{k=1}^{p}a_kR_x[j-k]=R_x[j],\qquad j=1,2,\ldots,p \tag{3.13}$$

onde $R_x[j]=E\left\{\mathbf{x}[n]\mathbf{x}[n-j]\right\}$ é a auto-correlação de $\mathbf{x}[n]$.

As equações descritas em (3.13) também são chamadas de equações normais, ou ainda equações de Yule-Walker, e constituem um sistema de p equações e p incógnitas, de cuja resolução depende então o cálculo dos coeficientes de predição.

Utilizando a propriedade $R_x[-j]=R_x[j]$, pode-se representar o sistema em questão na forma matricial:

$$\mathbf{R}_x\mathbf{a}=\mathbf{r}_x, \tag{3.14}$$

onde

$$\mathbf{R}_x=\begin{bmatrix} R_x[0] & R_x[1] & R_x[2] & \ldots & R_x[p-1] \\ R_x[1] & R_x[0] & R_x[1] & \ldots & R_x[p-2] \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ R_x[p-1] & R_x[p-2] & R_x[p-3] & \ldots & R_x[0] \end{bmatrix}\in\mathbb{R}^{p\times p} \tag{3.15}$$

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \end{bmatrix} \in \mathbb{R}^p \qquad \mathbf{r}_x = \begin{bmatrix} R_x[1] \\ R_x[2] \\ \vdots \\ R_x[p] \end{bmatrix} \in \mathbb{R}^p \tag{3.16}$$

Pode-se notar facilmente que a matriz $\mathbf{R}$ é simétrica e Toeplitz. Além disso, sabe-se que a matriz de auto-correlação é definida positiva e, portanto, possui inversa [26]. A solução do sistema da equação (3.14) é dada então por:

$$\mathbf{a} = \mathbf{R}_x^{-1}\mathbf{r}_x \tag{3.17}$$

Existem vários algoritmos que, explorando o fato da matriz $\mathbf{R}_x$ ser Toeplitz, resolvem o problema do cálculo dos coeficientes de predição linear de forma computacionalmente eficiente, entre eles o algoritmo de Levinson-Durbin [26] é o mais conhecido.

Método da auto-correlação

Embora o sinal de voz não seja um processo estocástico estacionário, pode-se aplicar os resultados da seção anterior ao processamento de fala fazendo-se algumas considerações.

Geralmente no processamento de sinais de voz a predição linear é aplicada em blocos (*frames*) suficientemente pequenos do sinal, onde se pode considerá-lo como estacionário (estacionariedade local). O cálculo dos coeficientes de predição linear através da equação (3.17) pode ser feito então considerando-se uma estimação da auto-correlação [25] dada por:

$$\hat{R}_x[j] = \frac{1}{L_p} \sum_{n=0}^{L_p-j-1} x[n]x[n+j], \tag{3.18}$$

onde L_p é o número de amostras do intervalo em que é aplicada a predição.

O uso de (3.18) corresponde a considerar que o segmento de sinal analisado é uma realização de um processo estocástico estacionário e ergódico $\mathbf{x}[n]$. Pode-se notar ainda que o método da auto-correlação corresponde na verdade à aplicação de um janelamento ao sinal, o que torna nulas as amostras fora do intervalo analisado.

A predição linear em um bloco de sinal de voz pode ser feita, portanto, calculando-se as auto-correlações de acordo com (3.18) e resolvendo o sistema de equações (3.14). Pode-se notar ainda que o termo $1/L_p$ em (3.18) pode ser descartado, já que não contribui para a solução do sistema de equações normais.

Método da covariância

Para derivar o método da covariância será analisada aqui uma formulação determinística do problema do cálculo dos coeficientes de predição linear[1]. Considerando-se dessa vez o sinal real $x[n]$, o erro quadrático E_m a ser minimizado é definido apenas dentro do intervalo de L_p amostras onde será feita a predição:

$$E_m = \sum_{n=0}^{L_p-1} e^2[n] \tag{3.19}$$

O problema então corresponde a:

$$\min_{\{a_j\}} E_m = \min_{\{a_j\}} \sum_{n=0}^{L_p-1} \left(x[n] - \sum_{k=1}^{p} a_k x[n-k] \right)^2 \tag{3.20}$$

O mínimo da equação (3.20) pode ser encontrado igualando a zero a derivada do lado direito da equação (3.20) em relação a cada um dos p coeficientes procurados.

$$\frac{\partial(E_m)}{\partial(a_j)} = \sum_{n=0}^{L_p-1} \left(-2x[n]x[n-j] + 2x[n-j] \sum_{k=1}^{p} a_k x[n-k] \right) = 0 \tag{3.21}$$

$$\sum_{k=1}^{p} a_k \sum_{n=0}^{L_p-1} x[n-j]x[n-k] = \sum_{n=0}^{L_p-1} x[n]x[n-j] \qquad j = 1, 2, \ldots, p \tag{3.22}$$

A equações descritas em (3.22) são novamente as equações normais, semelhantes às equações em (3.13).

O sistema em questão pode ser representado de forma matricial:

$$\begin{bmatrix} \phi_{11} & \phi_{12} & \cdots & \phi_{1p} \\ \phi_{21} & \phi_{12} & \cdots & \phi_{2p} \\ \vdots & \vdots & & \vdots \\ \phi_{p1} & \phi_{p2} & \cdots & \phi_{pp} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \end{bmatrix} = \begin{bmatrix} \phi_{01} \\ \phi_{02} \\ \vdots \\ \phi_{0p} \end{bmatrix}, \tag{3.23}$$

[1] O método da covariância também pode ser derivado através de uma abordagem estocástica, considerando uma estimação da função de auto-correlação dependente do tempo.

onde os elementos ϕ_{jk} são dados por[2]:

$$\phi_{jk} = \sum_{n=0}^{L_p-1} x[n-j]x[n-k] \tag{3.24}$$

Como $\phi_{jk} = \phi_{kj}$, tem-se que a matriz da equação (3.23) é simétrica. Entretanto, nenhuma afirmação pode ser feita quanto a existência de solução para o sistema, posto que a matriz não possui necessariamente uma inversa. Nesse caso a matriz não é Toeplitz, e o algoritmo de Levinson-Durbin não pode ser utilizado, entretanto, a matriz ainda pode ser invertida de forma eficiente através da chamada decomposição de Cholesky, ou do método de Morf et al [36].

Note que para o cálculo dos elementos ϕ_{jk} são necessárias amostras do sinal que estão fora do intervalo de predição, ou seja, fora do intervalo onde o erro é minimizado. Por essa razão, muitas vezes define-se o problema da predição no intervalo $[0, L_p - 1]$, como a minimização do erro no intervalo $[p, L_p - 1]$. Nesse caso, o erro quadrático é dado por:

$$E_m = \sum_{n=p}^{L_p-1} e^2[n] \tag{3.25}$$

e os valores ϕ_{jk} são dados por:

$$\phi_{jk} = \sum_{n=p}^{L_p-1} x[n-j]x[n-k] \tag{3.26}$$

Em geral, o método da covariância apresenta resultados mais precisos do que os apresentados pelo método da auto-correlação. De qualquer maneira, para valores de L_p grandes, as duas abordagens convergem para valores praticamente iguais.

3.1.2 Extração de Correlação Através da Predição Linear

Uma das características principais dos filtros de erro de predição é a de extrair a correlação existente entre as amostras do sinal processado, tendendo a transformá-lo num

[2]Embora na área de processamento da fala esse método tenha recebido o nome de método da covariância, os elementos ϕ_{jk} não correspondem ao conceito estatístico de covariância do sinal, ou seja, à auto-correlação com a média removida. O nome na verdade deve-se ao fato de que a matriz da equação (3.23) possui propriedades semelhantes às da matriz de covariância.

sinal de espectro branco. Ao sofrer tal ação diz-se que o sinal foi "branqueado", e ao filtro em questão dá-se o nome de filtro de "branqueamento", do inglês *whitening filter*.

Essa propriedade dos filtros de erro de predição pode ser esclarecida analisando-se a função de auto-correlação do sinal de saída do filtro. Para isso será considerado inicialmente um processo de predição linear de ordem infinita, onde os coeficientes de predição a_k, $k = 1, 2, \ldots$, constituem o filtro de erro de predição $A(z)$. Será considerado ainda que o erro $e[n]$ e o sinal $x[n]$ são representados pelos processos estacionários $\mathbf{e}[n]$ e $\mathbf{x}[n]$ respectivamente.

A auto-correlação do erro de predição é então dada por:

$$R_e(j) = E\{\mathbf{e}[n]\mathbf{e}[n-j]\} \tag{3.27}$$

$$R_e(j) = E\left\{\mathbf{e}[n]\left(\mathbf{x}[n-j] - \sum_{k=1}^{\infty} a_k \mathbf{x}[n-j-k]\right)\right\} \tag{3.28}$$

$$R_e(j) = E\{\mathbf{e}[n]\mathbf{x}[n-j]\} - \sum_{k=1}^{\infty} a_k E\{\mathbf{e}[n]\mathbf{x}[n-j-k]\} \tag{3.29}$$

Pela suposta estacionariedade dos processos e de acordo com o princípio da ortogonalidade, representado pela equação (3.11), nota-se que:

$$R_e(j) = 0 \qquad \forall j \neq 0 \tag{3.30}$$

O resultado da última equação mostra que, em um preditor de ordem infinita, as amostras do erro de predição não possuem qualquer correlação entre si. Ou seja, caso o sinal analisado seja estacionário, a saída de um filtro de erro de predição de ordem infinita é um ruído branco.

Além disso, substituindo (3.3) em (3.28), e utilizando em seguida a equação de Wiener-Hopf, pode-se provar ainda que a energia do erro é dada por:

$$\sigma_e^2 = R_e(0) = R_x(0) - \sum_{k=1}^{\infty} a_k R_x(k) \tag{3.31}$$

Embora o desenvolvimento acima tenha sido feito para um preditor de ordem infinita, é fácil notar que a redução da correlação acontece também em filtros de ordem finita. De fato, quanto maior a ordem do preditor, maior a sua capacidade de transformar o sinal num sinal de espectro branco.

Deve-se ressaltar ainda que os filtros de erro de predição contêm importantes in-

formações espectrais sobre o sinal analisado. Para verificar esse fato basta notar que o filtro $1/A(z)$ transformaria o erro $e[n]$ de volta no sinal $x[n]$:

$$X(z) = \frac{1}{A(z)}E(z) \tag{3.32}$$

O espectro de $E(z)$ tende a ser branco, e em particular no caso do preditor de ordem infinita tem-se que:

$$X(e^{jw}) = \frac{1}{A(e^{jw})}\sigma_e^2 \tag{3.33}$$

Ou seja, o filtro $1/A(z)$ possui espectro semelhante ao do sinal analisado.

3.1.3 Predição Linear e Processos Auto-Regressivos

Diz-se que um processo estocástico $\mathbf{y}[n]$ foi gerado a partir de um modelo auto-regressivo de ordem p, também chamado de $AR(p)$, quando vale a regra:

$$\mathbf{y}[n] = \sum_{k=1}^{p} a'_k \mathbf{y}[n-k] + \mathbf{v}[n], \tag{3.34}$$

onde a'_k são coeficientes do processo e o processo $\mathbf{v}[n]$ é um ruído branco, estacionário e de média nula.

O sistema da Figura 3.2 representa a geração do processo $\mathbf{y}[n]$, que pode ser considerada como a filtragem do ruído branco $\mathbf{v}[n]$ por um filtro de resposta ao impulso infinita $S(z)$, dado por:

$$S(z) = \frac{1}{1 - \sum_{k=1}^{p} a'_k z^{-k}} \tag{3.35}$$

O ruído branco $\mathbf{v}[n]$ que alimenta o filtro $S(z)$ na Figura 3.2 evidentemente não possui qualquer correlação entre suas amostras. Entretanto, a atuação do filtro recursivo faz com que as amostras na saída do filtro estejam correlacionadas.

Uma relação entre a auto-correlação do processo auto-regressivo $\mathbf{y}[n]$ e os coeficientes do processo a'_k pode ser obtida multiplicando a equação (3.34) por $\mathbf{y}[n-j]$ e em seguida calculando-se a esperança matemática:

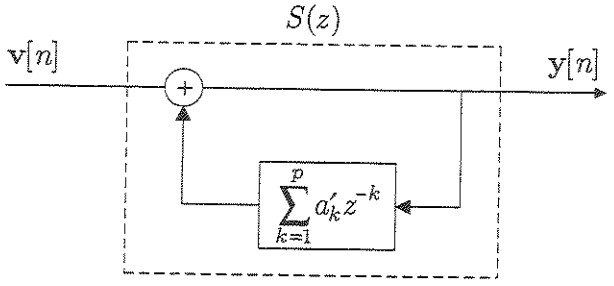


Figura 3.2: Diagrama representando a geração do processo auto-regressivo $\mathbf{y}[n]$ de ordem p.

$$E\{\mathbf{y}[n]\mathbf{y}[n-j]\} = E\left\{\sum_{k=1}^{p} a'_k \mathbf{y}[n-k]\mathbf{y}[n-j]\right\} + E\{\mathbf{v}[n]\mathbf{y}[n-j]\} \tag{3.36}$$

E, como $\mathbf{v}[n]$ é um ruído de média nula e sem correlação com $\mathbf{y}[n]$,

$$E\{\mathbf{y}[n]\mathbf{y}[n-j]\} = \sum_{k=1}^{p} a'_k E\{\mathbf{y}[n-k]\mathbf{y}[n-j]\} \tag{3.37}$$

$$\sum_{k=1}^{p} a'_k R_y[j-k] = R_y[j] \qquad j = 1, 2, \ldots, p \tag{3.38}$$

Nota-se imediatamente a semelhança entre essa última equação e a equação de Yule-Walker em (3.13). De fato, as duas equações são idênticas, o que significa que caso um sinal $x[n]$ tenha sido gerado por um processo $AR(p)$, num processo de predição linear de $x[n]$ de ordem p, os coeficientes de predição que minimizam o erro quadrático médio serão idênticos aos coeficientes do processo auto-regressivo que gerou o sinal $x[n]$. Ou seja, considerando que o sinal $x[n]$ da Figura 3.1 foi gerado através de uma realização de um processo auto-regressivo de ordem p com os coeficientes a'_k, os coeficientes de predição a_k que minimizam o erro quadrático médio são:

$$a_k = a'_k \qquad k = 1, 2, \ldots, p \tag{3.39}$$

Ou seja, o filtro $S(z)$ é na verdade o inverso do filtro de erro de predição $A(z)$:

$$S(z) = \frac{1}{A(z)} \tag{3.40}$$

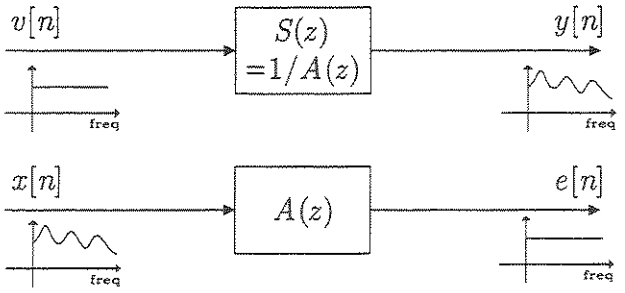


Figura 3.3: Característica de "branqueamento" do filtro de erro de predição $A(z)$, e a geração do sinal $y[n]$ a partir de um processo auto-regressivo.

Além disso, pela comparação entre as equações (3.3) e (3.34), pode-se mostrar facilmente que o erro de predição será igual ao ruído branco que gerou o processo auto-regressivo:

$$e[n] = v[n] \tag{3.41}$$

Conforme destacado na Figura 3.3, as duas últimas equações confirmam que o filtro de erro de predição atua de forma a reduzir a correlação existente entre as amostras do sinal considerado, tendendo a transformá-lo num sinal de espectro branco. O interessante aqui é que, caso o sinal analisado tenha sido gerado por um processo auto-regressivo de ordem igual ou menor do que a ordem do preditor, o erro de predição será de fato um ruído branco. Ou seja, para analisar um sinal gerado por um processo $AR(p)$, é suficiente utilizar um preditor de ordem p; um preditor de ordem maior não conseguirá reduzir ainda mais o erro.

3.2 Codificação de Voz Baseada na Predição Linear

A capacidade de extrair características espectrais do sinal de voz através da predição linear de maneira suficientemente precisa e computacionalmente eficiente pode ser utilizada facilmente em esquemas de codificação da voz. A idéia no caso seria a codificação apenas dos coeficientes de predição e de alguma informação sobre o resíduo.

3.2.1 Processamento Segmentado do Sinal de Voz

O sinal de voz é geralmente codificado num esquema bloco-a-bloco, onde os blocos (*frames*) possuem usualmente cerca de 10 a 30ms de sinal, o que corresponde a cerca de 80

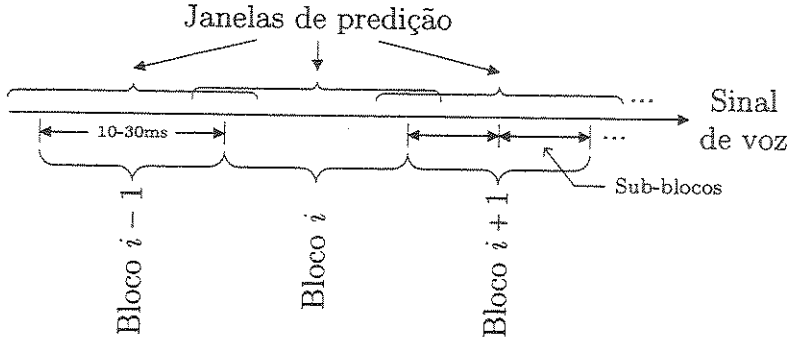


Figura 3.4: Segmentação do sinal de voz para a análise preditiva. Em alguns casos é utilizada uma janela de predição maior do que o bloco a ser codificado.

a 240 amostras de sinal amostrado a $8kHz$. Em segmentos pequenos como estes a voz pode ser considerada como estacionária, e caso se trate de um trecho de som sonoro, o segmento será quasi-periódico. A Figura 2.5 mostra trechos de $25ms$ de som surdo e sonoro.

A predição linear é aplicada muitas vezes em trechos maiores do que o bloco a ser codificado, havendo então uma sobreposição entre as janelas de análise preditiva, ou janelas de predição. A Figura 3.4 mostra o esquema em questão. Na verdade, durante a codificação de um certo bloco, a extração de características espectrais do trecho é feita geralmente utilizando-se também algumas amostras anteriores e posteriores ao bloco.

A Figura 3.4 mostra ainda a divisão dos blocos em sub-blocos, que ocorre em vários codificadores baseados em predição linear. Enquanto que a transição de um bloco para outro representa a adoção de novos coeficientes de predição, a transição de sub-blocos dentro do mesmo bloco representa a mudança de outros parâmetros, como por exemplo novas informações sobre a excitação a ser usada na reconstrução da voz. O Capítulo 4 apresenta maiores detalhes sobre a segmentação da voz durante a codificação.

3.2.2 Resíduo LPC do Sinal de Voz

A Figura 3.5 mostra o resíduo resultante da análise preditiva de um bloco de $25ms$ de som sonoro feita com 12 coeficientes de predição, número semelhante ao utilizado em vários em padrões de codificação de voz atuais. Em virtude de lidar apenas com a correlação entre amostras próximas, esse tipo de análise é chamada de predição de curta duração, ou STP (do inglês *Short-Term Prediction*).

Pode-se notar na Figura 3.5 que o erro de predição decorrente da análise de curta

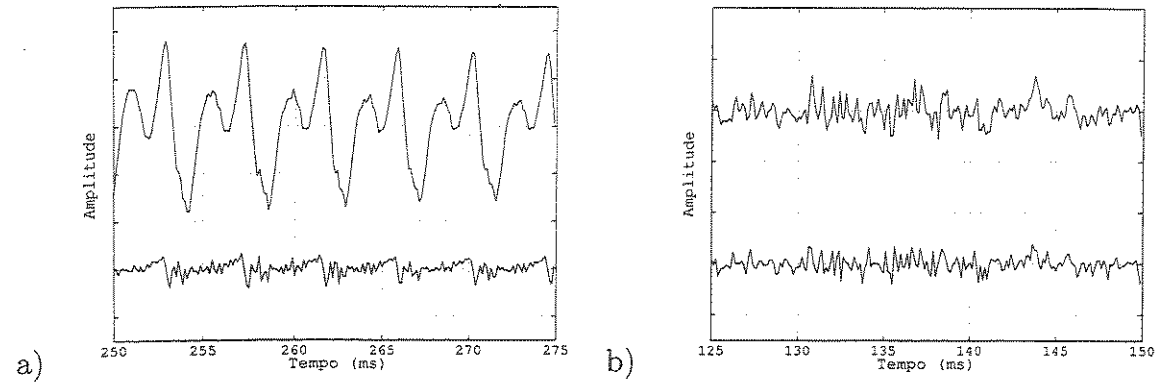


Figura 3.5: Resíduo LPC resultante de uma análise preditiva de curta duração com 12 coeficientes. (a) Trecho de som sonoro (curva superior) e resíduo LPC correspondente (b) Trecho de som surdo (curva superior) e resíduo LPC correspondente.

duração está longe ser um sinal branco, apresentando consideráveis variações no espectro. Ocorre que a análise STP, como o próprio nome sugere, não é eficaz para extrair a chamada correlação de longo prazo, correlação existente entre amostras distantes entre si no domínio temporal. Resultado desse fato é que em trechos de som sonoro o resíduo STP apresenta oscilações com o período de pitch, que fazem com que o resíduo se assemelhe a uma seqüência de pulsos periódicos. No caso dos trechos de som surdo, como não há vibração das cordas vocais e conseqüente oscilação com a freqüência fundamental, o erro de predição realmente se assemelha a um ruído.

3.2.3 Modelo Para a Produção da Voz

A Figura 3.6 mostra um modelo discreto de produção da voz adotado na codificação baseada em predição linear. Esse modelo é geralmente chamado de fonte-filtro, e é descendente do modelo proposto por Fant em 1960 [11]. Na Figura 3.6 a síntese da voz é feita representando o trato vocal por um filtro só pólos com função de transferência $S(z)$, que é alimentado por um sinal escolhido como excitação. Por essa razão o filtro $S(z)$ recebe o nome de *filtro de síntese*.

A utilização de um filtro só-pólos para representar o trato vocal está de acordo com os resultados obtidos quando se modela o trato vocal através de uma concatenação de tubos acústicos sem perdas [29, 8]. Além disso, embora se saiba que os sons nasalizados são melhor representados com a introdução de zeros no filtro de síntese, a mesma melhoria pode ser

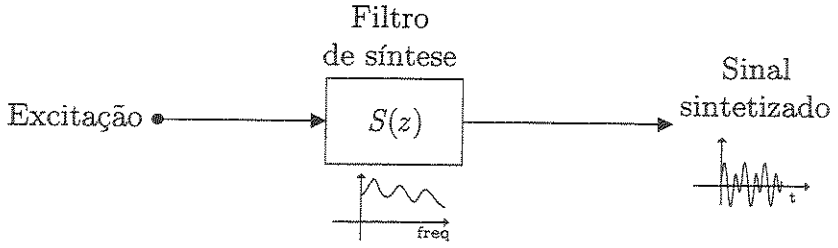


Figura 3.6: Modelo de produção da voz usado na codificação baseada em predição linear. O filtro de síntese $S(z)$ é um filtro só-pólos e a excitação é periódica ou ruidosa, de acordo com o tipo de som a ser produzido, sonoro ou surdo.

obtida simplesmente aumentando-se o número de pólos [4]. Além disso, com a utilização de um filtro só-pólos para a modelagem do sinal de voz, os picos em freqüência tendem a ser melhor representados do que os vales [26], o que resulta em uma boa representação para os formantes.

Pode-se notar ainda que existe uma semelhança entre o modelo de produção da voz da Figura 3.6 e um processo auto-regressivo, como o descrito pela equação (3.34) e apresentado na Figura 3.2. Em ambos os casos o sinal de saída é formado pela passagem de um sinal de entrada por um filtro só-pólos. Ou seja, a adoção do modelo fonte-filtro da Figura 3.6, corresponde à suposição de que a voz está sendo gerada por um modelo auto-regressivo [8].

3.2.4 Análise e Reconstrução da Voz

A grande relação entre os processos auto-regressivos e o processamento de voz através da predição linear deve-se ao fato da voz poder ser bem modelada por um filtro auto-regressivo. Com isso, o sinal de voz pode ser analisado através da predição linear e em seguida reconstruído, utilizando um esquema semelhante ao da Figura 3.6.

Reconstrução ideal

Considerando a análise preditiva de um sinal de voz $x[n]$, onde são utilizados os p coeficientes de predição a_k, $k = 1, 2, \ldots, p$, e em seguida rearranjando os termos da equação (3.3), pode-se chegar ao seguinte resultado:

$$x[n] = e[n] + \sum_{k=1}^{p} a_k x[n-k] \tag{3.42}$$

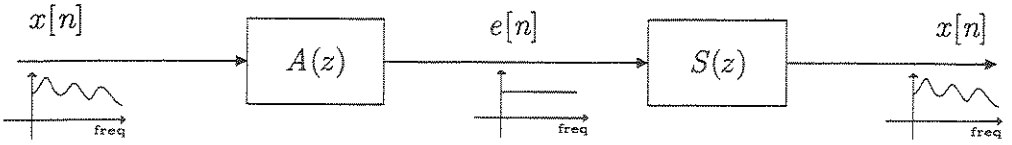


Figura 3.7: Reconstrução ideal do sinal de voz.

A equação (3.42) leva a uma conclusão importante: *o sinal $x[n]$ pode ser gerado simplesmente a partir do erro de predição $e[n]$ e dos coeficientes de predição a_k.*

Como ilustrado na Figura 3.7, a reprodução do sinal original a partir do erro e dos coeficientes de predição pode ser interpretada como a passagem do erro $e[n]$ pelo filtro $S(z)$, que possui resposta ao impulso infinita, e função de transferência dada por:

$$S(z) = \frac{1}{1 - \sum_{k=1}^{p} a_k z^{-k}} = \frac{1}{A(z)} \tag{3.43}$$

No processo da Figura 3.7, o sinal $x[n]$ é inicialmente transformado no resíduo $e[n]$ pelo filtro $A(z)$ e, em seguida, é reconstruído ao passar pelo filtro $S(z)$. Por essa razão, enquanto $S(z)$ é chamado de filtro de síntese, o filtro $A(z)$ recebe o nome de *filtro de análise* (sendo também chamado de filtro inverso).

$$E(z) = A(z)X(z) \Rightarrow X(z) = S(z)E(z) \tag{3.44}$$

O filtro de síntese $S(z)$, que deve ser formado com os coeficientes de predição a_k, irá representar a envoltória freqüencial do sinal de voz analisado. Além disso, o sinal de excitação $e[n]$, que alimenta o filtro de síntese, terá espectro aproximadamente branco.

Ao observar a resposta em freqüência do filtro de síntese, pode-se notar geralmente a presença de picos, que na verdade correspondem às freqüências de ressonância do trato vocal. Cada um desses picos corresponde à um par de pólos do filtro de síntese.

3.2.5 Codificação Através da Modelagem da Excitação

De acordo com as equações (3.42) e (3.44), a excitação que produz a reconstrução ideal do sinal de fala é o próprio erro de predição. A idéia por trás da codificação baseada em predição linear está então numa representação mais simples do erro de predição, mesmo que isso traga alguma degradação ao sinal sintetizado.

Assim, a grosso modo, a voz pode ser comprimida de maneira eficiente analisando-

se o sinal bloco-a-bloco e aplicando-se a predição linear em cada bloco. A série temporal de amostras do bloco analisado pode então ser representada pelos coeficientes de predição calculados e por um sinal que se assemelhe ao erro de predição. É justamente a maneira de representar o erro de predição que constitui a maior diferença entre os codificadores atuais baseados em predição linear.

A seguir serão apresentados dois tipos de modelagem para a excitação do sistema fonte-filtro. Ao assumir um modelo para a excitação, não é feita a codificação do erro de predição encontrado, mas sim dos parâmetros do modelo da excitação. No capítulo seguinte será abordado um outro método de codificação, que consiste em tentar codificar o próprio erro de predição, tão eficientemente quanto possível.

Modelo de excitação por trem de impulsos ou ruído branco

Uma abordagem simples, mas que leva a uma grande redução da quantidade de informação a ser transmitida, é considerar que a excitação a ser submetida ao filtro de síntese é simplesmente um ruído ou uma seqüência de impulsos periódicos. Esse esquema está representado na Figura 3.8(a), e se baseia no modelo proposto por Fant [11], resultante do estudo do sistema de produção da voz. Além disso, como apresentado na Seção 3.2.2, o resíduo LPC realmente assemelha-se à uma seqüência de pulsos periódicos, no caso de trechos de som sonoro, ou a um ruído, no caso de trechos de som surdo, o que justifica a adoção do modelo de excitação em questão.

O modelo onde a excitação é escolhida entre um ruído e um trem de impulsos também é chamado de modelo de excitação de dois estados (ou ainda de modelo de excitação binária), e tem como representante mais conhecido o padrão FS-1015 [27] (*Federal Standard* 1015) de 1984[3], a primeira especificação de codificação de voz a utilizar a predição linear. O FS-1015 tem taxa de transmissão de 2.4kbps, e também é conhecido como LPC-10, por fazer uso de um esquema de predição linear de décima ordem.

Em codificadores que fazem uso do modelo de excitação de dois estados, o erro de predição não é transmitido. Em seu lugar é codificado apenas o ganho, a informação que diz se o bloco é sonoro ou surdo, e o período de pitch. A indicação de sonoridade é utilizada na decodificação para determinar se deve ser usado um trem de impulsos ou um ruído para reconstruir o bloco.

[3]Embora o padrão FS-1015 tenha sido estabelecido somente em 1984, o algoritmo já existia desde 1976, quando foi desenvolvido para uso militar por um consórcio estabelecido pelo Departamento de Defesa Norte-Americano (DoD - *Department of Defense*).

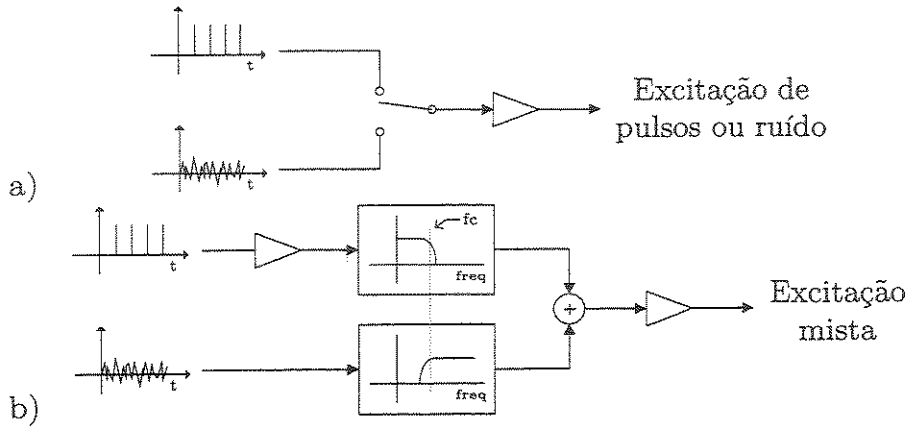


Figura 3.8: Alguns modelos de excitação utilizados. (a) Modelo de excitação clássico, onde a excitação é escolhida entre um ruído e uma seqüência de pulsos. (b) Modelo de excitação mista proposto por Makhoul et al [24].

Durante o processamento de um bloco de sinal, codificadores que fazem uso do modelo da Figura 3.8(a) têm as seguintes tarefas principais (não necessariamente nessa ordem):

- Calcular os coeficientes de predição;
- Definir se o bloco corresponde a um trecho de som sonoro ou surdo;
- Calcular o período de pitch (no caso de som sonoro);
- Calcular o ganho a ser utilizado na geração da excitação.

O cálculo dos coeficientes de predição pode ser feito através do método da auto-correlação, usando o algoritmo de Levinson-Durbin, ou através do método da covariância, usando a decomposição de Cholesky. Já o ganho a ser usado na reconstrução do bloco é geralmente escolhido de forma a fazer com que a energia do sinal sintetizado seja igual à do sinal original.

A detecção da sonoridade, por sua vez, pode ser feita através da análise da energia e da taxa de cruzamento por zero[4] do bloco. Como exposto no item 2.2, os sinais surdos têm energia bem menor do que os sinais sonoros. Além disso, trechos de som surdo, por terem característica ruidosa e freqüências mais altas, possuem taxas de cruzamento por zero maiores.

[4]A taxa de cruzamento por zero é definida como o número médio de vezes em que a seqüência discreta de amostras muda de sinal.

Modelo de excitação mista

Apesar de possibilitar a transmissão da voz em taxas extremamente baixas, a utilização do modelo de excitação de dois estados causa perdas de qualidade consideráveis. Um dos problemas desse modelo é a sua inadequação para representar fonemas fricativos sonoros e situações de transição entre trechos sonoros e surdos, onde a voz apresenta características dos dois tipos de som. Além disso, erros na decisão quanto à sonoridade do bloco podem acabar por degradar de forma comprometedora a qualidade e a inteligibilidade do sinal reconstruído. Todos esses fatores levaram ao desenvolvimento de outros modelos de representação da excitação para o sistema fonte-filtro, entre os quais encontram-se os modelos de excitação mista.

Codificadores que utilizam modelos de excitação mista recebem o nome de codificadores MELP (*Mixed-Excitation Linear Predictive*). A Figura 3.8(b) mostra o modelo de excitação mista proposto por Makhoul et al., em 1978. Nesse modelo o trem de impulsos e o ruído são filtrados e somados para compor a excitação. Dessa forma os impulsos agem somente sobre as baixas freqüência do filtro de síntese, enquanto que o ruído excita a região de altas freqüências. A freqüência de corte usada nos filtros é a mesma e é calculada a cada bloco, de acordo com o caráter sonoro ou surdo do mesmo.

3.2.6 Detecção do Período de Pitch

Nos processos de codificação de voz baseados em predição linear geralmente é necessário executar a detecção do período de pitch do sinal a cada bloco. Tal tarefa é necessária, por exemplo, nos codificadores baseados nos dois modelos de excitação apresentados na Seção 3.2.5, onde o período de pitch detectado será necessário posteriormente na decodificação para a geração do trem de pulsos a ser usado como excitação do filtro de síntese.

Ao contrário do que se pode imaginar, a determinação do período de pitch não é uma tarefa simples. A presença de ruído ou de imperfeições consideráveis na periodicidade do bloco de sinal de voz analisado são fatores que dificultam a estimação do pitch. Além disso, as ressonâncias do trato vocal podem alterar de maneira comprometedora os efeitos causados pela vibração da glote. Tais problemas levaram ao desenvolvimento de algoritmos mais complexos, que são capazes de estimar o pitch com precisão até sub-amostral. No Capítulo 4 será dada maior ênfase aos algoritmos utilizados em codificadores CELP, enquanto que a

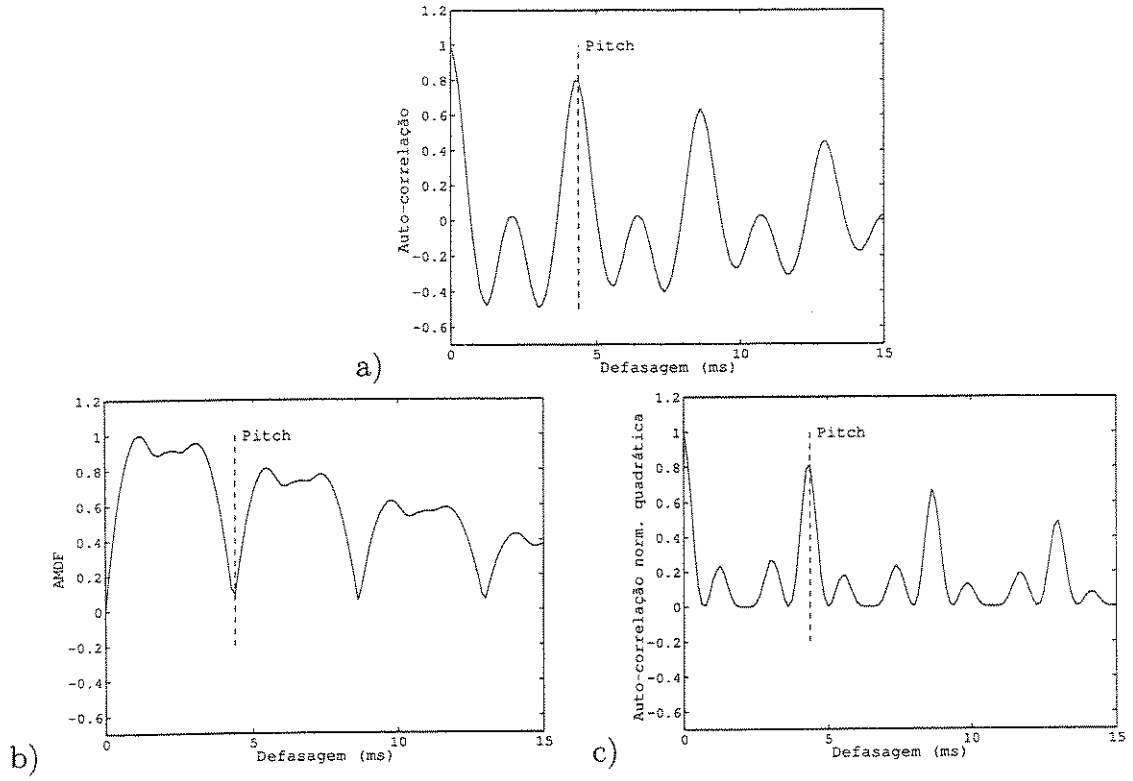


Figura 3.9: Estimação do período de pitch. (a) Função de auto-correlação. (b) AMDF. (c) Auto-correlação normalizada (quadrática). As figuras apresentam valores normalizados e correspondem ao trecho de $25ms$ de som sonoro da Figura 2.5, item (c).

seguir serão apresentadas de maneira breve algumas técnicas de detecção do pitch que se baseiam na análise do sinal no domínio temporal[5].

Os PDAs que trabalham no domínio temporal procuram encontrar o período de pitch através da comparação entre o sinal original e sua versão deslocada no tempo. Uma das técnicas mais triviais é a simples análise da auto-correlação do sinal, que deve apresentar picos nos pontos onde a defasagem é igual a múltiplos do pitch. A Figura 3.9(a) apresenta a função de auto-correlação, calculada de acordo com (3.18), de um trecho de $25ms$ de som sonoro.

Outra técnica temporal utilizada é a AMDF (*Average Magnitude Difference Function*), que se baseia na análise da seguinte função:

[5]Os algoritmos de detecção de pitch, também chamados de PDAs (*Pitch Detection Algorithms*), podem ser divididos entre aqueles que se baseiam na análise do sinal no domínio freqüencial, no domínio temporal, ou em ambos.

$$AMDF(j) = \frac{1}{L_p} \sum_{n=0}^{L_p-1} |x[n] - x[n+j]| \tag{3.45}$$

A AMDF [27] é chamada de medida de *anti-correlação*, ou ainda de medida de dissimilaridade, já que o valor da função se aproxima de zero nos pontos onde a correlação é maior. Como vantagem sobre a auto-correlação a AMDF apresenta menor complexidade computacional e uma maior faixa dinâmica[6] para o sistema, que é especialmente importante em implementações em processadores de ponto-fixo.

Também é utilizada para a detecção do período de pitch a análise da auto-correlação normalizada, dada por:

$$R_{NORM}(j) = \frac{\sum_{n=0}^{L_p-1} x[n]x[n+j]}{\sqrt{\sum_{n=0}^{L_p-1} x^2[n+j]}} \tag{3.46}$$

A auto-correlação normalizada exige um esforço computacional bem maior do que o exigido pela AMDF ou pela auto-correlação direta, entretanto apresenta melhores resultados no que diz respeito à detecção do pitch, pois evidencia melhor os picos que correspondem ao período de oscilação. A Figura 3.9(c) mostra a estimação do período de pitch feita com a auto-correlação normalizada (quadrática).

Escolhida a função a ser analisada o próximo passo consiste na elaboração de um algoritmo que selecione da curva calculada o ponto que corresponde ao pitch. Em geral são utilizados procedimentos baseados em limiares calibrados através de testes exaustivos. No caso da auto-correlação, por exemplo, pode-se procurar a partir da defasagem zero o pico que está acima de um certo limite, uma certa fração de $R(0)$.Um dos problemas comuns que podem ocorrer nesse processo é a escolha de um múltiplo do pitch. Outro problema comum é que muitas vezes ressonâncias fortes do trato vocal apresentam-se como sub-múltiplos do período de pitch. Procura-se resolver tais problemas através da verificação da presença ou não de picos consideráveis em múltiplos e sub-múltiplos do período encontrado, procurando confirmar assim se o período encontrado se trata realmente do pitch ou de algum múltiplo ou sub-múltiplo dele.

[6] A faixa dinâmica (*dynamic range*) de um sistema pode ser definida como a relação entre o maior e o menor valor numérico do sinal que pode ser tratado pelo sistema.

Capítulo 4

CODIFICADORES CELP

A codificação baseada em predição linear com excitação de códigos, ou simplesmente codificação CELP (*Code-Excited Linear Prediction*), foi proposta inicialmente em 1985, por Schroeder e Atal [35], tornando-se em seguida um dos métodos de maior sucesso na codificação de voz. Vários padrões modernos, que procuram transmitir sinais de voz de boa qualidade a taxas extremamente baixas, adotam a codificação CELP ou esquemas derivados dela, como o ACELP (*Algebraic* CELP) ou o VSELP (*Vector Sum Excited Linear Prediction*).

A codificação CELP é uma variação da codificação LPC apresentada no capítulo anterior, e utiliza um conjunto (dicionário) de seqüências de amostras (códigos) que correspondem à possíveis representantes do erro de predição. Assim, na codificação de cada bloco de sinal, é escolhido, através de uma técnica denominada análise-por-síntese, o componente desse conjunto que melhor representa a excitação ideal. No processo são consideradas ainda características do sistema auditivo humano através da aplicação de um filtro de ponderação espectral perceptiva.

4.1 Codificação Com Dicionários de Códigos

No capítulo anterior foi mostrado que após uma análise preditiva, a reconstrução ideal do sinal original poderia ser obtida usando-se o próprio erro de predição como excitação de um filtro só-pólos correspondente ao inverso do filtro de análise (Figura 3.7). De acordo com esse esquema, após uma análise preditiva bloco-a-bloco, um sinal de voz poderia ser transmitido sem qualquer perda através da transmissão do erro e dos coeficientes de predição

de cada bloco. Entretanto, a transmissão do próprio sinal de erro completo não representa nenhuma vantagem. Dessa forma, alguns esquemas de codificação baseiam-se na transmissão dos coeficientes de predição e de apenas algumas informações sobre a excitação a ser utilizada no filtro de síntese. Geralmente a informação sobre a excitação diz respeito principalmente à sonoridade do trecho analisado, sendo a reconstrução do sinal feita usando-se como excitação um ruído ou um trem de impulsos, ou ainda uma combinação elaborada de ambos. Nesses sistemas, uma redução da quantidade de informação a ser transmitida é alcançada ao custo de certa perda na qualidade da voz.

Um esquema mais elaborado de codificação de voz consiste na codificação eficiente do próprio erro de predição. A princípio pode-se pensar que, por ser em última análise um sinal como outro qualquer, a transmissão do resíduo seria tão complexa como a transmissão do próprio sinal original. Entretanto, resultados práticos mostram que a representação apenas aproximada do resíduo pode produzir voz de qualidade satisfatória, e é essa a idéia associada à codificação com excitação de códigos.

Na codificação CELP é escolhido para representar o erro de predição um dos sinais de um banco de possíveis candidatos. Tal conjunto de candidatos é geralmente chamado de dicionário de códigos (*codebook*), e os elementos do dicionário são segmentos de sinal chamados de inovações, vetores-código (*codevectors*), ou ainda simplesmente códigos. No esquema inicial, proposto por Schroeder e Atal [35] em 1985, o dicionário possuía vetores-código constituídos de amostras de um processo de ruído branco, de média nula e variância unitária. Hoje, entretanto, são utilizados dicionários de códigos de composições diversas, e uma das técnicas de maior sucesso é a adoção de dicionários esparsos [23].

A Figura 4.1 mostra de maneira simplificada o processo de codificação com a utilização de dicionários de códigos. Na codificação de cada bloco as características espectrais do trecho analisado são extraídas através da predição linear. Logo após, juntamente com o ganho correspondente, é selecionada uma das entradas do dicionário para representar o resíduo. Na decodificação o sinal é reconstruído aplicando-se o ganho ao vetor-código selecionado e, em seguida, utilizando-se o resultado para alimentar o filtro de síntese.

4.2 Análise-Por-Síntese

Nos codificadores modernos a escolha do vetor do dicionário que melhor representa o resíduo não é feita tendo-se como base a semelhança entre o resíduo e o vetor-código. Na

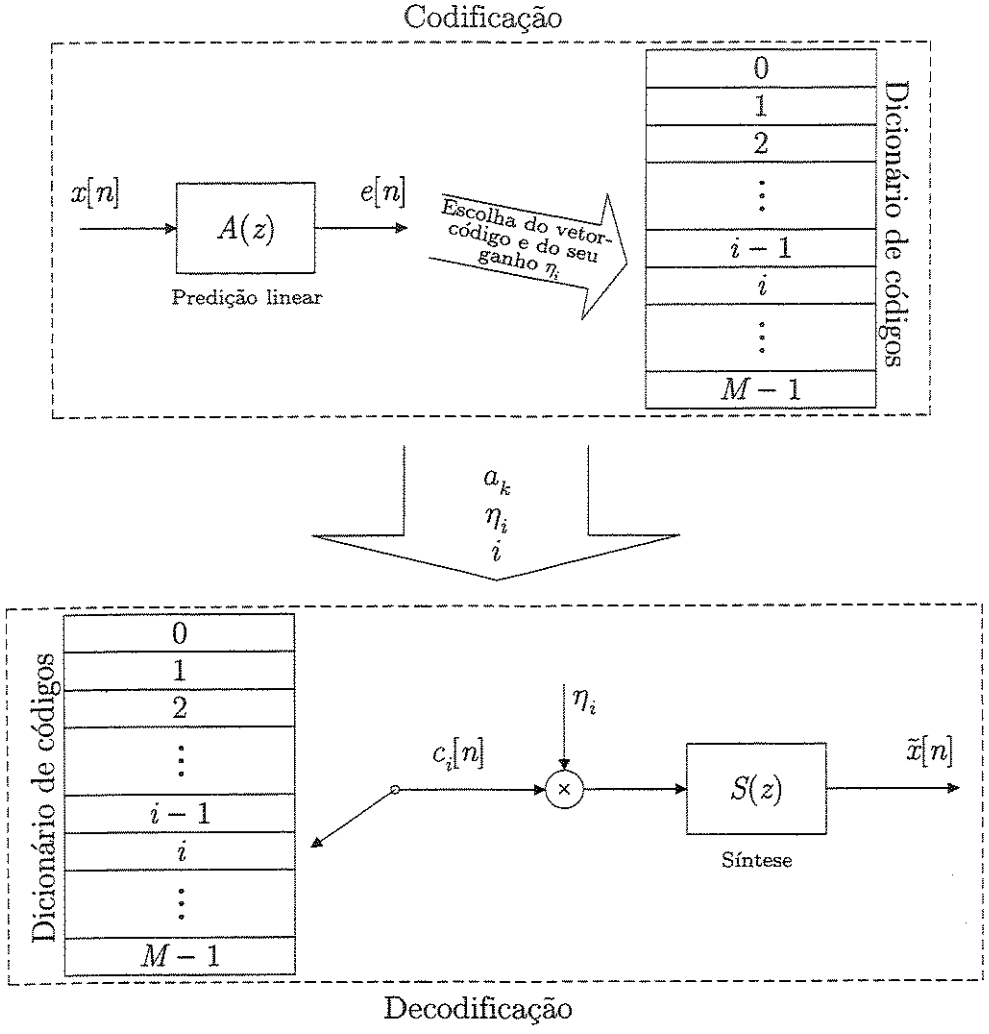


Figura 4.1: Esquema simplificado da codificação através de dicionários de códigos. São codificados apenas os coeficientes de predição, um ganho, e a entrada do dicionário escolhida através da análise por síntese

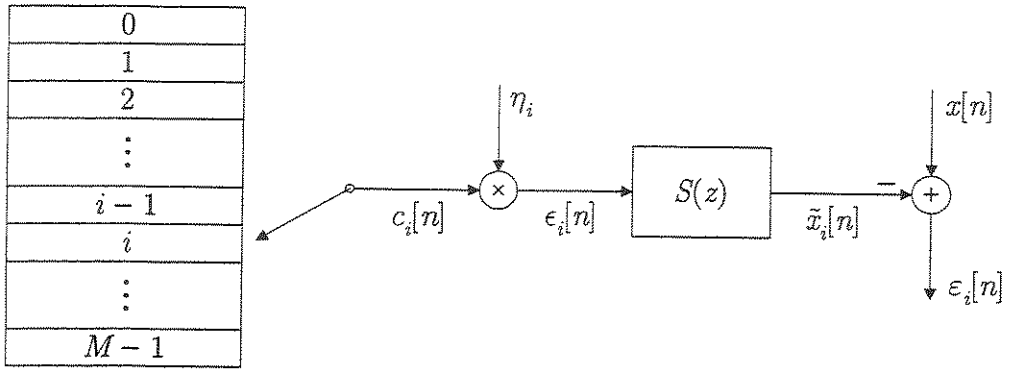


Figura 4.2: Esquema simplificado de reconstrução da voz nos codificadores CELP. Geração do erro a ser minimizado na seleção de inovações através da análise-por-síntese.

verdade, a escolha é feita através da comparação do sinal original com o sinal reconstruído para cada vetor do dicionário. Ou seja, deve ser selecionado o vetor-código que produza o sinal sintetizado que melhor aproxime o sinal original. Esse critério de busca no dicionário é denominado análise-por-síntese (*analysis-by-synthesis*), e foi proposto formalmente pela primeira vez por Schroeder e Atal juntamente com a codificação CELP. Codificadores que utilizam o processo de análise-por-síntese para selecionar o vetor-código a ser codificado são chamados de codificadores LPAS (*Linear Predictive Analysis-by-Synthesis*). O processo de seleção do melhor vetor-código do dicionário é chamado usualmente de processo de busca (*search process*), ou ainda simplesmente de busca.

A Figura 4.2 descreve o processo a ser executado na fase de codificação para que seja escolhido o vetor-código que minimize a diferença entre o sinal original e o sinal que será sintetizado na decodificação. Na Figura 4.2 o dicionário possui M vetores-código representados por $c_i[n]$, $i = 0, 1, \ldots, M-1$. A excitação $\epsilon_i[n]$ correspondente a cada vetor-código é gerada após a multiplicação do mesmo por seu ganho η_i. O sinal sintetizado $\tilde{x}_i[n]$ é gerado na saída do filtro de síntese com função de transferência $S(z)$ dada por:

$$S(z) = \frac{1}{1 - \sum_{k=1}^{p} a_k z^{-k}}, \tag{4.1}$$

onde a_k são os coeficientes de predição linear de ordem p.

O erro $\varepsilon_i[n]$ gerado pela utilização i-ésimo vetor-código $c_i[n]$ recebe o nome de erro de quantização, ou ainda erro de reconstrução, e é dado pela diferença entre o sinal sintetizado e o sinal original:

$$\varepsilon_i[n] = x[n] - \tilde{x}[n] \tag{4.2}$$

Geralmente é definido como critério a minimização da energia do erro $\varepsilon_i[n]$. Com essa consideração o objetivo da análise-por-síntese passa a ser o de encontrar o vetor-código ξ que minimize o erro quadrático no intervalo considerado:

$$\xi = \underset{i}{\operatorname{argmin}} \left\{ \sum_{n=0}^{L-1} \varepsilon_i^2[n] \right\}, \qquad i = 0, 1, \ldots, M-1 \tag{4.3}$$

onde L é o comprimento dos vetores-código.

Por requerer a minimização do erro quadrático entre o sinal sintetizado e o sinal original, a aplicação da análise-por-síntese envolve, a princípio, o cálculo do sinal sintetizado para cada vetor-código do dicionário. Desse fato fica claro que, apesar de apresentar bons resultados, a análise-por-síntese é por natureza um processo de alto custo computacional. Em decorrência disso, na prática são utilizados processos de busca sub-ótimos, ou seja, processos que não resultam necessariamente na minimização global do erro de quantização, mas que atingem níveis satisfatórios de desempenho quanto à qualidade da voz codificada.

4.3 Ponderação Espectral Perceptiva

No esquema apresentado na Figura 4.2 a busca de inovações procura simplesmente minimizar o erro quadrático entre a voz reconstruída e a voz original. Entretanto, como se trata da codificação de voz, o melhor resultado não é necessariamente obtido pela minimização da diferença matemática entre o sinal original e o reconstruído. A energia do erro de quantização não representa de maneira adequada a sua perceptibilidade pelo sistema auditivo humano, ou seja, mesmo possuindo energia maior, um erro de quantização pode ser menos perceptível que outro. Em vista disso também devem ser levadas em consideração as características espectrais de ambos, e não apenas a sua energia. Fica claro então que o vetor que deve ser procurado na análise-por-síntese é o que produz o som de melhor qualidade numa comparação com o sinal original, ou seja, aquele onde o erro de quantização foi minimizado no sentido de tornar-se o menos perceptível possível. Para atuar nesse sentido os codificadores CELP fazem uso das características de mascaramento do sistema auditivo, procurando concentrar o ruído de quantização da análise-por-síntese em regiões de freqüência onde ele se torna menos perceptível.

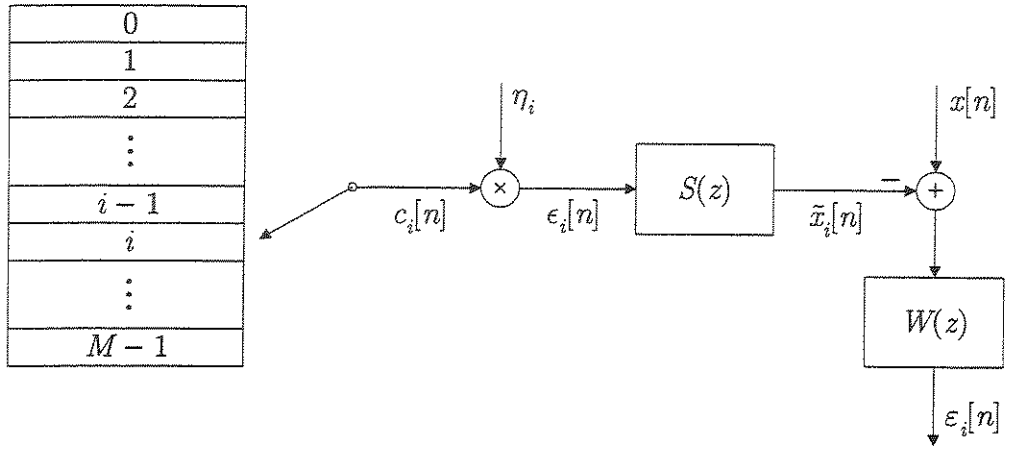


Figura 4.3: Influência do filtro de ponderação espectral perceptiva no processo de busca das inovações.

De acordo com as características de mascaramento em freqüência do sistema auditivo, um sinal concentrado em determinada freqüência mascara um ruído de energia menor que esteja na mesma faixa de freqüência [28]. A perceptibilidade do ruído aumenta com a sua amplitude e com a sua distância do sinal no campo freqüencial. Caso o ruído esteja concentrado em um faixa de freqüência distante da freqüência do sinal, ele será facilmente percebido ainda que possua energia bem menor que a do sinal. Desses fatos fica claro que seria interessante fazer com que o erro de quantização da análise-por-síntese estivesse concentrado nas regiões de freqüência onde o sinal possui maior energia, ou seja, nas regiões dos formantes. Nesse sentido, como ilustrado na Figura 4.3, é utilizado no sistema de análise-por-síntese um filtro $W(z)$ que procura ponderar o erro de quantização[1], causando atenuação nas regiões dos formantes e atribuindo maior peso as regiões de vales entre eles. O filtro $W(z)$ recebe o nome de filtro de ponderação espectral perceptiva.

Como ilustrado no esquema da Figura 4.3, o erro de quantização é dado pela diferença entre o sinal original e o sinal sintetizado filtrada por $W(z)$. Como se deve atenuar a região dos formantes, o filtro de ponderação deve possuir uma forma semelhante ao inverso do filtro de síntese $S(z)$, que descreve a envoltória espectral do sinal. A simples utilização de $1/S(z)$ como filtro de ponderação entretanto não é adequada, pois ocorrem atenuações demasiadas da regiões dos formantes, ou mesmo ganhos demasiados nas regiões dos vales. Tal efeito pode

[1]O símbolo ε_i, que aparece nas figuras 4.2 e 4.3, será usado para representar o erro de quantização, quer ele seja ponderado ou não.

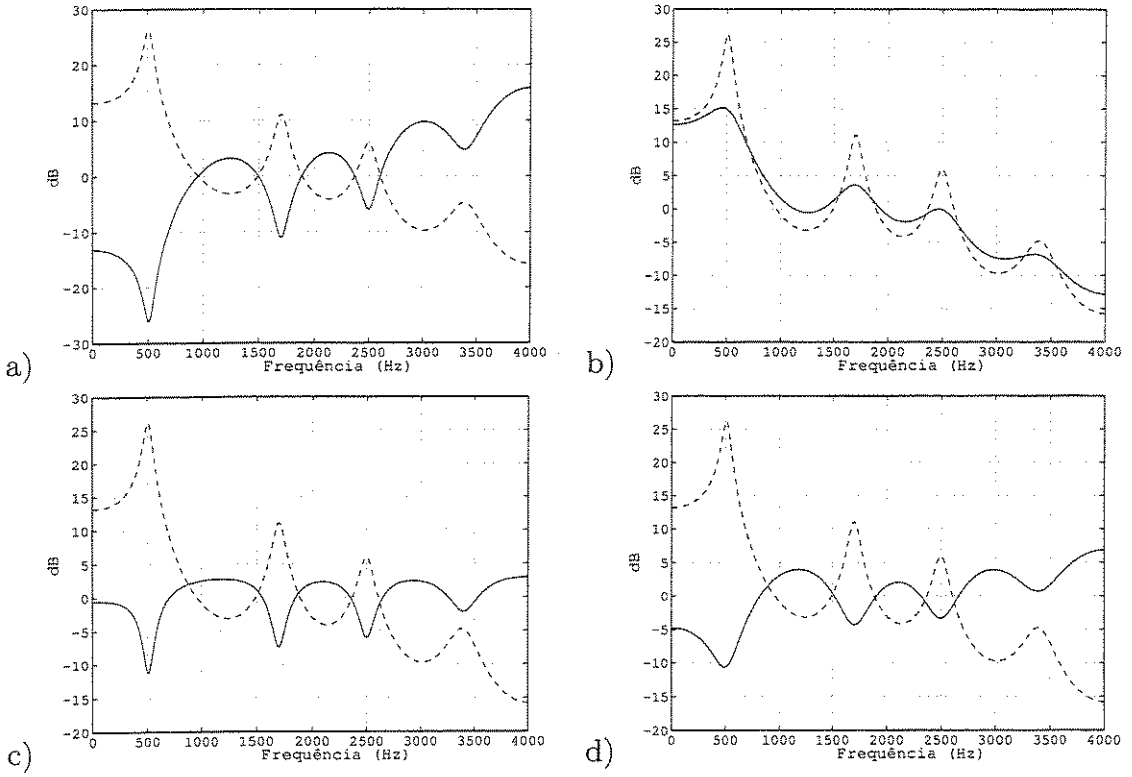


Figura 4.4: Funções de transferência relacionadas ao filtro de ponderação espectral perceptiva. (a) Filtro de síntese inverso $1/S(z)$ (b) Resposta em freqüência de $1/S(z/0.9)$. (c) Filtro de ponderação com $\gamma_1 = 1$ e $\gamma_2 = 0.9$. (d) Filtro de ponderação com $\gamma_1 = 0.5$ e $\gamma_2 = 0.95$. Em todos os ítens a curva tracejada representa o filtro de síntese LPC correspondente.

ser notado na Figura 4.4(a). Para contornar esse problema o filtro inverso é multiplicado pelo filtro de síntese com certa expansão de faixa:

$$W(z) = \frac{S(z/\gamma)}{S(z)}, \qquad \gamma \leq 1, \tag{4.4}$$

onde γ é o coeficiente de expansão de faixa.

A expansão de faixa é explicada pelo afastamento dos pólos em relação círculo unitário, na direção da origem. Os pólos z'_k de $S(z/\gamma)$ são dados por:

$$z'_k = \gamma z_k, \qquad k = 1, 2, \ldots, p \tag{4.5}$$

onde z_k são os pólos de $S(z)$.

A multiplicação dos pólos pelo fator γ faz com que o filtro $S(z/\gamma)$ apresente picos menos acentuados, e com largura maior, fato ilustrado na Figura 4.4(b). No filtro da ponderação representado pela equação (4.4), o fator γ controla a atenuação a ser dada na região dos formantes.

A Figura 4.4(c) mostra um filtro de ponderação com o fator $\gamma = 0.9$. Pode-se notar a atenuação nas proximidades dos formantes, que permite a alocação de mais ruído nessas regiões, onde a energia do sinal provocará o mascaramento por parte do sistema auditivo. Nas regiões dos vales, onde o sinal apresenta pequena energia, o filtro de ponderação penaliza o ruído aplicando um pequeno ganho ao erro de quantização.

Uma forma mais generalizada da equação (4.4) consiste em considerar a expansão de faixa também no filtro inverso:

$$W(z) = \frac{S(z/\gamma_2)}{S(z/\gamma_1)} \qquad 1 \leq \gamma_1 < \gamma_2 \tag{4.6}$$

Na última equação γ_1 e γ_2 são os coeficientes que controlam a expansão de faixa dos filtros correspondentes. A Figura 4.4(d) mostra um filtro de ponderação espectral com valores de γ_1 e γ_2 diferentes de 1.

4.3.1 Implementação Eficiente do Filtro de Ponderação

Com a adoção do filtro de ponderação espectral perceptiva a escolha do vetor-código que melhor representa o erro de predição passa a consistir da minimização da diferença entre o sinal original e o sinal reconstruído filtrada por $W(z)$. Trata-se então, a princípio, de

examinar o erro de quantização ponderado gerado por cada vetor-código e escolher aquele que possui menor energia.

Embora o diagrama da Figura 4.3 represente bem a idéia da ponderação perceptiva, não é esse o esquema utilizado na prática. O mesmo processo pode ser realizado de maneira mais eficiente, graças à propriedade distributiva dos sistemas lineares, através da aplicação do filtro de ponderação antes da subtração do sinal original pelo sinal sintetizado. Com isso os filtros $S(z)$ e $W(z)$ podem ser agrupados, como mostra a Figura 4.5, formando um novo filtro $H(z)$ que recebe o nome de filtro de síntese ponderado.

$$H(z) = W(z)S(z) \tag{4.7}$$

Com a ponderação espectral perceptiva, o erro de reconstrução passa a ser dado pela diferença entre o sinal original e o sinal reconstruído no domínio ponderado:

$$\varepsilon_i[n] = y[n] - \tilde{y}_i[n] \tag{4.8}$$

Analisando o esquema da Figura 4.5, pode-se notar facilmente que a filtragem do sinal $x[n]$ por $W(z)$ precisa ser feita uma única vez a cada processo de análise-por-síntese, já que o mesmo sinal $y[n]$ será usado para gerar o erro de quantização referente a todos os vetores-código. Dessa forma, com o re-posicionamento do filtro de ponderação, a geração do erro de quantização pode ser feita poupando-se uma filtragem, o que representa uma grande redução na complexidade quando se considera que o processo de análise-por-síntese exige, a princípio, o cálculo do erro de quantização para cada vetor do dicionário.

4.4 Considerações Sobre o Processamento Bloco-a-Bloco

Antes de prosseguir é importante fazer algumas observações relacionadas ao tratamento segmentado do sinal de voz nos processos de síntese e análise em codificadores CELP. Como apresentado previamente na Seção 3.2, a codificação baseada em predição linear é geralmente feita por blocos. Os codificadores CELP em particular consideram três segmentações do sinal de voz (Figura 3.4):

Os Blocos de Filtragem são trechos concatenados de cerca de 10 a $30ms$ de sinal de voz (80 a 240 amostras a $8kHz$). Os blocos de filtragem também são chamados simplesmente de blocos (*frames*), e a cada um deles é associado um conjunto de coeficientes

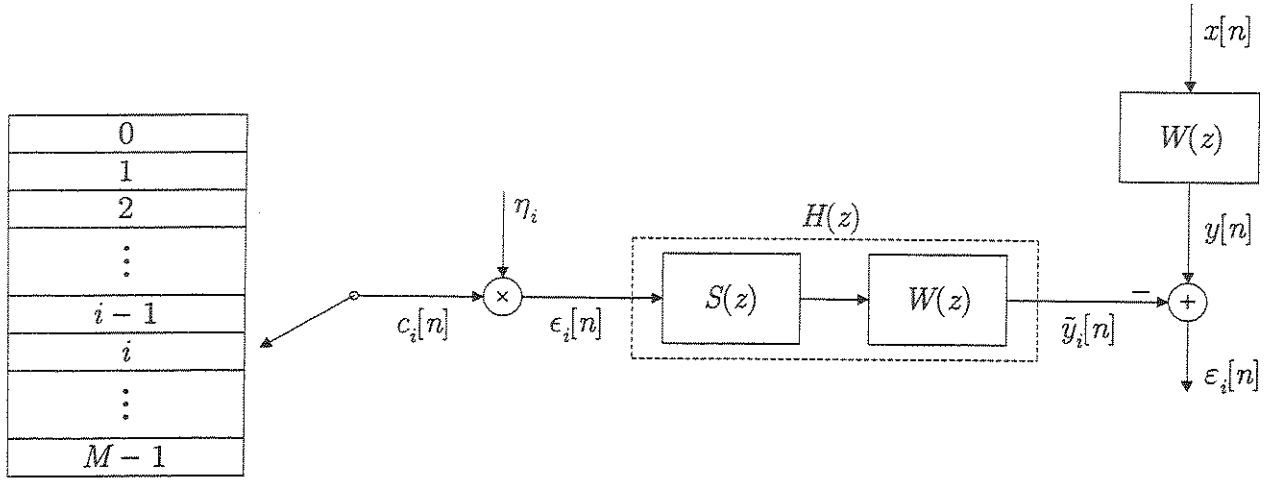


Figura 4.5: Esquema eficiente de busca através da análise-por-síntese. Diz-se que a subtração do sinal original pelo sinal sintetizado está sendo feita no domínio ponderado.

de predição linear. Na síntese, a transição de um bloco para outro corresponde a atualização dos coeficientes do filtro de síntese, ou seja, a troca dos coeficientes anteriores pelos coeficientes extraídos da análise preditiva do novo bloco.

As Janelas de Análise Preditiva correspondem ao trecho de sinal usado para o cálculo dos coeficientes de predição linear. Geralmente a janela de análise preditiva, ou janela de predição, é maior do que o bloco, e em muitos casos chega a possuir o dobro do seu tamanho. Na maioria dos casos utiliza-se janelas de hamming, hanning, ou derivadas delas.

Os Sub-Blocos de Excitação são sub-divisões dos blocos que possuem o mesmo tamanho dos vetores-código do dicionário. Geralmente cada bloco é dividido em 2 ou 4 sub-blocos. Enquanto que o cálculo dos coeficientes de predição é feito a cada bloco, a cada sub-bloco é feita a busca pelo melhor vetor-código e por seu ganho. Na síntese, a transição de um sub-bloco para outro corresponde ao início do uso de um novo ganho e de um novo vetor-código como excitação do filtro de síntese, que pode permanecer inalterado ou não, caso os sub-blocos pertençam ou não ao mesmo bloco[2]. Nessas transições o estado do filtro de síntese é mantido, o que ajuda a manter a suavidade do processo de filtragem.

[2]Vale ressaltar que em muitos codificadores os coeficientes do filtro de síntese não permanecem inalterados em sub-blocos de um mesmo bloco, sendo na verdade ajustados através de uma interpolação.

4.5 Filtro LTP

Conforme o apresentado até o momento a codificação CELP consiste na utilização de um dicionário de códigos para a representação do resíduo de predição de linear. A inovação é escolhida de forma com que o sinal reconstruído se aproxime o máximo possível do sinal original, em um esquema de análise-por-síntese onde o critério utilizado é o da minimização do erro quadrático de quantização ponderado por um filtro perceptual. Nesse processo é considerado que o sinal de voz original é branqueado por um filtro de análise preditiva e que depois poderá ser reconstruído pelo filtro de síntese. Entretanto, a análise preditiva adotada na maioria dos codificadores CELP é de curto prazo, o que quer dizer que o resíduo LPC a ser codificado ainda conterá correlação de longo prazo, conforme o apresentado na Seção 3.2.2. A correlação de longo prazo está presente especialmente nos trechos de som sonoro, decorrendo da vibração das cordas vocais.

Por conter um número reduzido de coeficientes, o que reduz a complexidade do processo e a quantidade de informação a ser codificada, a análise preditiva adotada pela maioria dos codificadores CELP é chamada de análise preditiva de curto prazo, ou STP (*Short-Term Prediction*).

Em geral os codificadores CELP utilizam filtros de análise STP com cerca de 10 a 12 coeficientes, onde é considerada um freqüência de amostragem de $8kHz$. O período de pitch, entretanto, varia de cerca de 2 a $20ms$ [28], o que corresponde a cerca de 16 a 160 amostras a $8kHz$. A solução adotada é a utilização de um filtro de análise preditiva de longo prazo (LTP - *Long-Term Prediction*) destinado exclusivamente à extração da correlação relativa à freqüência fundamental de vibração das cordas vocais. O filtro de análise LTP $P(z)$ pode ser dado de forma generalizada por:

$$P(z) = 1 - \sum_{k=-I}^{k=I} \mu_k z^{-D+k}, \tag{4.9}$$

onde os valores μ_k são os coeficientes do filtro, D é o período de pitch, e I define o tamanho intervalo onde será feita a predição.

Dessa forma, enquanto o filtro STP trata da predição baseada nos valores imediatamente anteriores ao valor a ser predito, a análise LTP trata da predição baseada em valores distantes do valor atual por um período de pitch. A utilização de um filtro LTP em conjunto com o filtro STP resulta num maior branqueamento do sinal analisado, como mostra Figura 4.6. Pode-se notar ainda na Figura 4.6 que o filtro LTP pouco ajuda no processamento

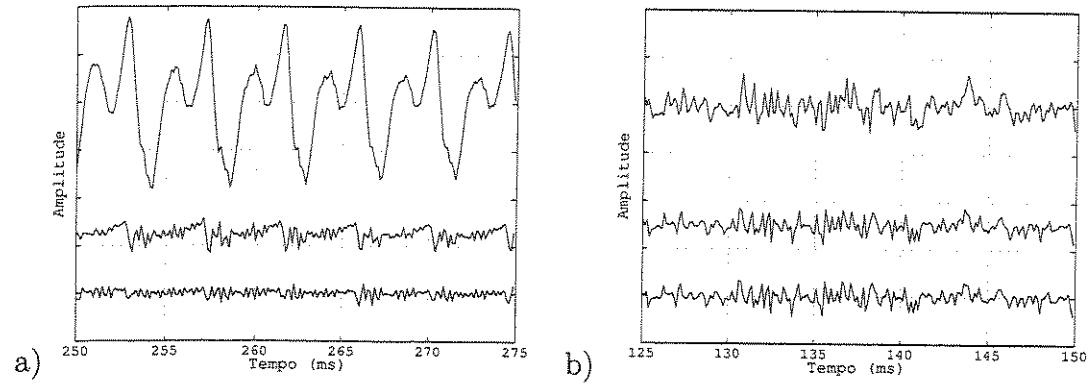


Figura 4.6: Resíduos de longo e curto prazo (a) de trecho de som sonoro, e (b) de um trecho de som surdo. Em ambos os casos a curva superior representa o sinal original. A curva do meio é o resíduo STP, e a curva inferior o resíduo após a filtragem LTP.

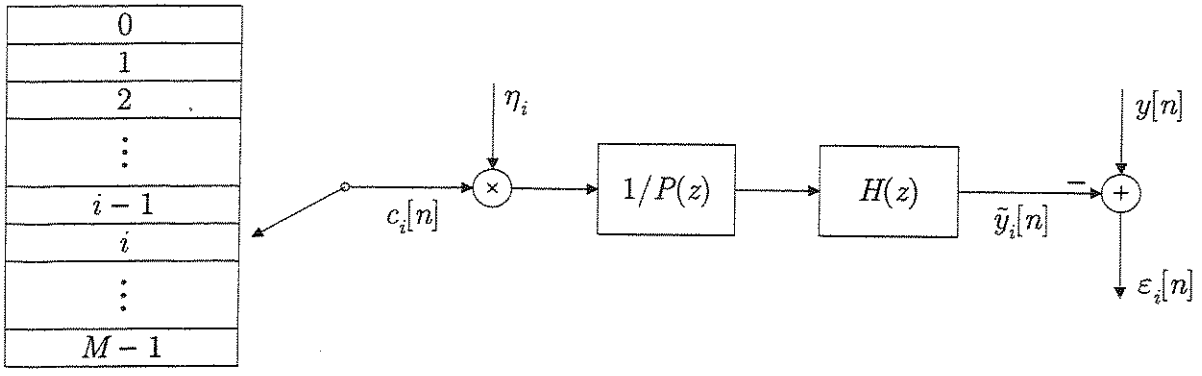


Figura 4.7: Geração do erro de quantização na análise-por-síntese.

de sinais surdos, que não se caracterizam pela presença da vibração da glote.

A introdução do filtro de síntese LTP $1/P(z)$ no processo de análise-por-síntese é apresentada na Figura 4.7. O filtro de síntese LTP é posicionado em série com o filtro de síntese STP ponderado, representado por $H(z)$.

Note que, de acordo com a equação (4.9), a utilização do filtro $P(z)$ exige o conhecimento do período de pitch D, além do cálculo dos coeficientes μ_k. Embora a equação (4.9) descreva um filtro de análise LTP generalizado, é bastante utilizado na prática um filtro bem mais simples, que possui apenas um coeficiente, ou seja, onde $I = 0$, o que resulta no seguinte filtro de síntese:

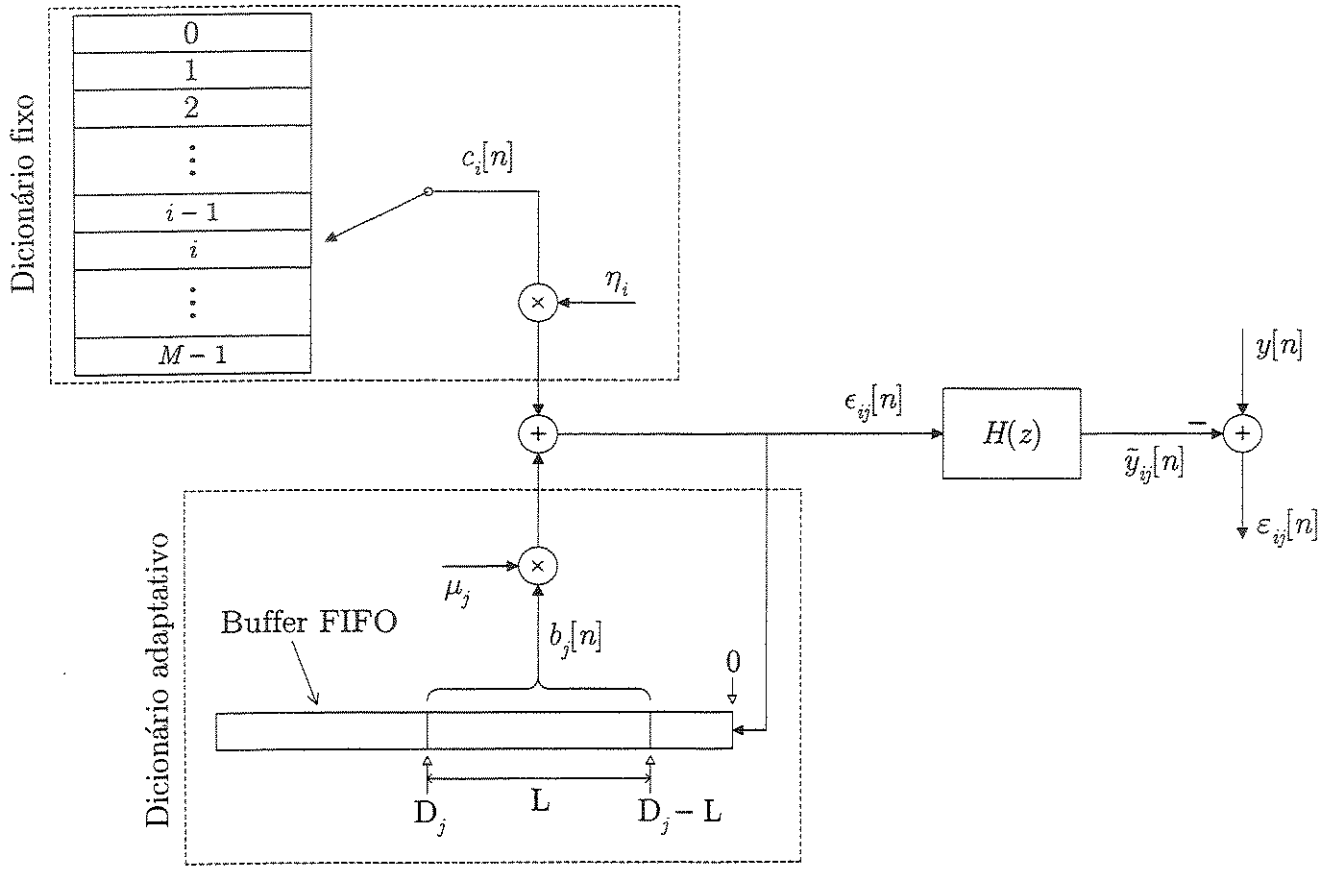


Figura 4.8: Participação do dicionário adaptativo na análise-por-síntese.

$$\frac{1}{P(z)} = \frac{1}{1 - \mu z^{-D}} \tag{4.10}$$

De acordo com a equação (4.10), o filtro de síntese LTP $1/P(z)$ atua apenas de forma a subtrair recursivamente do sinal a sua versão deslocada no tempo pelo período de pitch D e ponderada por μ. Tal processamento pode ser implementado através do armazenamento do sinal em um buffer FIFO[3] (*First-In-First-Out*); a filtragem LTP corresponderia então a subtração entre o sinal e o sinal atrasado armazenado no buffer, fato que está ilustrado na Figura 4.8.

Na Figura 4.8 é evidenciada ainda a consideração da estrutura de buffer FIFO da fil-

[3]O buffer FIFO em questão se caracteriza por efetuar, a cada sub-bloco de comprimento L, o esquecimento das L amostras mais antigas e a anexação das L amostras mais recentes de sinal.

tragem LTP como um *dicionário adaptativo*. De fato, a denominação do sistema em questão por dicionário adaptativo procede visto que o buffer FIFO é atualizado a cada sub-bloco, e que os valores do período de pitch e do ganho podem ser considerados respectivamente como a entrada do dicionário e seu ganho correspondente. Ainda na Figura 4.8, o índice j é utilizado para se referir aos vetores-código do dicionário adaptativo, que na verdade correspondem a trechos atrasados do sinal de excitação:

$$b_j[n] = \epsilon_{ij}[n - D_j] \tag{4.11}$$

A utilização do índice j caracteriza a situação em que não se conhece o período de pitch, devendo esse ser encontrado juntamente com a entrada do dicionário fixo. Em geral procura-se o valor ótimo do pitch entre um D_{min} menor que o comprimento do sub-bloco L e um D_{max} igual ao comprimento do buffer FIFO utilizado. Considerando um dicionário adaptativo com M_a vetores-código, ou *níveis*, vale a relação:

$$D_j = j + D_{min} \qquad 0 \leq j \leq M_a - 1 \tag{4.12}$$

Um caso comum, por exemplo, é a utilização de sub-blocos de tamanho 40 e a consideração dos limites $D_{min} = 20$ e $D_{max} = 143$, compondo um dicionário adaptativo de $M_a = 122$ níveis. Nesse caso, como a freqüência de amostragem é de $8kHz$, consegue-se cobrir a faixa de freqüências fundamentais de 60 a $400Hz$[4].

A entrada e o ganho do dicionário adaptativo podem ser determinados da mesma maneira que para o dicionário fixo através do esquema de análise-por-síntese, ou seja, devem ser escolhidos a cada sub-bloco a entrada do dicionário adaptativo e seu respectivo ganho que minimizem o erro entre o sinal de voz original e o sinal reconstruído. Dessa forma o processo completo de análise-por-síntese passa a ser composto composto pela busca em dois dicionários, sendo o erro ε_{ij} decorrente da utilização do vetor-código c_i do dicionário fixo e do vetor-código b_j do dicionário adaptativo.

Quando o período de pitch é determinado através da análise-por-síntese o processo é chamado de detecção de pitch em malha fechada (*closed-loop pitch detection*), em oposição à detecção de pitch em malha aberta (*open-loop pitch detection*), que corresponde a uma análise

[4]Vale ressaltar que para os casos onde o valor do pitch é menor que o comprimento do sub-bloco (D_j entre 20 e 40) faltam amostras do buffer FIFO para compor o vetor-código do dicionário adaptativo. Nesses casos o vetor-código é chamado de *incompleto*, sendo complementado através de repetições periódicas de si próprio [21].

mais simples não realizada através análise-por-síntese, mas sim de acordo com um algoritmo PDA similar aos descritos na Seção 3.2.6. Muitos codificadores CELP usam inicialmente um algoritmo de malha aberta apenas para obter um valor aproximado do pitch. Em seguida esse valor é utilizado como referência para simplificar o processo de busca em malha fechada.

4.6 Representação Vetorial das Variáveis na Análise-por-Síntese

Como a busca dos vetores-código através da análise-por-síntese é realizada a cada sub-bloco, convém representar vetorialmente as variáveis envolvidas no processo através do uso de vetores de comprimento igual ao do sub-bloco.

Dessa forma, considerando um dicionário com vetores-código de comprimento L, pode-se dizer que a excitação do filtro de síntese é dada por:

$$\boldsymbol{\epsilon}_{ij} = \eta_i \mathbf{c}_i + \mu_j \mathbf{b}_j, \tag{4.13}$$

onde os símbolos em negrito correspondem a vetores de comprimento L, e onde $\mathbf{c}_i$ e $\mathbf{b}_j$ correspondem respectivamente aos vetores-código dos dicionários fixo e adaptativo.

Por sua vez o erro de reconstrução da análise-por-síntese é dado por:

$$\boldsymbol{\varepsilon}_{ij} = \mathbf{y} - \tilde{\mathbf{y}}_{ij}, \tag{4.14}$$

onde os vetores $\tilde{\mathbf{y}}_{ij}$ e $\mathbf{y}$ correspondem respectivamente ao sinal reconstruído $\tilde{y}_{ij}$ e ao sinal original no domínio ponderado y, conforme o esquema da Figura 4.8.

É interessante ressaltar que a manutenção do estado do filtro de síntese ponderado nas transições de sub-blocos faz com que o vetor do sinal reconstruído $\tilde{\mathbf{y}}_{ij}$ possa ser decomposto na soma de dois outros vetores: a resposta do filtro $H(z)$ à excitação nula com o estado do filtro mantido, e a resposta do filtro $H(z)$ à excitação com o estado inicial nulo. Tal fato é representado pela seguinte equação:

$$\tilde{\mathbf{y}}_{ij} = \mathbf{y}_0 + \mathbf{H}\boldsymbol{\epsilon}_{ij}, \tag{4.15}$$

onde $\mathbf{y}_0$ é um vetor de comprimento L que corresponde à resposta de $H(z)$ à excitação nula.

Ainda na equação (4.15), $\mathbf{H}$ é a chamada *matriz de resposta impulsiva* do filtro $H(z)$,

uma matriz Toeplitz triangular inferior dada por:

$$\mathbf{H} = \begin{bmatrix} h[0] & 0 & \dots & 0 & 0 \\ h[1] & h[0] & \ddots & \vdots & 0 \\ \vdots & h[1] & \ddots & 0 & \vdots \\ h[L-2] & \vdots & \ddots & h[0] & 0 \\ h[L-1] & h[L-2] & \dots & h[1] & h[0] \end{bmatrix}, \tag{4.16}$$

onde $h[k]$, $k = 0, 1, \dots, L-1$, são os L primeiros elementos da resposta impulsiva do filtro de síntese ponderado $H(z)$.

Utilizando a equação (4.15) para substituir o valor de $\tilde{\mathbf{y}}_{ij}$ em (4.14), chega-se à seguinte representação para o erro de reconstrução:

$$\boldsymbol{\varepsilon}_{ij} = \mathbf{y} - \mathbf{y}_0 - \mathbf{H}\boldsymbol{\epsilon}_{ij} \tag{4.17}$$

Ou ainda,

$$\boldsymbol{\varepsilon}_{ij} = \mathbf{r} - \tilde{\mathbf{r}}_{ij}, \tag{4.18}$$

onde $\mathbf{r}$ é o chamado *vetor-alvo* da busca, definido como:

$$\mathbf{r} = \mathbf{y} - \mathbf{y}_0 \tag{4.19}$$

O vetor $\tilde{\mathbf{r}}_{ij}$ corresponde à reconstrução do vetor-alvo, e é definido como:

$$\tilde{\mathbf{r}}_{ij} = \mathbf{H}\boldsymbol{\epsilon}_{ij} \tag{4.20}$$

A equação (4.18) estabelece um novo esquema para a análise-por-síntese, onde a resposta do filtro $H(z)$ à entrada zero é subtraída do sinal original ponderado antes da busca, dando origem ao vetor-alvo $\mathbf{r}$. A busca é então feita no sentido de encontrar o vetor-código que minimize o erro entre o vetor-alvo $\mathbf{r}$ e sua reconstrução $\tilde{\mathbf{r}}_{ij}$. A Figura 4.9 mostra o esquema em questão. O filtro $H_D(z)$ corresponde ao filtro de síntese ponderado $H(z)$ com o estado inicial nulo, enquanto que $H_C(z)$ corresponde ao mesmo filtro, porém com o estado preservado do bloco anterior.

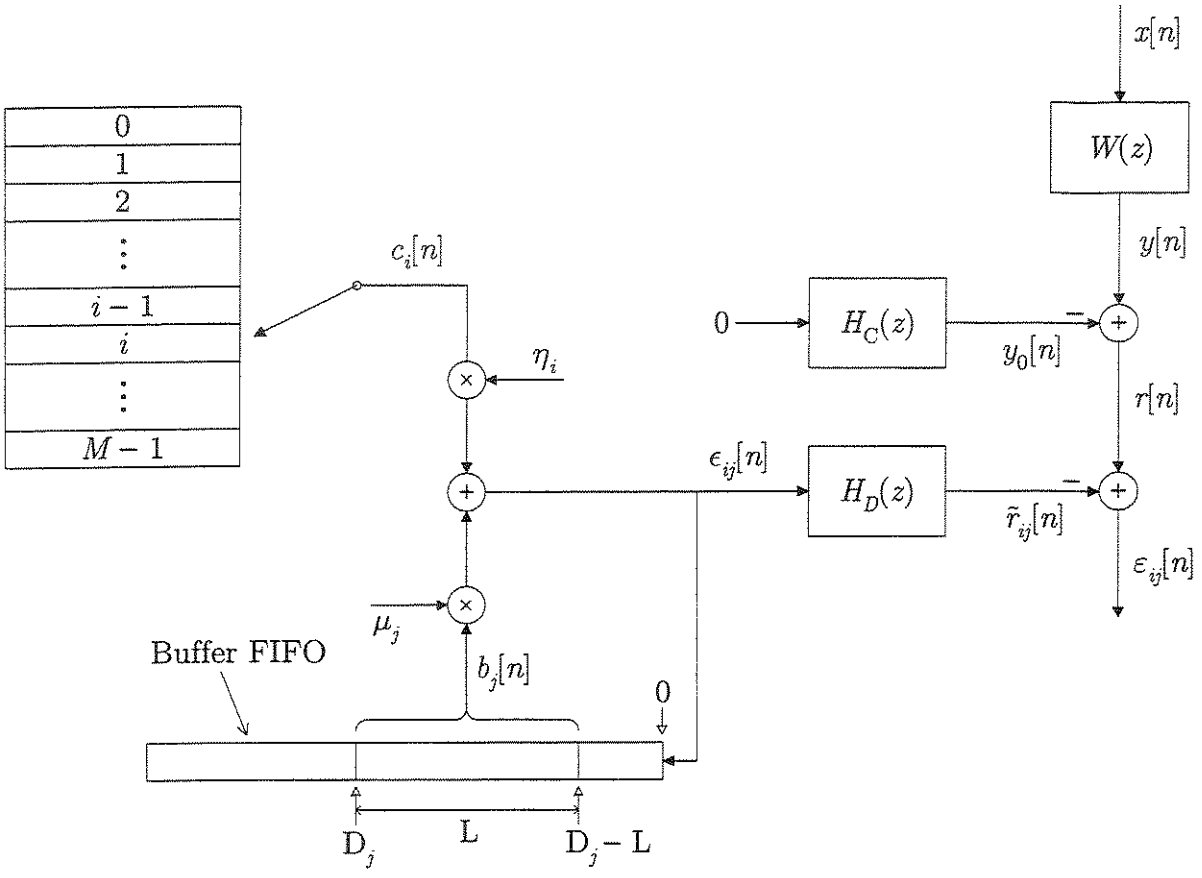


Figura 4.9: Subtração da resposta relativa ao estado inicial do filtro de síntese ponderado.

4.7 Busca Seqüencial nos Dicionários Adaptativo e Fixo

O processo ótimo de busca em dois dicionários consiste na verificação de cada uma das combinações possíveis entre as entradas dos dois dicionários. Dessa forma, considerando que os dicionários fixo e adaptativo constam respectivamente de M e M_a vetores-código cada, a busca ótima exige a análise de M vezes M_a possibilidades. Em geral, para simplificar o processo de busca, os codificadores CELP utilizam um esquema de busca seqüencial nos dicionários, onde inicialmente é feita a busca em um dos dicionários considerando-se a influência do outro como nula, em seguida é feita a busca no segundo dicionário após a subtração da contribuição do primeiro. A busca seqüencial nos dicionários reduz o número de verificações para $M + M_a$, e não causa grande impacto na qualidade da voz codificada.

A Figura 4.10 expõe um esquema de busca seqüencial nos dicionários adaptativo e fixo. Inicialmente é feita a busca no dicionário adaptativo, que trata da determinação do período de pitch. O vetor-alvo da busca no dicionário adaptativo, é dado pela equação (4.19), reproduzida aqui por conveniência:

$$\mathbf{r} = \mathbf{y} - \mathbf{y}_0$$

A reconstrução do vetor-alvo da busca adaptativa, por sua vez, é dada por:

$$\tilde{\mathbf{r}}_j = \mu_j \mathbf{H}\mathbf{b}_j \tag{4.21}$$

O *vetor-alvo da busca fixa* $\mathbf{u}$ é então definido como o resíduo da primeira busca, ou seja:

$$\mathbf{u} = \mathbf{r} - \tilde{\mathbf{r}}_j \tag{4.22}$$

E a reconstrução do vetor-alvo da busca fixa é dada por:

$$\tilde{\mathbf{u}}_i = \eta_i \mathbf{H}\mathbf{c}_i \tag{4.23}$$

O erro de reconstrução final é então dado pela diferença entre $\mathbf{u}$ e $\tilde{\mathbf{u}}_i$:

$$\boldsymbol{\varepsilon}_i = \mathbf{u} - \tilde{\mathbf{u}}_i \tag{4.24}$$

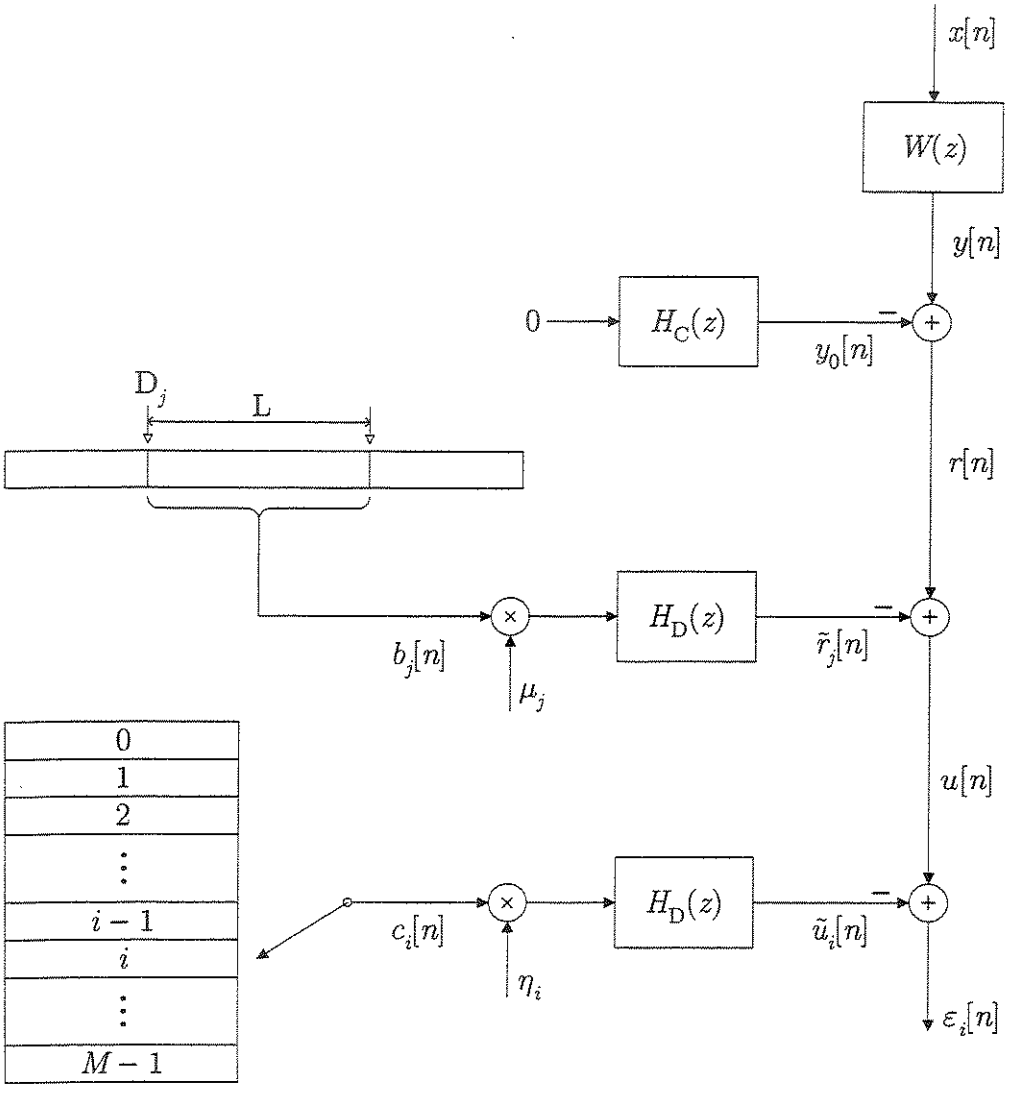


Figura 4.10: Esquema completo de busca seqüencial nos dicionários adaptativo e fixo em codificadores CELP.

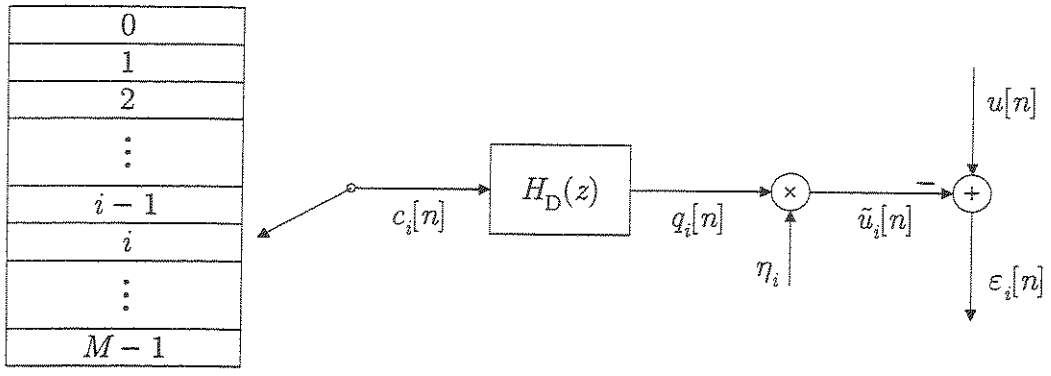


Figura 4.11: Modelo de busca no dicionário fixo em codificadores CELP.

4.8 Processo Básico de Busca no Dicionário Fixo

Como apresentado no item 4.2, o processo de busca no dicionário consiste na procura pelo vetor-código que minimize o erro quadrático de reconstrução. É importante ressaltar que a descrição a seguir será voltada para a busca no dicionário fixo, entretanto, os resultados apresentados também se aplicam de maneira análoga à busca no dicionário adaptativo.

Para melhor compreensão será considerado a seguir o esquema apresentado na Figura 4.11, onde a aplicação do ganho η_i é feita após a filtragem do vetor-código pelo filtro de síntese ponderado $H(z)$ com estados iniciais nulos. Tal reposicionamento explicita o sinal $q_i[n]$, chamado de vetor-código filtrado.

4.8.1 Determinação do Ganho

Considerando uma representação vetorial das variáveis envolvidas na análise-por-síntese, como a apresentada no item 4.6, a minimização do erro quadrático total no sub-bloco é dada por:

$$\min_i \{\|\boldsymbol{\varepsilon}_i\|^2\} \tag{4.25}$$

Utilizando a equação (4.24) chega-se a:

$$\|\boldsymbol{\varepsilon}_i\|^2 = \boldsymbol{\varepsilon}_i^T \boldsymbol{\varepsilon}_i = \mathbf{u}^T\mathbf{u} - 2\tilde{\mathbf{u}}_i^T\mathbf{u} + \tilde{\mathbf{u}}_i^T\tilde{\mathbf{u}}_i \tag{4.26}$$

De acordo com a Figura 4.11, o vetor-alvo reconstruído é dado pela multiplicação do

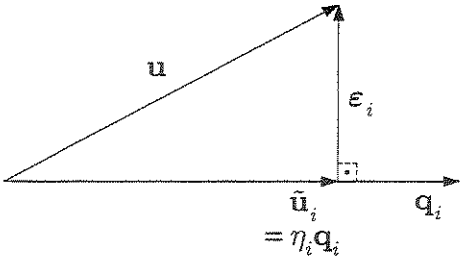


Figura 4.12: Relação entre o vetor-alvo $\mathbf{u}$, o vetor-código filtrado $\mathbf{q}_i$ e o vetor-alvo reconstruído $\tilde{\mathbf{u}}_i$. O fator de ganho deve ser escolhido de forma a fazer com que o erro de reconstrução seja perpendicular ao vetor-código filtrado.

ganho pelo vetor-código filtrado:

$$\tilde{\mathbf{u}}_i = \eta_i \mathbf{q}_i \tag{4.27}$$

Portanto, pode-se expressar o erro quadrático por:

$$\boldsymbol{\varepsilon}_i^T \boldsymbol{\varepsilon}_i = \mathbf{u}^T \mathbf{u} - 2\eta_i \mathbf{q}_i^T \mathbf{u} + \eta_i^2 \mathbf{q}_i^T \mathbf{q}_i \tag{4.28}$$

O ganho η_i que minimiza o erro quadrático pode então ser obtido derivando a equação anterior e igualando a zero:

$$\frac{\partial(\boldsymbol{\varepsilon}_i^T \boldsymbol{\varepsilon}_i)}{\partial(\eta_i)} = -2\mathbf{q}_i^T \mathbf{u} + 2\eta_i \mathbf{q}_i^T \mathbf{q}_i = 0 \tag{4.29}$$

$$\Rightarrow \eta_i = \frac{\mathbf{q}_i^T \mathbf{u}}{\mathbf{q}_i^T \mathbf{q}_i} \tag{4.30}$$

Para cada vetor código $\mathbf{c}_i$ existe então um fator de ganho ideal que minimiza o erro quadrático no sub-bloco. A escolha do fator de ganho pode ser melhor compreendida através de uma análise gráfica do problema considerado. A Figura 4.12 mostra a relação entre o vetor-alvo $\mathbf{u}$, o vetor-código filtrado $\mathbf{q}_i$, e o vetor-alvo reconstruído $\tilde{\mathbf{u}}_i$. A escolha do fator de ganho a ser aplicado a $\mathbf{q}_i$ procura minimizar o vetor do erro de reconstrução $\boldsymbol{\varepsilon}_i$; portanto, o fator de ganho deve ser escolhido de forma que o vetor-alvo reconstruído $\tilde{\mathbf{u}}_i$ seja a projeção do vetor-alvo $\mathbf{u}$ em $\mathbf{q}_i$. Qualquer outro fator de ganho resultará num erro quadrático maior.

4.8.2 Redefinição do Problema da Análise-por-Síntese

Aplicando o valor obtido para η_i de (4.30) na equação (4.28) pode-se obter:

$$\boldsymbol{\varepsilon}_i^T \boldsymbol{\varepsilon}_i = \mathbf{u}^T \mathbf{u} - 2\frac{\mathbf{q}_i^T \mathbf{u}}{\mathbf{q}_i^T \mathbf{q}_i} \mathbf{q}_i^T \mathbf{u} + \left(\frac{\mathbf{q}_i^T \mathbf{u}}{\mathbf{q}_i^T \mathbf{q}_i} \right)^2 \mathbf{q}_i^T \mathbf{q}_i \tag{4.31}$$

$$\boldsymbol{\varepsilon}_i^T \boldsymbol{\varepsilon}_i = \mathbf{u}^T \mathbf{u} - \frac{\left(\mathbf{q}_i^T \mathbf{u}\right)^2}{\mathbf{q}_i^T \mathbf{q}_i} \tag{4.32}$$

A última equação permite que o critério de busca seja redefinido. Como o vetor-alvo u é o mesmo para todos os vetores-código, a minimização do erro quadrático total do sub-bloco corresponde à maximização da fração à direita da equação (4.32), ou seja,

$$\min_i \left\{ \|\boldsymbol{\varepsilon}_i\|^2 \right\} \equiv \max_i \left\{ \frac{\left(\mathbf{q}_i^T \mathbf{u}\right)^2}{\mathbf{q}_i^T \mathbf{q}_i} \right\} \tag{4.33}$$

Pode-se dizer então que o processo de análise-por-síntese tem por objetivo encontrar o vetor-código que maximize o termo τ_i, onde τ_i é definido como

$$\tau_i = \frac{\left(\mathbf{q}_i^T \mathbf{u}\right)^2}{\mathbf{q}_i^T \mathbf{q}_i}, \tag{4.34}$$

Além disso, outro resultado interessante pode ser obtido a partir de (4.32), e com a utilização da equação (4.30):

$$\boldsymbol{\varepsilon}_i^T \boldsymbol{\varepsilon}_i = \mathbf{u}^T \mathbf{u} - \eta_i^2 \mathbf{q}_i^T \mathbf{q}_i \tag{4.35}$$

E, usando a equação (4.27), pode-se chegar finalmente a:

$$\boldsymbol{\varepsilon}_i^T \boldsymbol{\varepsilon}_i = \mathbf{u}^T \mathbf{u} - \tilde{\mathbf{u}}_i^T \tilde{\mathbf{u}}_i \tag{4.36}$$

A equação (4.36) corresponde ao Teorema de Pitágoras, comprovando que o vetor erro deve mesmo ser perpendicular ao vetor-alvo reconstruído, conforme o representado na Figura 4.12.

Da equação (4.36) fica claro ainda que o processo de busca também pode ser considerado como a maximização do comprimento do vetor-alvo reconstruído $\tilde{\mathbf{u}}_i$.

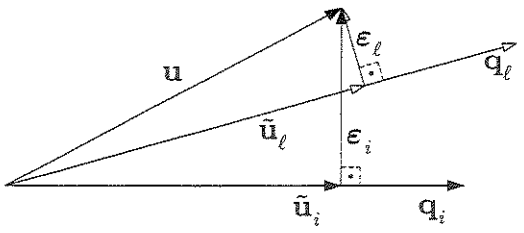


Figura 4.13: Influência da direção do vetor-código filtrado na minimização do erro de reconstrução.

$$\max_i \left\{ \tilde{\mathbf{u}}_i^T \tilde{\mathbf{u}}_i \right\} \tag{4.37}$$

A relação entre a direção do vetor-código filtrado e a minimização do erro de reconstrução está ilustrada na Figura 4.13. No caso ideal o vetor-código filtrado $\mathbf{q}_i$ possui a mesma direção do vetor-alvo $\mathbf{u}$, ocorrendo nesse caso a igualdade $\mathbf{u}^T\mathbf{u} = \tilde{\mathbf{u}}_i^T\tilde{\mathbf{u}}_i$.

4.8.3 Esquema Eficiente de Busca no Dicionário

A redefinição do problema da análise-por-síntese, de acordo com (4.33), é útil para o estabelecimento de um esquema eficiente de busca das inovações.

Para analisar a maximização de τ_i, é interessante definir uma variável C_i, dada por:

$$C_i = \mathbf{q}_i^T \mathbf{u} \tag{4.38}$$

O vetor-código filtrado pode ser representado através da multiplicação do vetor-código pela matriz de resposta impulsiva:

$$\mathbf{q}_i = \mathbf{H}\mathbf{c}_i \tag{4.39}$$

Portanto,

$$C_i = (\mathbf{H}\mathbf{c}_i)^T \mathbf{u} \tag{4.40}$$

$$C_i = \mathbf{u}^T \mathbf{H}\mathbf{c}_i \tag{4.41}$$

Como o termo $\mathbf{u}^T\mathbf{H}$ é o mesmo para todos os vetores-código, C_i pode ser representado

por:

$$C_i = \mathbf{d}^T \mathbf{c}_i, \tag{4.42}$$

onde o vetor $\mathbf{d}$ é o vetor-alvo filtrado regressivamente (*backward-filtered target vector*), e representa a correlação entre o vetor-alvo e a resposta impulsiva do filtro de síntese ponderado, sendo definido como:

$$\mathbf{d} = \mathbf{H}^T \mathbf{u} \tag{4.43}$$

Além da definição de C_i é interessante também representar por $E_{\mathbf{q}_i}$ a energia do vetor-código filtrado, ou seja,

$$E_{\mathbf{q}_i} = \mathbf{q}_i^T \mathbf{q}_i \tag{4.44}$$

De (4.39) vem que:

$$E_{\mathbf{q}_i} = \mathbf{c}_i^T \left(\mathbf{H}^T \mathbf{H}\right) \mathbf{c}_i \tag{4.45}$$

Ou seja,

$$E_{\mathbf{q}_i} = \mathbf{c}_i^T \mathbf{\Phi} \mathbf{c}_i, \tag{4.46}$$

onde a matriz $\mathbf{\Phi}$ é chamada de *matriz de correlação impulsiva* de $H(z)$, sendo definida da seguinte forma:

$$\mathbf{\Phi} = \mathbf{H}^T \mathbf{H} \tag{4.47}$$

Desse modo, τ_i pode ser escrito como:

$$\tau_i = \frac{C_i^2}{E_{\mathbf{q}_i}} = \frac{\left(\mathbf{d}^T \mathbf{c}_i\right)^2}{\mathbf{c}_i^T \mathbf{\Phi} \mathbf{c}_i} \tag{4.48}$$

Como a matriz $\mathbf{\Phi}$ e o vetor $\mathbf{d}$ são os mesmos para todos os vetores-código, a equação (4.48) mostra que o processo de busca da análise-por-síntese pode ser realizado de maneira mais eficiente através do cálculo prévio desses elementos comuns a todo o dicionário. Dessa forma, o processo de busca pode ser esquematizado de maneira simplificada nas seguintes etapas:

1. Cálculo do vetor $\mathbf{d}$;

2. Cálculo da matriz de correlação impulsiva Φ;

3. Cálculo do valor τ_i para cada vetor-código do dicionário;

4. Escolha do vetor-código que produz o maior τ_i.

Capítulo 5

ALGORITMOS DE BUSCA EM DICIONÁRIOS ALGÉBRICOS

Desde o momento em que foi proposto pela primeira vez, em 1985 por Schroeder e Atal, o esquema de codificação por excitação de códigos passou a representar a possibilidade de transmissão de voz de qualidade a taxas de cerca de 4 a 12kbps [35]. O bom desempenho apresentado por codificadores CELP, entretanto, tem como custo a necessidade de um grande esforço computacional para realizar a compressão da voz, o que dificulta a implementação de tais codificadores em tempo-real. Deve-se ressaltar que a complexidade da codificação por excitação de códigos é inerente à idéia da análise-por-síntese, que supõe que sejam verificados todos os vetores-código do dicionário a cada processo de busca.

Uma das maneiras de reduzir a complexidade do processo de busca é a adoção de dicionários esparsos, o que simplifica os cálculos necessários para a verificação de cada vetor-código. Um tipo de dicionário esparso amplamente utilizado é o chamado dicionário de multi-pulsos algébricos (*Multi-Pulse Algebraic Codebook*), que apresenta bons resultados no que diz respeito tanto à redução do custo computacional do processo de busca quanto à preservação da qualidade da voz codificada. Codificadores CELP que fazem uso de tais dicionários são denominados de codificadores ACELP (*Algebraic Code-Excited Linear Prediction*).

A adoção de algoritmos sub-ótimos é outra maneira de reduzir a complexidade do procedimento de busca. Tais algoritmos consideram simplificações no processo de busca, o que faz basicamente com que o vetor-código ideal da análise-por-síntese não seja mais necessariamente o escolhido. A conseqüência disso é que os algoritmos sub-ótimos apresentam, em relação ao processo de busca ótimo, certa degradação da qualidade da voz codificada,

o que representa o custo da redução de complexidade alcançada. O uso de algoritmos sub-ótimos, entretanto, é necessário visto que usando-se codificadores ACELP a implementação em tempo-real de um algoritmo de busca ótimo seria impraticável em processadores digitais de sinais atuais.

5.1 Dicionários de Multi-Pulsos Algébricos

Os dicionários algébricos são utilizados como meio de reduzir a complexidade do processo de busca. Em tais dicionários as inovações são vetores esparsos, ou seja, elas contém apenas um pequeno número de valores diferentes de zero, que são chamados de *pulsos*. Tal característica, como será visto mais adiante, produz uma grande simplificação dos cálculos necessários na análise-por-síntese.

5.1.1 Sinais de Excitação Esparsos

É conveniente representar um sinal de excitação esparso $\epsilon[n]$ por:

$$\epsilon[n] = \eta \sum_{k=0}^{N-1} \alpha_k \delta(n - m_k), \tag{5.1}$$

onde η é um fator de ganho, e N corresponde ao número de pulsos, que possuem amplitudes e posições dadas respectivamente por α_k e m_k.

Considerando que o vetor-código possui comprimento L, devem ainda ser obedecidas as seguintes relações:

$$0 \leq n \leq L-1 \qquad 0 \leq m_k \leq L-1 \qquad N << L \tag{5.2}$$

A relação mais à direita, que diz que o número de pulsos deve ser bem menor que o comprimento do vetor-código, deve ser observada para a caracterização da esparsidade do sinal de excitação.

A estrutura de um dicionário esparso é definida então pelas regras, ou restrições, feitas tanto às amplitudes α_k como às posições dos pulsos m_k. A chamada excitação de pulsos regulares (RPE - *Regular Pulse Excitation*), por exemplo, considera que os pulsos constituem uma seqüência com espaçamento constante, ou seja:

$$m_k = kS + \varphi, \qquad 0 \leq \varphi \leq S-1 \tag{5.3}$$

onde S é uma constante que representa o espaçamento entre os pulsos, e φ é um valor que determina o deslocamento da seqüência de pulsos em relação ao início do vetor-código, razão pela qual φ pode ser interpretado como a *fase* da seqüência.

No caso dos pulsos regulares a determinação da posição dos pulsos se resume principalmente à determinação de φ, a fase da seqüência de pulsos. Além disso, entretanto, o algoritmo de busca deve ainda determinar as amplitudes dos pulsos, que em geral podem assumir apenas os valores $+1$ ou -1. Tradicionalmente o espaçamento adotado é 3 ou 4, e o número de pulsos é de cerca de 10 a 13 a cada $5ms$ (que corresponde a 40 amostras a $8kHz$).

5.1.2 Multi-Pulsos Algébricos

Nos dicionários de multi-pulsos algébricos, ou simplesmente dicionários algébricos, cada pulso pode possuir as amplitudes ± 1, e considera-se uma divisão das posições do vetor-código em um certo número de trilhas intercaladas, que possuem cada qual uma fase associada.

Para uma certa fase φ, o conjunto de posições:

$$\{\varphi,\ S+\varphi,\ 2S+\varphi,\ \ldots,\ (L_\varphi - 1)S + \varphi\}, \qquad 0 \leq \varphi \leq S-1 \tag{5.4}$$

é chamado de grade ou trilha de posições da fase φ. O valor S representa o espaçamento das trilhas e L_φ o número de posições da fase φ.

Os vetores-código dos dicionários algébricos possuem um número fixo de pulsos, e geralmente aloca-se um único pulso para cada trilha dos vetores-código algébricos, técnica chamada de ISPP (*Interleaved Single Pulse Permutation*). Entretanto, como será visto mais adiante, em alguns dicionários existem trilhas com dois ou mais pulsos, e em outros casos ocorre o inverso, ou seja, o pulso pode ser alocado em mais de uma trilha.

De uma maneira geral as posições possíveis para os pulsos nos dicionários algébricos podem ser dadas por:

$$m_k = j_k S + \varphi_k, \qquad 0 \leq \varphi_k \leq S-1 \tag{5.5}$$

onde φ_k representa a fase do pulso, e j_k corresponde ao seu deslocamento dentro da grade de posições da fase.

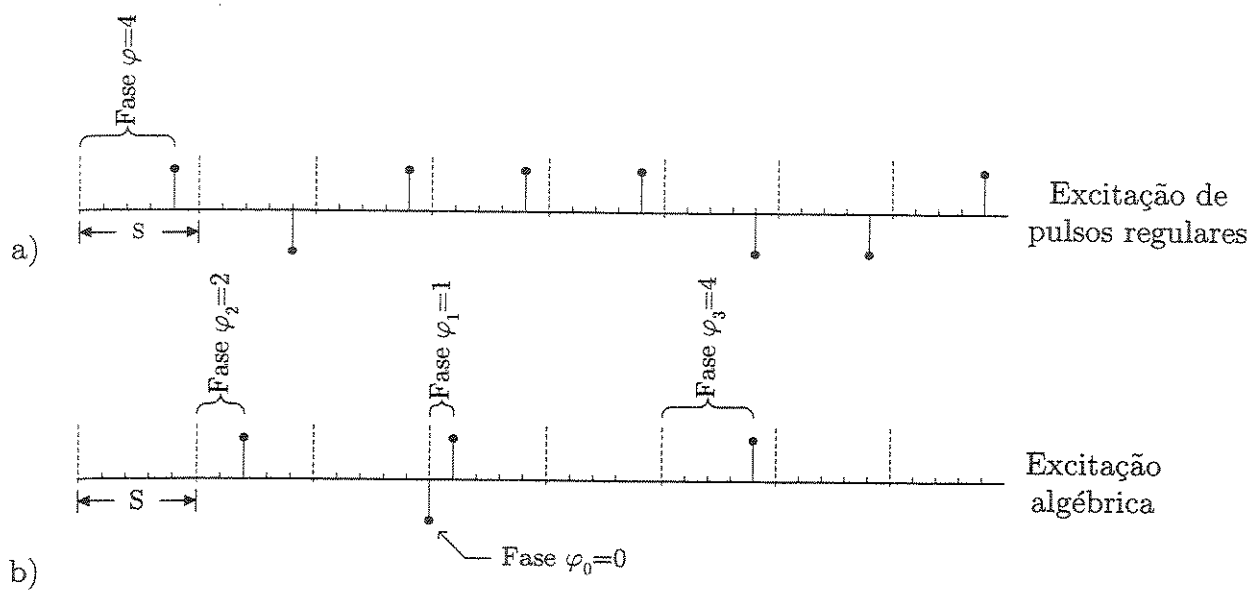


Figura 5.1: Vetores de excitação esparsos. (a) Esboço de um vetor de excitação por pulsos regulares. (b) Esboço de um vetor de excitação algébrica referente ao dicionário da Tabela 5.2.

O vetor-código completo é representado pelo conjunto de pulsos $\{i_0, i_1, \ldots, i_{N-1}\}$, sendo cada pulso é dado por:

$$i_k = \alpha_k \delta(n - m_k), \qquad 0 \leq k \leq N-1 \tag{5.6}$$

onde no contexto da estrutura dos dicionários algébricos o índice k é chamado de identificador do pulso (pulse ID). Dessa forma, o pulso com amplitude α_k e posição m_k é chamado de pulso k.

Comparando as equações (5.1) e (5.5) pode-se notar que o dicionário algébrico assemelha-se, de certa forma, ao sistema de excitação por pulsos regulares. A Figura 5.1 mostra uma comparação entre ambos. A diferença básica é que na excitação algébrica não existem pulsos em todos os espaçamentos, e eles não possuem todos a mesma fase.

Outra característica dos dicionários algébricos é a sua simetria. Caso $\mathbf{c}_i$ seja um vetor-código de um dicionário algébrico, o vetor $-\mathbf{c}_i$ também pertencerá a esse dicionário. Tal fato é facilmente comprovado quando se verifica que as amplitudes dos pulsos algébricos podem assumir os valores ± 1 sem restrição quanto à posição.

Quanto às possíveis posições para os pulsos pode-se dizer que existem basicamente

três tipos de dicionários algébricos:

1. Aqueles onde existe exatamente um pulso por trilha, chamados de completos;

2. Aqueles onde existem menos pulsos do que trilhas, havendo pelo menos um pulso com possibilidade de escolha de fase;

3. Aqueles onde existem mais pulsos do que trilhas, havendo pelo menos uma trilha onde será alocado mais de um pulso.

A Tabela 5.1 mostra um exemplo de dicionário algébrico do primeiro tipo, onde existe exatamente um pulso por trilha. Nesse caso em particular os vetores de inovação têm comprimento $L = 40$, e possuem 4 pulsos cada. A grade de posições foi dividida em 4 trilhas, resultando em 10 posições por trilha.

É interessante notar que faz parte da estrutura do dicionário algébrico a associação prévia de cada pulso à (pelo menos) uma trilha. Nesse caso, em particular, o pulso i_0 é alocado para a trilha 0, o pulso i_1 para a trilha 1, e assim por diante. Dessa forma tem-se que $\varphi_0 = 0$, $\varphi_1 = 1$, $\varphi_2 = 2$, e $\varphi_3 = 3$.

Tabela 5.1: Possíveis posições dos pulsos em um dicionário algébrico de 4 trilhas com 1 pulso por trilha.

Fases	Pulsos	Posições									
0	i_0	0	4	8	12	16	20	24	28	32	36
1	i_1	1	5	9	13	17	21	25	29	33	37
2	i_2	2	6	10	14	18	22	26	30	34	38
3	i_3	3	7	11	15	19	23	27	31	35	39

Um outro exemplo, onde existem menos pulsos do que trilhas, é dado pela Tabela 5.2, que mostra um dicionário algébrico típico utilizado em vários padrões de codificação de voz, como por exemplo no G.729 da ITU-T e no IS-641 da TIA/EIA. Nesse caso a grade de posições foi formada com 5 trilhas, resultando em 8 posições por trilha. O número de pulsos por inovação é apenas 4, de forma que o quarto pulso poderá pertencer tanto à trilha 3 como à trilha 4, sendo essa escolha feita a cada sub-bloco.

O número de vetores-código desse dicionário pode ser calculado da seguinte maneira:

$$M = L_0 L_1 L_2 (L_3 + L_4) 2^N = 8 \cdot 8 \cdot 8 \cdot 16 \cdot 2^4 = 131072 \tag{5.7}$$

onde a potência de dois representa as possíveis combinações de amplitudes, e os demais termos as combinações de posições.

Tabela 5.2: Possíveis posições dos pulsos no dicionário algébrico adotado pelo padrão G.729 da ITU-T.

Fases	Pulsos	Posições							
0	i_0	0	5	10	15	20	25	30	35
1	i_1	1	6	11	16	21	26	31	36
2	i_2	2	7	12	17	22	27	32	37
3	i_3	3	8	13	18	23	28	33	38
4		4	9	14	19	24	29	34	39

A Tabela 5.3 mostra a estrutura do dicionário algébrico utilizado no padrão GSM-AMR (*Adaptive Multi-Rate*) na taxa 10.2kbps. Nesse caso os vetores-código, que possuem tamanho 40, são divididos em 4 trilhas, com 2 pulsos pulsos por trilha. É importante ressaltar que nesse caso em especial os pulsos com a mesma fase podem inclusive assumir a mesma posição, resultando em um pulso de amplitude $+2$ ou -2.

Tabela 5.3: Possíveis posições dos pulsos no dicionário algébrico adotado pelo padrão GSM-AMR na taxa de transmissão de 10.2kbps.

Fases	Pulsos	Posições									
0	i_0, i_4	0	4	8	12	16	20	24	28	32	36
1	i_1, i_5	1	5	9	13	17	21	25	29	33	37
2	i_2, i_6	2	6	10	14	18	22	26	30	34	38
3	i_3, i_7	3	7	11	15	19	23	27	31	35	39

Além dos três tipos de dicionários algébricos expostos, alguns codificadores ACELP adotam ainda dicionários onde é considerada a existência de duas grades distintas, uma par e outra ímpar. Nesse caso as posições dos pulsos são dadas por:

$$m_k = j_k \cdot S + \varphi_k + \varphi_g, \tag{5.8}$$

onde $\varphi_g = 0$ ou $\varphi_g = 1$ indica respectivamente a seleção da grade par ou da grade ímpar. A Tabela 5.4 mostra a grade de posições pares de um codificador ACELP com sub-bloco de comprimento 60 e 4 pulsos em cada vetor de inovação. Os números entre parênteses não são válidos, estando presentes apenas para completar a tabela.

Tabela 5.4: Exemplo de grade par de um dicionário algébrico com 8 fases de excitação.

Pulsos	Posições							
i_0	0	8	16	24	32	40	48	56
i_1	2	10	18	26	34	42	50	58
i_2	4	12	20	28	36	44	52	(60)
i_3	6	14	22	30	38	46	54	(62)

Nas seções a seguir será dada maior atenção aos dicionários algébricos com apenas uma grade. Entretanto, o tratamento do casos onde é feita a divisão do conjunto de posições nas grades par e ímpar é geralmente análogo.

5.2 O Problema da Busca em Dicionários Algébricos

O processo de busca no dicionário, como apresentado na Seção 4.8, corresponde matematicamente a encontrar a inovação $\mathbf{c}_i$ que maximiza o valor τ_i dado por:

$$\tau_i = \frac{C_i^2}{E_{\mathbf{q}_i}}, \tag{5.9}$$

onde $C_i = \mathbf{d}^T\mathbf{c}_i$ e $E_{\mathbf{q}_i} = \mathbf{c}_i^T\mathbf{\Phi}\mathbf{c}_i$.

Devido à esparsidade dos vetores-código do dicionário de multi-pulsos algébricos, o cálculo de C_i e de $E_{\mathbf{q}_i}$ é bastante facilitado, como será visto nesta seção.

Os vetores-código $\mathbf{c}_i$ dos dicionários algébricos podem ser representados por:

$$c_i[n] = \sum_{k=0}^{N-1} \alpha_k \delta(n - m_k), \tag{5.10}$$

onde N é o número de pulsos, e m_k e α_k são respectivamente as suas posições e amplitudes.

Dessa forma, o valor de C_i resume-se a:

$$C_i = \sum_{k=0}^{N-1} \alpha_k d[m_k] \tag{5.11}$$

Já a energia do vetor código filtrado $E_{\mathbf{q}_i}$ pode ser dada por:

$$E_{\mathbf{q}_i} = \sum_{k=0}^{N-1} \alpha_k^2 \phi[m_k, m_k] + \sum_{k=0}^{N-1} \sum_{k \neq \ell = 0}^{N-1} \alpha_k \alpha_\ell \phi[m_k, m_\ell], \tag{5.12}$$

onde os valores $\phi[m_k, m_\ell]$ são elementos da matriz $\mathbf{\Phi}$.

Como a matriz de correlação impulsiva $\mathbf{\Phi}$ é simétrica, e como as amplitudes dos pulsos podem assumir apenas os valores ± 1, resulta que:

$$E_{\mathbf{q}_i} = \sum_{k=0}^{N-1} \phi[m_k, m_k] + 2 \sum_{k=1}^{N-1} \sum_{\ell=0}^{k-1} \alpha_k \alpha_\ell \phi[m_k, m_\ell] \tag{5.13}$$

Nesse ponto é conveniente re-escrever as equações (5.11) e (5.13) da seguinte maneira:

$$C_i = \alpha_0 d[m_0] + \alpha_1 d[m_1] + \alpha_2 d[m_2] + \cdots \tag{5.14}$$

$$\begin{aligned} E_{\mathbf{q}_i} &= \phi[m_0, m_0] \\ &+ \phi[m_1, m_1] + 2\alpha_0 \alpha_1 \phi[m_0, m_1] \\ &+ \phi[m_2, m_2] + 2\alpha_0 \alpha_2 \phi[m_0, m_2] + 2\alpha_1 \alpha_2 \phi[m_1, m_2] \\ &+ \cdots \end{aligned} \tag{5.15}$$

As equações acima indicam que o cálculo de C_i e $E_{\mathbf{q}_i}$ pode ser feito por etapas, de maneira acumulativa. Cada etapa corresponderia à uma trilha, e em cada etapa seriam somadas as contribuições da trilha aos valores de C_i e $E_{\mathbf{q}_i}$. O procedimento pode ficar mais evidente com as considerações a seguir.

Seja $C_i^{(j)}$ o valor dado por:

$$C_i^{(j)} = \sum_{k=0}^{j} \alpha_k d[m_k] \tag{5.16}$$

O valor $C_i^{(j)}$ pode ser considerado como o valor acumulado na etapa j do cálculo de C_i, onde após N etapas, $j = 0, 1, \ldots, N-1$, é atingido o valor final de C_i:

```
/* Trilha 0 */
For (m0 = 0, m0 < L, m0 += STEP) {
    C0 = α0*d(m0);
    E0 = φ(m0,m0);
    /* Trilha 1 */
    For (m1 = 1, m1 < L, m1 += STEP) {
        C1 = C0 + α1*d(m1);
        E1 = E0 + φ(m1,m1) + 2*α0*α1*φ(m0,m1);
        /* Trilha 2 */
        For (m2 = 2, m2 < L, m2 += STEP) {
            C2 = C1 + α2*d(m2);
            E2 = E1 + φ(m2,m2) + 2*α0*α2*φ(m0,m2) + 2*α1*α2*φ(m1,m2);
            /* Trilha 3 */
            For (m3 = 3, m3 < L, m3 += STEP) {
                :
            }
        }
    }
}
```

Figura 5.2: Pseudo-código referente ao cálculo de C_i e $E_{\mathbf{q}_i}$ através de uma seqüência de laços encaixados. As variáveis Cn e En são utilizadas respectivamente para o cálculo por etapas dos valores de C_i e $E_{\mathbf{q}_i}$.

$$C_i = C_i^{(N-1)} = \sum_{k=0}^{N-1} \alpha_k d[m_k] \tag{5.17}$$

Da mesma forma o valor parcial da etapa j do cálculo de $E_{\mathbf{q}_i}$ pode ser dado por:

$$E_{\mathbf{q}_i}^{(j)} = \sum_{k=0}^{j} \phi[m_k, m_k] + 2 \sum_{k=1}^{j-1} \sum_{\ell=0}^{k-1} \alpha_k \alpha_\ell \phi[m_k, m_\ell], \tag{5.18}$$

sendo o valor de $E_{\mathbf{q}_i}$ atingido após as N etapas:

$$E_{\mathbf{q}_i} = E_{\mathbf{q}_i}^{(N-1)} \tag{5.19}$$

Tanto para $C_i^{(j)}$ quanto para $E_{\mathbf{q}_i}^{(j)}$ pode ser obtida uma forma recursiva:

$$\begin{cases} C_i^{(j)} = C_i^{(j-1)} + \alpha_j d[m_j] \\ E_{\mathbf{q}_i}^{(j)} = E_{\mathbf{q}_i}^{(j-1)} + \phi[m_j, m_j] + 2\sum_{k=0}^{j-1} \alpha_k \alpha_j \phi[m_k, m_j] \end{cases} \tag{5.20}$$

Da análise das recursões em (5.20), pode-se notar que o cálculo de C_i e $E_{\mathbf{q}_i}$ pode ser feito de maneira eficiente através de um esquema de "laços encaixados" (*nested loops*), onde cada laço corresponde a uma trilha, e nele são somadas as contribuições de cada posição da trilha. A eficiência desse sistema vem do fato de que dessa forma evita-se a repetição desnecessária do cálculo das parcelas que compõe os valores C_i e $E_{\mathbf{q}_i}$.

Dois vetores-código com os j primeiros pulsos em comum, possuirão os mesmos valores parciais $E_{\mathbf{q}_i}^{(j-1)}$ e $C_i^{(j-1)}$. O uso do esquema de laços encaixados evita justamente que esses termos sejam calculados repetidamente. A Figura 5.2 mostra o pseudo-código referente à implementação do esquema de laços encaixados para o cálculo eficiente de C_i e $E_{\mathbf{q}_i}$. No pseudo-código apresentado a maioria das variáveis são semelhantes as utilizadas nas equações até aqui, com exceção de `STEP`, que corresponde ao espaçamento S, e de `Cj` e `Ej`, que correspondem respectivamente à $C_i^{(j)}$ e $E_{\mathbf{q}_i}^{(j)}$. Como se pode notar na Figura 5.2, os resultados parciais obtidos são constantemente reutilizados, fato que justifica a redução da complexidade computacional alcançada por esse método.

5.2.1 Cálculo da Matriz de Correlação Impulsiva

Nesse ponto é importante fazer uma consideração sobre o cálculo da matriz de correlação impulsiva $\mathbf{\Phi}$, que é feito antes dos processos de busca.

A partir das equações (4.47) e (4.16), pode-se notar que os elementos da matriz $\mathbf{\Phi}$ são dados por:

$$\phi[m,n] = \phi[n,m] = \sum_{k=0}^{L-1-m} h[k]h[k+m-n] \qquad m > n \tag{5.21}$$

Dessa forma, os elementos da mesma diagonal podem ser calculados da seguinte maneira:

$$\phi[m-1,n-1] = h[L-m]h[L-n] + \phi[m,n] \tag{5.22}$$

com as seguintes condições iniciais:

$$\phi[L-1,n] = h[0]h[L-1-n] \qquad n = 0,1,\ldots,L-1 \tag{5.23}$$

A equação (5.22) mostra que a matriz $\mathbf{\Phi}$ pode ser gerada de maneira eficiente num esquema onde os elementos das diagonais são calculados recursivamente. Esse esquema não

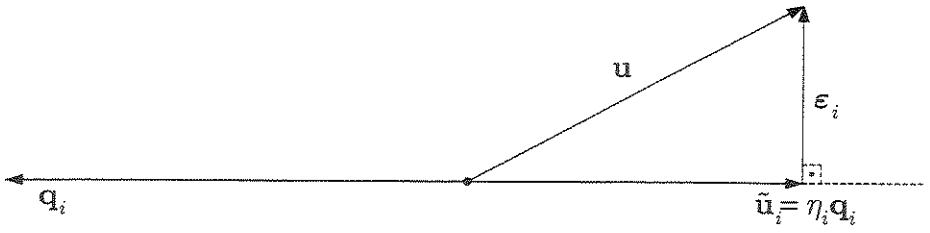


Figura 5.3: Exemplo de situação onde o ganho é negativo.

é válido somente para os codificadores algébricos, mas sim para qualquer codificador CELP em geral.

5.2.2 Determinação do Ganho em Codificadores ACELP

Embora a determinação do ganho não faça parte exatamente do processo de busca no dicionário devem ser feitas algumas observações pertinentes sobre a relação entre o ganho e os vetores-código dos dicionários algébricos.

Na determinação da excitação em codificadores excitados por código o ganho é encontrado após o processo de busca, em função do vetor-código encontrado. De acordo com o exposto na Seção 4.8.1, dado um vetor-código qualquer $\mathbf{c}_i$, o ganho que minimiza o erro quadrático de reconstrução é dado por:

$$\eta_i = \frac{\mathbf{q}_i^T \mathbf{u}}{\mathbf{q}_i^T \mathbf{q}_i} \tag{5.24}$$

Com algumas substituições simples pode-se chegar ainda a:

$$\eta_i = \frac{\mathbf{d}^T \mathbf{c}_i}{\mathbf{c}_i^T \mathbf{\Phi} \mathbf{c}_i} = \frac{C_i}{E_{\mathbf{q}_i}} \tag{5.25}$$

Como mostra a Figura 4.12 no processo de busca é de interesse especial a direção do vetor-código filtrado $\mathbf{q}_i$, já que procura-se justamente minimizar o ângulo entre as direções do vetor-código filtrado $\mathbf{q}_i$ e do vetor-alvo $\mathbf{u}$.

Deve-se ressaltar, entretanto, que o valor do ganho η_i, como dado por (5.25), não é positivo para todos os vetores-código do dicionário. Valores negativos podem ocorrer e correspondem a casos onde o vetor-código filtrado $\mathbf{q}_i$ faz ângulo maior do que 90° com o vetor-alvo, indicando a necessidade de inversão no sentido do vetor-código, fato que está ilustrado na Figura 5.3

É interessante notar que em dicionários simétricos existem matematicamente duas soluções ótimas para o problema da busca no dicionário, dadas pelos pares $(\mathbf{c}_\xi, \eta_\xi)$ e $(-\mathbf{c}_\xi, -\eta_\xi)$. Isso acontece porque um mesmo vetor-alvo reconstruído pode ser composto de duas maneiras:

$$\tilde{\mathbf{u}}_i = \eta_i \mathbf{H}\mathbf{c}_i = (-\eta_i)\mathbf{H}(-\mathbf{c}_i) \tag{5.26}$$

Ou seja, do ponto de vista da reconstrução do sinal alvo, o vetor-código $-\mathbf{c}_i$ produz matematicamente o mesmo resultado que $\mathbf{c}_i$, desde que também se utilize um ganho inverso. Além disso, tanto o vetor-código como seu oposto produzem o mesmo valor τ_i:

$$\tau_i = \frac{(\mathbf{d}^T\mathbf{c}_i)^2}{\mathbf{c}_i^T\mathbf{\Phi}\mathbf{c}_i} = \frac{(\mathbf{d}^T(-\mathbf{c}_i))^2}{(-\mathbf{c}_i^T)\mathbf{\Phi}(-\mathbf{c}_i)} \tag{5.27}$$

Na prática, evidentemente, são de interesse apenas os casos onde o ganho é positivo. *É conveniente então definir como vetor-código ótimo aquele que maximiza o valor de τ_i e cujo ganho é positivo.* Da equação (5.25) pode-se notar que a determinação do sinal do ganho pode ser feita através do sinal de C_i. O ganho é positivo para inovações que produzam uma correlação C_i positiva.

5.3 Busca Ótima

O algoritmo ótimo de busca é aquele que encontra com certeza o vetor-código ótimo. A determinação da solução ótima para o problema da busca exige, a princípio, que sejam verificados cada um dos vetores-código do dicionário considerado, ou seja, que sejam verificadas todas as possíveis combinações tanto de amplitudes quanto de posições para os pulsos. Como apresentado na Seção 5.2 tal tarefa pode ser realizada mais eficientemente com o uso de um esquema de laços encaixados. No caso da busca ótima, em particular, basta que ao final do cálculo de cada par de valores C_i e $E_{\mathbf{q}_i}$ seja verificado se eles compõem um τ_i maior do que o armazenado até o momento.

Como exemplo, a Figura 5.4 mostra um pseudo-código referente a uma implementação da busca ótima no dicionário algébrico da Tabela 5.2. O trecho de código é semelhante ao apresentado na Figura 5.2, com a diferença de que nesse caso a estrutura dos laços foi adequada à grade do dicionário considerado. Pode-se notar, por exemplo, que de acordo com a estrutura do dicionário o pulso i_3 pode estar nas trilhas 3 ou 4, existindo portando no pseudo-código apresentado dois laços para o pulso i_3, que se referem justamente às fases 3 e

```
For (m0 = 0, m0 < 40, m0 += STEP) {
    C0 = α0*d(m0);
    E0 = φ(m0,m0);
    For (m1 = 1, m1 < 40, m1 += STEP) {
        C1 = C0 + α1*d(m1);
        E1 = E0 + φ(m1,m1) + 2*α0*α1*φ(m0,m1);
        For (m2 = 2, m2 < 40, m2 += STEP) {
            C2 = C1 + α2*d(m2);
            E2 = E1 + φ(m2,m2) + 2*α0*α2*φ(m0,m2) + 2*α0*α2*φ(m0,m2);
            For (m3 = 3, m3 < 40, m3 += STEP) {
                C3 = C2 + α3*d(m3);
                E3 = E2 + φ(m3,m3) + 2*α0*α3*φ(m0,m3) + 2*α1*α3*φ(m1,m3) + 2*α2*α3*φ(m2,m3);
                VerificaMaximo(C3, E3, m0, m1, m2, m3, α0, α1, α2, α3);
            }
            For (m3 = 4, m3 < 40, m3 += STEP) {
                C3 = C2 + α3*d(m3);
                E3 = E2 + φ(m3,m3) + 2*α0*α3*φ(m0,m3) + 2*α1*α3*φ(m1,m3) + 2*α2*α3*φ(m2,m3);
                VerificaMaximo(C3, E3, m0, m1, m2, m3, α0, α1, α2, α3);
            }
        }
    }
}
```

Figura 5.4: Pseudo-código que representa uma implementação do algoritmo de busca ótimo no dicionário algébrico da Tabela 5.2. O trecho de código mostrado deve ser executado para cada combinação de amplitudes $(\alpha_0, \alpha_1, \alpha_2, \alpha_3)$.

4.

Na a Figura 5.4 a função `VericaMaximo` tem como papel simplesmente armazenar as posições e amplitudes onde ocorreu o máximo de τ_i. Isso pode ser feito facilmente comparando o valor de um máximo temporário contra o valor atual de $C_i^2/E_{\mathbf{q}_i}$, que caso seja maior, deverá ser salvo como novo máximo temporário. Também devem ser salvas evidentemente as posições e amplitudes dos pulsos onde ocorreu o máximo. Na Figura 5.5 está um esquema de uma possível função de cálculo do máximo.

É importante lembrar que os laços no trecho de código da Figura 5.4 se referem simplesmente ao cálculo das posições. Para que a busca ótima seja completa o algoritmo apresentado deve ainda ser repetido para as combinações de amplitudes dos pulsos. Como são considerados 4 pulsos por inovação e existem duas possibilidades de amplitude para cada pulso, há 16 possíveis combinações de amplitudes. Entretanto, como o dicionário é simétrico basta que o algoritmo da Figura 5.4 seja executado para uma metade do conjunto

```
VerificaMaximo(C, E, m0, m1, m2, m3, α0, α1, α2, α3)
{
    Csq = C * C;
    if (Csq * Emax > Csqmax * E) {
        Csqmax = Csq;
        Emax = E;
        m0max = m0; m1max = m1; m2max = m2; m3max = m3;
        α0max = α0; α1max = α1; α2max = α2; α3max = α3;
    }
}
```

Figura 5.5: Exemplo de função para a verificação do valor máximo de τ_i. Ao ser chamada após o cálculo de C_i e $E_{\mathbf{q}_i}$ para cada vetor-código, as variáveis com o índice `max` armazenarão dados sobre a inovação ótima. Para o correto funcionamento todas as variáveis possuidoras do índice `max` devem ser de um tipo em que seus valores não são esquecidos quando a função é executada uma outra vez.

de combinações de amplitudes, ou seja, 8 vezes, havendo depois da busca uma verificação do sinal do valor de C_i que resultou no τ_i máximo. O algoritmo pode ser executado, por exemplo, apenas para os casos onde a amplitude do primeiro pulso é positiva. Em seguida, caso o sinal da correlação C_i seja negativo, as amplitudes corretas serão o inverso das encontradas.

Apesar de determinar a solução ideal para o problema da análise-por-síntese, o algoritmo ótimo apresenta como ponto negativo uma complexidade computacional exageradamente grande. Embora o cálculo de C_i e $E_{\mathbf{q}_i}$ possa ser feito de maneira eficiente através do esquema de laços encaixados, a utilização de um algoritmo de busca ótimo em codificadores práticos está completamente fora de questão, não só pela complexidade desse algoritmo, mas também pelo fato do ganho em qualidade de voz em relação a outros algoritmos bem mais simples não ser plenamente significante.

5.4 Pré-Associação de Amplitudes às Posições

A pré-associação de amplitudes às posições [1] constitui uma simplificação no processo de busca no dicionário que elimina a necessidade de repetir a busca das posições para cada combinação possível de amplitudes. O processo consiste na associação prévia de uma amplitude para cada posição dos pulsos no vetor-código. Uma vez encontradas as posições ideais dos pulsos, suas amplitudes são dadas pela relação "posição↔amplitude" estabelecida previamente.

Em geral a correspondência entre posições e amplitudes é feita associando à cada posição n do vetor-código uma amplitude A_n dada por:

$$A_n = Q\left(b[n]\right), \qquad n = 0, 1, \ldots, L-1 \tag{5.28}$$

onde $Q(.)$ é uma função de quantização, e os valores $b[n]$ são os elementos do vetor $\mathbf{b}$, chamado de vetor de estimação de amplitudes[1] (*amplitude estimate vector*).

No caso dos dicionários algébricos, onde existem apenas duas amplitudes possíveis, $+1$ ou -1, a determinação das amplitudes se resume à determinação dos sinais (ou polarizações) dos pulsos. Nesse caso é conveniente utilizar a função de quantização

$$Q(v) = \text{sign}(v), \tag{5.29}$$

onde

$$\text{sign}(v) = \begin{cases} +1, \text{ para } v \geq 0 \\ -1, \text{ para } v < 1 \end{cases} \tag{5.30}$$

Dessa maneira resulta que:

$$A_n = \text{sign}(b[n]) \tag{5.31}$$

Encontradas as posições, as amplitudes do vetor-código são dadas por:

$$\alpha_k = A_{m_k}, \qquad k = 0, 1, \ldots, N-1 \tag{5.32}$$

onde N é o número de pulsos e m_k suas posições, de acordo com (5.10).

Para estimar as amplitudes dos pulsos existem duas abordagens possíveis, uma baseada na excitação ideal e outra no vetor-alvo filtrado regressivamente $\mathbf{d}$.

Na primeira abordagem, considera-se que as amplitudes devem ser estimadas de acordo com a excitação ideal. A idéia baseia-se no fato de que, quanto maior a liberdade referente à posições e amplitudes para os pulsos, mais a excitação escolhida deve se aproximar da excitação ideal. No caso da busca no dicionário fixo a excitação ideal corresponde

[1]Embora a notação $\mathbf{b}_j$ tenha sido utilizada anteriormente para fazer referência ao j-ésimo vetor-código do dicionário adaptativo, o vetor de estimação de amplitudes pode ser chamado de $\mathbf{b}$ sem nenhum conflito, dada presença do índice no primeiro caso e dado que o dicionário adaptativo não é tratado nem neste capítulo nem nos subseqüentes.

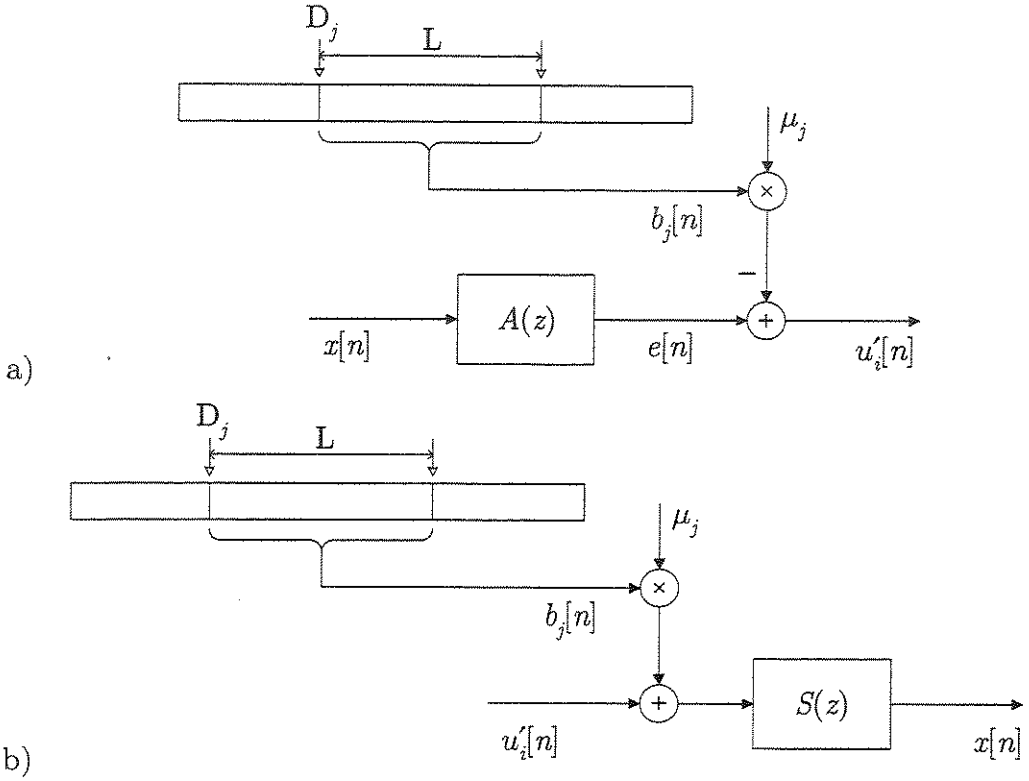


Figura 5.6: (a) Geração do erro de predição de longo prazo em codificadores CELP. $A(z)$ é o filtro de análise de curto prazo, $A(z) = 1/S(z)$. (b) O erro de predição de longo prazo no papel de excitação ideal da busca fixa, atuando na reconstrução do sinal.

ao resíduo de longo prazo $u'[n]$, resultante da subtração entre o resíduo de curto prazo e a contribuição do dicionário adaptativo. A Figura 5.6 mostra um diagrama onde o resíduo LTP $u'[n]$ é gerado a partir do sinal original $x[n]$. É importante enfatizar a diferença entre o resíduo de longo prazo $u'[n]$ e o sinal-alvo da busca fixa $u[n]$. A Figura 4.10 mostra que embora o vetor $u[n]$ também seja proveniente de uma subtração da contribuição do dicionário adaptativo, ele não se constitui como um resíduo, mas sim como um sinal a ser sintetizado. Além disso $u[n]$ sofre o efeito do filtro de ponderação, ao contrário de $u'[n]$.

Em uma outra abordagem, considera-se que uma boa alternativa para o vetor de estimação de amplitudes $\mathbf{b}$ é a utilização do vetor-alvo filtrado regressivamente $\mathbf{d}$, ou seja:

$$b[n] = d[n] \tag{5.33}$$

A utilização da equação (5.33) tem como base o fato de maximizar o valor de C_i para um dado conjunto de posições:

$$C_i = \sum_{k=0}^{N-1} A_{m_k} d[m_k] \tag{5.34}$$

$$= \sum_{k=0}^{N-1} \text{sign}(d[m_k]) d[m_k] \tag{5.35}$$

$$= \sum_{k=0}^{N-1} |d[m_k]| \tag{5.36}$$

Evidentemente a maximização de C_i representa uma simplificação no processo de busca. Ou seja, embora o objetivo da busca seja a maximização do termo $C_i^2/E_{\mathbf{q}_i}$ com qualquer combinação de amplitudes, são adotas a-priori amplitudes que maximizam C_i para um dado conjunto de posições.

A importância do vetor $\mathbf{d}$ na estimação das amplitudes cresce quando se considera dicionários de maior esparsidade. Pode-se notar, por exemplo, que para o caso limite de vetores-código com apenas um pulso, a determinação dos sinais seria feita de maneira ideal através de (5.33).

Considerando as duas abordagens apresentadas, define-se geralmente o vetor de estimação de amplitudes como uma soma ponderada entre o vetor-alvo filtrado regressivamente e o resíduo de predição de longo prazo, ambos normalizados [1]:

$$\mathbf{b} = (1-\lambda)\frac{\mathbf{d}}{\|\mathbf{d}\|} + \lambda\frac{\mathbf{u}'}{\|\mathbf{u}'\|}, \qquad 0 \leq \lambda \leq 1 \tag{5.37}$$

onde o fator λ é uma constante que determina o peso a ser dado aos vetores $\mathbf{u}'$ e $\mathbf{d}$ na estimação de amplitudes. A escolha do fator λ está relacionada com a esparsidade do dicionário, de forma que quanto maior a esparsidade menor deve ser o valor de λ.

Um valor típico para λ é zero, utilizado para dicionários com esparsidade inferior a 15%, como é caso, por exemplo, da ITU-T G.729 e TIA IS-641. Fazendo $\lambda = 0$ na equação (5.37) resulta que:

$$\mathbf{b} = \frac{\mathbf{d}}{\|\mathbf{d}\|} \tag{5.38}$$

E, como é de interesse apenas o sinal dos elementos de $\mathbf{b}$, esse caso se torna semelhante ao representado pela equação (5.33):

$$\mathbf{b} = \mathbf{d} \tag{5.39}$$

Outro caso comum é a utilização de $\lambda = 1/2$, que acontece em dicionários com esparsidade de cerca de 20%, como é o caso, por exemplo, do GSM *Enhanced Full Rate* e do GSM A*daptive Multi-Rate* nas suas duas taxas mais altas. Nesse caso, da equação (5.37), tem-se que:

$$\mathbf{b} = \frac{\mathbf{d}}{2\|\mathbf{d}\|} + \frac{\mathbf{u}'}{2\|\mathbf{u}'\|} \tag{5.40}$$

As divisões por dois não interferem nos sinais dos elementos de $\mathbf{b}$, resultando:

$$\mathbf{b} = \frac{\mathbf{d}}{\|\mathbf{d}\|} + \frac{\mathbf{u}'}{\|\mathbf{u}'\|} \tag{5.41}$$

É importante ressaltar que o vetor $\mathbf{b}$, além de servir para a estimação das amplitudes, também pode atuar num contexto mais geral como um vetor de estimação de probabilidade das posições dos pulsos (*pulse-position likelihood-estimate vector*). Ou seja, o valor $b[k]$ do vetor de estimação $\mathbf{b}$ está relacionado com a possibilidade de existir um pulso na posição k do vetor-código ótimo.

5.5 Busca Exaustiva de Posições

A busca exaustiva de posições (*position-exhaustive search*) representa uma simplificação da busca ótima de posições e amplitudes. A simplificação considerada consiste justamente na pré-associação de amplitudes às posições de acordo com o descrito na Seção 5.4. Dessa forma, enquanto que no algoritmo ótimo é feita uma busca de posições para cada combinação de amplitudes, na busca exaustiva de posições é feita apenas uma única varredura sobre as combinações de posições sendo considerado que as amplitudes do vetor-código escolhido serão dadas automaticamente em função das posições encontradas para os pulsos, de acordo com a equação (5.32).

Considerando a pré-associação de amplitudes às posições é conveniente definir o vetor $\mathbf{d}'$ e a matriz $\mathbf{\Phi}'$ como:

$$d'[n] = A_n d[n] \tag{5.42}$$

$$\phi'[k,\ell] = A_k A_\ell \phi[k,\ell] \tag{5.43}$$

Dessa forma, as equações (5.11) e (5.13) podem ser reescritas da seguinte forma:

$$C_i = \sum_{k=0}^{N-1} d'[m_k] \tag{5.44}$$

$$E_{\mathbf{c}_i} = \sum_{k=0}^{N} \phi'[m_k, m_k] + 2\sum_{k=1}^{N-1}\sum_{\ell=0}^{k-1} \phi'[m_k, m_\ell] \tag{5.45}$$

Por sua vez, o cálculo recursivo da correlação C_i e da energia $E_{\mathbf{c}_i}$, utilizado no esquema de laços encaixados, é feito através das equações:

$$\begin{cases} C_i^{(j)} = C_i^{(j-1)} + d'[m_j] \\ E_{\mathbf{c}_i}^{(j)} = E_{\mathbf{c}_i}^{(j-1)} + \phi'[m_j, m_j] + 2\sum_{k=0}^{j-1} \phi'[m_k, m_j] \end{cases} \tag{5.46}$$

Como exemplo, a Figura 5.7 mostra um pseudo-código referente à implementação da busca exaustiva no codificador G.729, que possui o dicionário algébrico representado pela Tabela 5.2. Esse caso é bastante semelhante ao algoritmo de busca ótima apresentado na Figura 5.4, com a excessão de que no caso em questão são utilizados o vetor $\mathbf{d}'$ e a matriz $\mathbf{\Phi}'$, calculados a partir da pré-associação de amplitudes às posições.

```
For (m0 = 0, m0 < 40, m0 += STEP) {
    C0 = d'(m0);
    E0 = φ'(m0,m0);
    For (m1 = 1, m1 < 40, m1 += STEP) {
        C1 = C0 + d'(m1);
        E1 = E0 + φ'(m1,m1) + 2*φ'(m0,m1);
        For (m2 = 2, m2 < 40, m2 += STEP) {
            C2 = C1 + d'(m2);
            E2 = E1 + φ'(m2,m2) + 2*φ'(m0,m2) + 2*φ'(m0,m2);
            For (m3 = 3, m3 < 40, m3 += STEP) {
                C3 = C2 + d'(m3);
                E3 = E2 + φ'(m3,m3) + 2*φ'(m0,m3) + 2*φ'(m1,m3) + 2*φ'(m2,m3);
                VerificaMaximo(C3, E3, m0, m1, m2, m3);
            }
            For (m3 = 4, m3 < 40, m3 += STEP) {
                C3 = C2 + d'(m3);
                E3 = E2 + φ'(m3,m3) + 2*φ'(m0,m3) + 2*φ'(m1,m3) + 2*φ'(m2,m3);
                VerificaMaximo(C3, E3, m0, m1, m2, m3);
            }
        }
    }
}
```

Figura 5.7: Pseudo-código que representa uma implementação do algoritmo de busca exaustiva de posições no dicionário algébrico da Tabela 5.2.

Como não realiza a busca de amplitudes, a busca exaustiva de posições corresponde a uma redução de complexidade na proporção de $2^N : 1$ em relação à busca ótima. Entretanto, ainda assim a busca exaustiva de posições apresenta custo computacional muito alto, não sendo utilizada na prática. Atualmente esse tipo de busca possui utilidade apenas como parâmetro de comparação entre diferentes algoritmos.

5.6 Busca Focalizada

Como apresentado na Seção 5.5, a busca exaustiva de posições utiliza a pré-associação de amplitudes às posições e calcula os valores de C_i e $E_{\mathbf{q}_i}$ através de um esquema de laços encaixados para todas as combinações de posições.

Uma alternativa possível para diminuir a complexidade do processo de busca é inserir restrições aos laços, de forma a reduzir o número de combinações de posições a serem testadas. Nesse sentido, a idéia da busca focalizada (*focused search*) consiste em impor como restrição um limite mínimo para o valor acumulado no cálculo de C_i.

Em geral, a restrição utilizada na busca focalizada é referente apenas a busca do último pulso. Dessa forma, a entrada no(s) laço(s) referente(s) ao último pulso é permitida somente caso o valor acumulado até então $C_i^{(N-2)}$ seja maior do que um limiar predefinido:

$$C_i^{(N-2)} > C_{thr}, \tag{5.47}$$

onde C_{thr} é uma constante que estabelece o limite mínimo para a busca do último pulso, e $C_i^{(N-2)}$ é o valor acumulado até a penúltima etapa do cálculo de C_i:

$$C_i^{(N-2)} = \sum_{k=0}^{N-2} d'[m_k] \tag{5.48}$$

O uso do limite mínimo C_{thr} baseia-se no fato de que os $N-1$ primeiros pulsos apresentam menor probabilidade de constituírem uma boa solução caso o valor parcial de C_i calculado para eles seja pequeno. O algoritmo focalizado então prefere descartar tais posições ao invés de investir na busca pelo último pulso.

Geralmente define-se o valor de C_{thr} em função dos valores médio e máximo de $C_i^{(N-2)}$:

$$C_{thr} = [C_i^{(N-2)}]_{med} + K([C_i^{(N-2)}]_{max} - [C_i^{(N-2)}]_{med}), \qquad 0 \leq K \leq 1 \tag{5.49}$$

onde os valores $[C_i^{(N-2)}]_{max}$ e $[C_i^{(N-2)}]_{med}$ representam respectivamente os valores máximo e médio de $C_i^{(N-2)}$ calculados sobre as posições dos primeiros $N-1$ pulsos. Para o caso de um dicionário completo[2], por exemplo, tem-se que:

$$[C_i^{(N-2)}]_{max} = \sum_{\ell=0}^{N-2} \max_{0 \le k \le L_\ell} \{d'[kS + \varphi_\ell]\} \tag{5.50}$$

$$[C_i^{(N-2)}]_{med} = \frac{1}{N-1} \sum_{\ell=0}^{N-2} \sum_{k=0}^{L_\ell} d'[kS + \varphi_\ell] \tag{5.51}$$

Ainda na equação (5.49) o fator limitante (*threshold factor*) K determina a quantidade de combinações de posições a serem testadas. Quanto maior o valor de K, maior será o limite C_{thr}, e conseqüentemente menor será o número de combinações verificadas. No caso extremo de K tendendo a 1, por exemplo, o limite C_{thr} tenderia ao valor máximo de $C_i^{(N-2)}$, estando a busca restrita a apenas um vetor-código, aquele que maximiza $C_i^{(N-2)}$.

Enquanto que o limiar representado pela constante C_{thr} é calculado antes de cada processo de busca, o valor de K, por outro lado, é fixo e se constitui como uma característica do codificador. Nos codificadores do G.729 e G.723.1 da ITU-T, por exemplo, são utilizados para K os valores 0.4 e 0.5 respectivamente.

Para reduzir ainda mais a complexidade do processo de busca, muitas vezes também é estabelecido na busca focalizada um limiar máximo para o número de vezes que o último pulso é procurado. No codificador G.729, por exemplo, é estabelecido o limite por bloco de 180 buscas do último pulso, que devem ser divididas entre os dois sub-blocos existentes, resultando em uma média de 90 buscas por sub-bloco. Além disso, é estabelecido nesse caso, para evitar uma distribuição desigual entre os sub-blocos, um limite para o primeiro sub-bloco de 105 buscas do último pulso.

Dessa forma, o algoritmo de busca focalizada possui dois fatores de controle da relação complexidade *versus* qualidade, o limite do número de buscas do último pulso e o fator λ que regula o limiar mínimo para $C_i^{(N-2)}$.

A Figura 5.8 mostra um pseudo-código relativo à busca focalizada onde é considerado o dicionário fixo do codificador G.729 da ITU-T, apresentado na Tabela 5.2. Nesse caso existem 4 pulsos, e um limiar mínimo é imposto após o terceiro pulso. A condição para a entrada nos últimos laços, $\mathtt{C2} \geq \mathtt{C_{thr}}$, descarta as combinações de posições que provavelmente

[2]Como apresentado na Seção 5.1.2, um dicionário completo possui um único pulso para cada trilha.

```
For (m0 = 0, m0 < 40, m0 += STEP) {
    C0 = d'(m0);
    E0 = φ'(m0,m0);
    For (m1 = 1, m1 < 40, m1 += STEP) {
        C1 = C0 + d'(m1);
        E1 = E0 + φ'(m1,m1) + 2*φ'(m0,m1);
        For (m2 = 2, m2 < 40, m2 += STEP) {
            C2 = C1 + d'(m2);
            E2 = E1 + φ'(m2,m2) + 2*φ'(m0,m2) + 2*φ'(m0,m2);
            If (C2 >= Cthr) {
                For (m3 = 3, m3 < 40, m3 += STEP) {
                    C3 = C2 + d'(m3);
                    E3 = E2 + φ'(m3,m3) + 2*φ'(m0,m3) + 2*φ'(m1,m3) + 2*φ'(m2,m3);
                    VerificaMaximo(C3, E3, m0, m1, m2, m3);
                }
                For (m3 = 4, m3 < 40, m3 += STEP) {
                    C3 = C2 + d'(m3);
                    E3 = E2 + φ'(m3,m3) + 2*φ'(m0,m3) + 2*φ'(m1,m3) + 2*φ'(m2,m3);
                    VerificaMaximo(C3, E3, m0, m1, m2, m3);
                }
                counter = counter - 1;
                If (counter == 0) {
                    Goto END_SEARCH;
                }
            }
        }
    }
}
END_SEARCH:
```

Figura 5.8: Pseudo-código que representa uma implementação do algoritmo de busca focalizada de posições no dicionário algébrico da Tabela 5.2. A variável `counter` é utilizada para a contagem do número de buscas do último pulso.

resultarão em valores pequenos para `C3`. A variável `counter` é utilizada para a contagem do número de vezes em que o último pulso foi procurado.

5.7 Busca em Árvore por Profundidade

A busca em árvore por profundidade (DFTS - *depth-first tree search*) [2] é um dos métodos de busca de baixa complexidade mais bem sucedidos, sendo utilizado em vários codificadores ACELP atuais, como por exemplo no G.729A e no GSM-EFR.

5.7.1 Árvores de Busca

Antes de apresentar o algoritmo DFTS é conveniente descrever a representação de algoritmos de busca através de árvores de busca. A Figura 5.9(a) mostra um exemplo de uma estrutura de árvore referente à busca exaustiva de posições em um dicionário com 5 pulsos, onde existem quatro posições possíveis para cada pulso. O nó inicial, ou nó do nível zero, conecta-se a quatro nós do nível um, relativos às quatro possibilidades de escolha do primeiro pulso. Cada um desses nós se conecta a quatro nós do nível seguinte, referentes a escolha do pulso seguinte, e assim por diante. Dessa forma, os nós terminais da árvore da Figura 5.9(a) ilustram todas as combinações possíveis de posições para os pulsos no dicionário considerado. A técnica de busca exaustiva de posições, como apresentada, basicamente verifica os nós terminais da árvore da esquerda para a direita. A Figura 5.9(b) ilustra a mesma árvore, ressaltando a utilização de um algoritmo de busca focalizada. Nesse caso, para reduzir a complexidade do processo, concentra-se a busca nas regiões mais promissoras da árvore. O prosseguimento para os níveis mais baixos deixa de ser sistemático, e passa a depender de que o desempenho do caminho exceda algum limiar estabelecido.

5.7.2 Algoritmo DFTS

Assim como a busca focalizada, o algoritmo DFTS consiste exclusivamente em um processo de busca de posições, sendo os sinais dos pulsos determinados através de uma pré-associação de amplitudes às posições, como apresentado na Seção 5.4.

Para a determinação das posições o algoritmo DFTS considera que os N pulsos a serem encontrados são divididos em P grupos, cada grupo contento N_p pulsos, $p = 0, 1, \ldots, P-1$, de forma que:

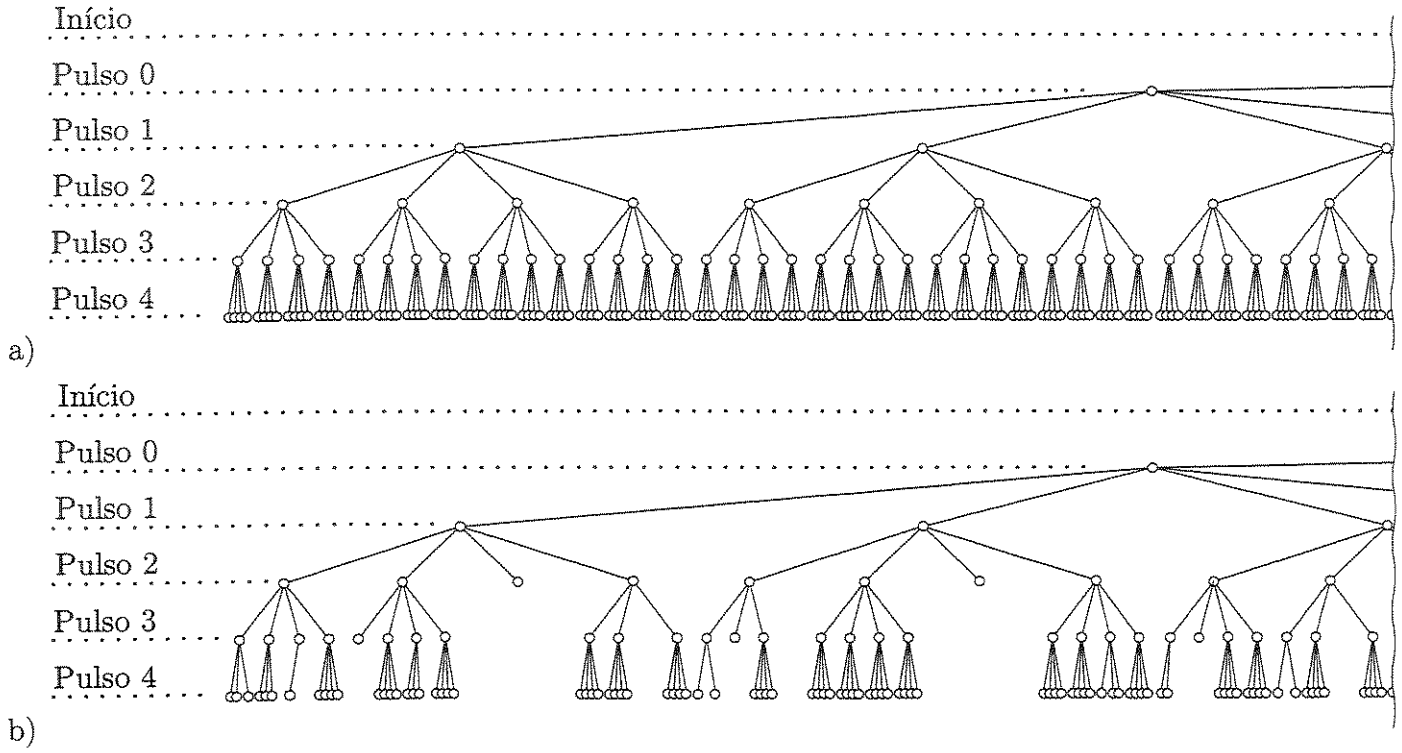


Figura 5.9: Exemplos de árvores de busca para a busca (a) exaustiva, e (b) focalizada.

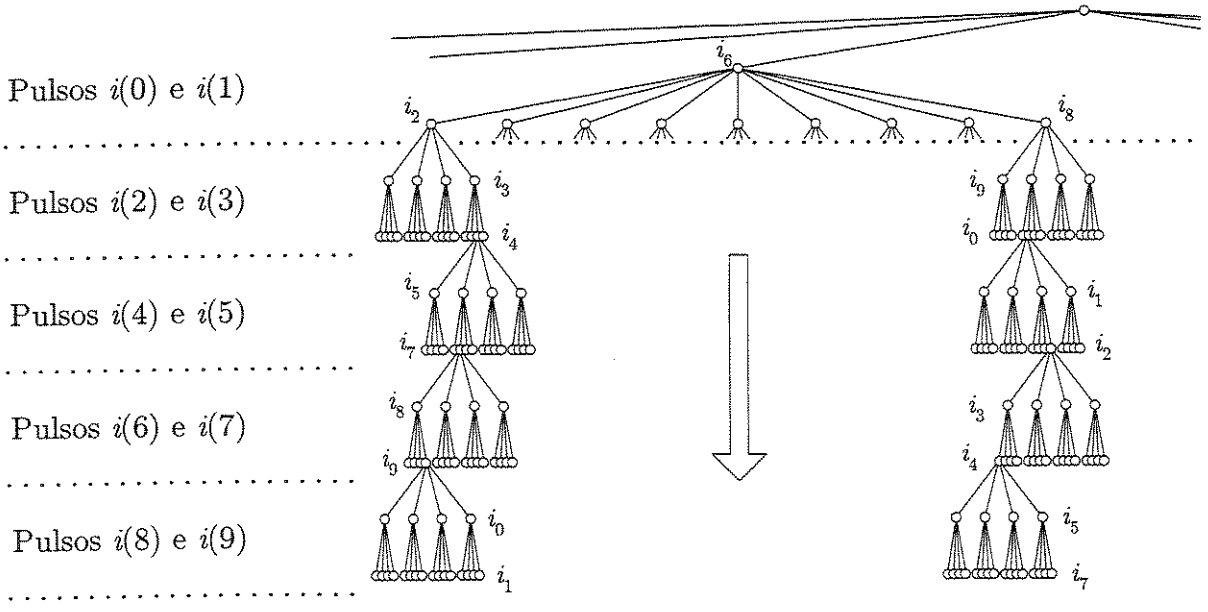


Figura 5.10: Árvore de busca do algoritmo DFTS apresentado na Tabela 5.5.

$$N_0 + N_1 + \ldots + N_{P-1} = N \tag{5.52}$$

Um dado conjunto de posições para os j primeiros pulsos, onde

$$j = N_0 + N_1 + \ldots + N_p, \tag{5.53}$$

é chamado de um caminho de comprimento j, ou ainda de um caminho de nível p.

Considerando a divisão dos pulsos em grupos, a busca das posições é feita por etapas, onde a etapa p consiste na análise dos N_p pulsos do grupo p.

A busca tem início com o primeiro grupo. O objetivo da busca no nível zero, entretanto, não é a determinação final das posições dos N_0 pulsos do primeiro grupo, e sim apenas a determinação de um ou mais caminhos de comprimento N_0 a serem analisados nas etapas seguintes, que são chamados de caminhos candidatos (*candidate paths*). A busca prossegue então nos grupos subseqüentes, de forma que um certo nível p procura-se estender os caminhos candidatos resultantes da busca no nível anterior, de forma a transformar cada um deles em um ou mais caminhos candidatos de nível p. Ou seja, com a anexação de N_p pulsos cada caminho candidato do nível $p-1$ se estende em um ou mais caminhos candidatos de comprimento $N_0 + N_1 + \ldots + N_p$. Ao final do processo, é escolhido o vetor código cujas posições dos pulsos descrevem o caminho de comprimento N que maximiza τ_i.

Vale ressaltar que na bifurcação de um caminho em um ou mais caminhos de maior comprimento são geralmente utilizados como critério ou o vetor $\mathbf{b}$ de estimação de probabilidade das posições dos pulsos, ou a maximização do valor parcial de τ_i referente a j pulsos:

$$\tau_i^{(j-1)} = \frac{C_i^{(j-1)}}{E_{\mathbf{q}_i}^{(j-1)}}, \tag{5.54}$$

onde os valores parciais de correlação e de energia são calculados através de (5.46).

Nos diferentes algoritmos DFTS adotados atualmente não existe uma regra pré-estabelecida para o critério a ser utilizado em cada etapa da busca. Pode-se dizer entretanto, que em geral o vetor $\mathbf{b}$ é utilizado na busca no primeiro nível para selecionar os primeiros pulsos dos caminhos candidatos, sendo o fator $\tau_i^{(j-1)}$ utilizado na busca nos outros níveis. Também não existe regra pré-estabelecida para a escolha do número de grupos P, do número de pulsos por grupo N_p, e nem do número bifurcações a serem escolhidas a cada etapa. Tais fatores são escolhidos para cada codificador como melhor lhe convém, de acordo

com a estrutura do dicionário e o com a qualidade de voz almejada.

Um exemplo bastante elucidativo é o apresentado pela Tabela 5.5, que mostra um procedimento de busca DFTS para um dicionário com $L = 40$ posições e $N = 10$ pulsos, onde existe exatamente 1 pulso por trilha, que deve ocupar uma das 4 posições da trilha ($L_k = 4$).

Tabela 5.5: Exemplo de técnica de busca DFTS.

Dicionário algébrico				
$L = 40$, $N = 10$, $L_0 = L_1 = \ldots L_9 = 4$				
Procedimento de busca				
Nível (p)	Número de pulsos (N_p)	Caminhos candidatos	Regra	Critério de seleção
0	2	9	R1	$\mathbf{b}$
1	2	1	R2	$\tau_i^{(3)}$
2	2	1	R2	$\tau_i^{(5)}$
3	2	1	R2	$\tau_i^{(7)}$
4	2	1	R2	$\tau_i^{(9)}$

No algoritmo DFTS apresentado por esta tabela, os 10 pulsos são divididos em 5 grupos de 2 pulsos cada. A busca no nível zero tem como critério de seleção o vetor $\mathbf{b}$, enquanto que a busca em outros níveis procura maximizar os valores parciais de τ_i. As regras R1 e R2 são:

R1. A posição do primeiro pulso é escolhida como aquela onde se encontra o valor absoluto máximo de $b[n]$, sendo analisadas todas as posições do vetor[3]:

$$m_{i(0)} = \underset{0 \leq n < L}{\operatorname{argmax}} \{|b[n]|\} \tag{5.55}$$

[3]Como o pulso é procurado em todas as posições do vetor-código, independente de fase, o primeiro pulso a ser encontrado não corresponde necessariamente ao pulso i_0. Em todo o restante do processo de busca a ordem cronológica de determinação dos pulsos não corresponde (necessariamente) à ordem seqüencial $i_0, i_1, \ldots, i_{N-1}$. Nesse sentido a função $i(k)$, chamada de função de ordem dos pulsos (*pulse-order function*), é utilizada para representar a relação entre os pulsos e a ordem cronológica de determinação dos mesmos. Dessa forma, $i(k)$ representa o k-ésimo pulso a ser determinado, que possui posição $m_{i(k)}$ e amplitude $\alpha_{i(k)}$.

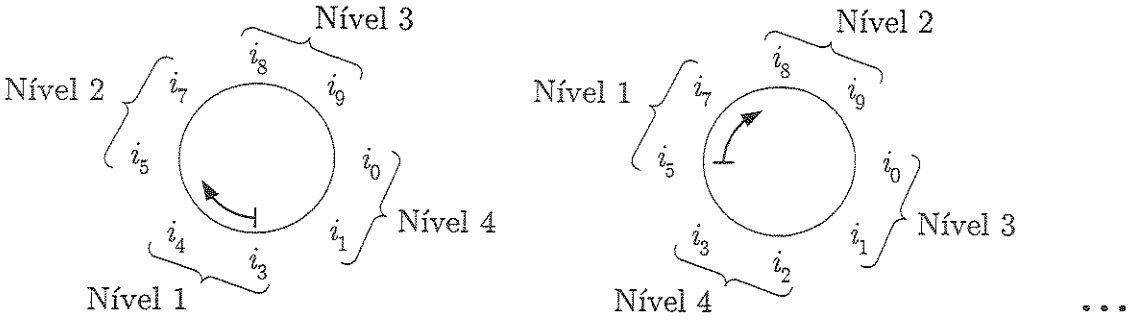


Figura 5.11: Exemplo de determinação da função de ordem dos pulsos para o caso do algoritmo da Tabela 5.5. São apresentadas as funções de ordem dos pulsos para dois caminhos candidatos. Os dois primeiros pulsos, $i(0)$ e $i(1)$, são respectivamente i_6 e i_2 no primeiro caso, e i_6 e i_4 no segundo caso.

São escolhidas 9 posições para o segundo pulso $i(1)$, que correspondem aos máximos de $|b[n]|$ em cada uma das nove trilhas restantes. A combinação de $i(0)$ com cada uma das 9 posições para $i(1)$ gera os 9 caminhos candidatos resultantes da busca no nível zero. O processo está representado pela Figura 5.10.

Ao fim da busca no nível 1 já pode ser determinada a função completa de ordem dos pulsos $i(k)$, que será utilizada no restante do processo de busca para cada caminho candidato. Essa determinação é feita dispondo os 8 pulsos restantes em um círculo escolhendo-os no sentido horário a partir do número que estaria à direita de $i(1)$. A Figura 5.11 mostra um exemplo onde o primeiro pulso $i(0)$ é i_6. Esse procedimento, que recebe o nome de permutação cíclica, é comum e inclusive caracteriza os algoritmos DFTS.

R2. Para os níveis seguintes os dois pulsos de cada nível são determinados através de uma busca exaustiva. Ou seja, para cada nível p são verificadas todas as 16 (4x4) combinações de posições para os dois pulsos em questão, $i(2p)$ e $i(2p+1)$. Os dois pulsos que maximizam $\tau_i^{(2p+1)}$ são escolhidos e anexados ao caminho considerado, de forma que de cada caminho candidato analisado resulta apenas um caminho candidato estendido.

Ao fim de todo o processo, os nove caminhos candidatos de comprimento N são comparados e escolhe-se aquele que produz melhor resultado no sentido da maximização do valor final $\tau_i^{(N-1)}$.

5.8 Busca Seqüencial de Posições

A busca seqüencial de posições (*position-sequential search*) é proposta aqui como uma alternativa de algoritmo de busca de baixa complexidade para dicionários algébricos [5]. O princípio utilizado para a redução da complexidade é a determinação das posições dos pulsos de maneira seqüencial, ou seja, uma após o outra.

Assim como na busca focalizada e no algoritmo DFTS, na busca seqüencial são determinadas apenas as posições dos pulsos, havendo uma associação prévia de amplitudes às posições, como o descrito na Seção 5.4.

Para a busca seqüencial convém reescrever as formas recursivas apresentadas em (5.46) da seguinte forma:

$$C^{(j)}_{m_{i(j)}} = C^{(j-1)}_{m_{i(j-1)}} + d'[m_{i(j)}] \tag{5.56}$$

$$E^{(j)}_{m_{i(j)}} = E^{(j-1)}_{m_{i(j-1)}} + \phi'[m_{i(j)}, m_{i(j)}] + 2\sum_{k=0}^{j-1} \phi'[m_{i(k)}, m_{i(j)}] \tag{5.57}$$

onde $i(k)$ é a função de ordem dos pulsos, apresentada na seção anterior e utilizada para fazer referência aos pulsos na ordem cronológica em que foram determinados. Além disso, o termo $m_{i(j)}$ é utilizado como índice em $C^{(j)}_{m_{i(j)}}$ e $E^{(j)}_{m_{i(j)}}$ para enfatizar a busca seqüencial, onde a determinação dos novos valores de correlação e energia é função apenas do novo pulso.

A busca seqüencial tem início com a determinação do primeiro pulso, que é feita da seguinte maneira:

$$m_{i(0)} = \underset{n}{\text{argmax}} \left\{ \frac{(d'[n])^2}{\phi'(n,n)} \right\}, \tag{5.58}$$

onde são verificadas todas as posições do vetor-código.

A busca prossegue então na determinação dos pulsos seguintes, um a um. Dessa forma, dados j pulsos já encontrados, a busca de um novo pulso é feita nas posições restantes do vetor-código, de acordo com o dicionário utilizado. No processo, as posições já encontradas $m_{i(0)}, m_{i(1)}, \ldots, m_{i(j-1)}$ são consideradas como fixas, e procura-se maximizar:

$$\tau^{(j)}_{m_{i(j)}} = \frac{\left(C^{(j)}_{m_{i(j)}}\right)^2}{E^{(j)}_{m_{i(j)}}} \tag{5.59}$$

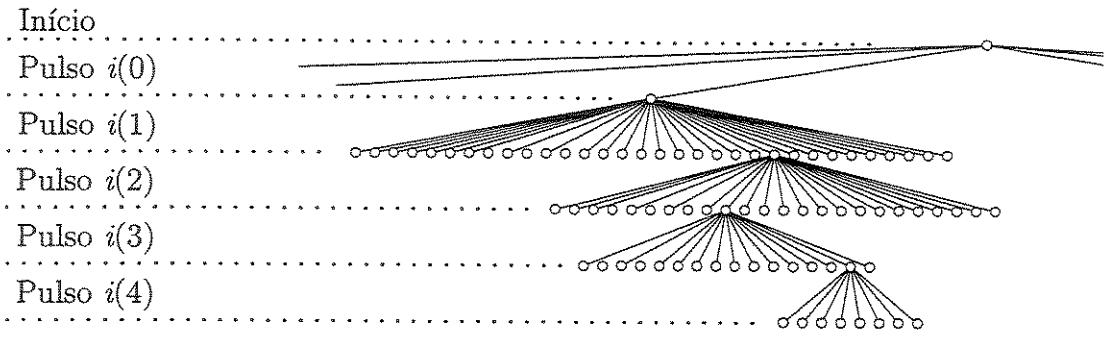


Figura 5.12: Árvore de busca do algoritmo de busca seqüencial. O dicionário considerado possui 40 posições divididas em 5 trilhas, existindo um pulso para cada trilha.

Após N iterações são determinadas as N posições dos pulsos, e o algoritmo chega ao fim.

A cada etapa o número de posições verificadas é menor, com a exclusão das posições relativas às trilhas que já estão ocupadas. A Figura 5.12 mostra a árvore de busca correspondente a utilização de um algoritmo de busca seqüencial em um dicionário completo de 40 posições com 5 trilhas e um pulso por trilha.

Pode-se notar que o algoritmo em questão assemelha-se a busca multi-pulso, diferenciando-se basicamente pela pré-associação de amplitudes às posições. Além disso, a busca seqüencial também pode ser considerada como um caso degenerado da busca em árvore por profundidade, onde existe apenas um pulso por grupo e onde a regra de escolha dos pulsos no nível p é a busca em todas as posições restantes procurando-se maximizar $\tau_i^{(p)}$. Ao contrário do que caracteriza o algoritmo DFTS, entretanto, a busca seqüencial não utiliza o vetor $\mathbf{b}$ em nenhuma etapa da busca como um vetor de estimação da probabilidade das posições dos pulsos. De qualquer maneira, mesmo que sua inspiração tenha se apoiado em outros codificadores, o algoritmo apresentado aqui não foi proposto anteriormente na literatura e nem é utilizado em qualquer padrão de codificação.

A Figura 5.13 mostra o pseudo-código referente a uma implementação do algoritmo de busca seqüencial para o dicionário algébrico da Tabela 5.2. Cada iteração do laço maior resulta na determinação da posição de um pulso, e as posições encontradas são armazenadas na ordem cronológica no vetor `m(j)`, que representa $m_{i(j)}$. Para que a busca de um novo pulso seja feita apenas nas posições restantes, é utilizado um vetor chamado $\varphi_{\texttt{OCCUPIED}}$, que possui um elemento para cada trilha, sendo atribuído o valor `FALSE` para as trilhas livres e `TRUE` para trilhas que já possuem pulso. Para cada posição `mj` restante são calculados

```
/* Procura cada um dos 4 pulsos */
For (j = 0, j < 4, j += 1) {
    Csq_max = 0; E_max = 1;
    /* Para cada trilha */
    For (φ = 0, φ < 5, φ += 1) {
        /* Verifica se a trilha está livre */
        If (φ_OCCUPIED(φ) == FALSE) {
            /* Testa cada posição da trilha */
            For (mj = φ, mj < 40, mj += STEP) {
                Ej = E0 + φ'(mj,mj)+2∑_{k=0}^{j-1} φ'(m(k),mj);
                Cj = C0 + d'(mj);
                Csq = Cj*Cj;
                If (Csq*E_max > Csq_max*Ej) {
                    Csq_max = Csq; C_max = Cj;E_max = Ej; mj_max = mj; φ_max = φ;
                }
            }
        }
    }
    /* Atualiza a correlação e a energia */
    E0 = E_max;
    C0 = C_max;
    /* Registra a nova posição e marca a trilha como ocupada */
    m(j) = mj_max;
    If (φ_max < 3) {
        φ_OCCUPIED(φ_max) = TRUE;
    }
    else {
        φ_OCCUPIED(3) = TRUE;
        φ_OCCUPIED(4) = TRUE;
    }
}
```

Figura 5.13: Pseudo-código referente à uma implementação do algoritmo de busca seqüencial de posições para o codificador G.729, que possui o dicionário algébrico apresentado na Tabela 5.2.

recursivamente os valores de $C^{(j)}_{m_{i(j)}}$ e $E^{(j)}_{m_{i(j)}}$, que são representados respectivamente pelas variáveis `Cj` e `Ej`. Após a determinação do pulso as trilhas devidas são marcadas como ocupadas, de acordo com o dicionário adotado, e as variáveis `E0` e `C0` utilizadas nos cálculos recursivos são atualizadas.

Numa implementação mais eficiente a primeira iteração do laço principal poderia ter sido isolada e executada separadamente. Dessa forma a determinação do primeiro pulso, que consiste simplesmente na verificação de (5.58) para todas as posições do vetor-código, seria feita mais rapidamente.

Por fim, vale ressaltar que a busca seqüencial de posições utiliza bem menos elementos da matriz de correlação impulsiva $\mathbf{\Phi}$ do que as buscas exaustiva, focalizada e DFTS. Para alguns dicionários tal fato pode representar uma redução a mais no processamento, num esquema onde os elementos de $\mathbf{\Phi}$ (e $\mathbf{\Phi}'$) deixam de ser calculados previamente e passam a ser calculados dentro do próprio algoritmo de busca, conforme o necessário. Tais implementações são válidas apenas para casos onde o número de correlações utilizadas seja realmente pequeno, já os elementos de $\mathbf{\Phi}$ são calculados recursivamente, ou seja, o cálculo de um elemento passa automaticamente pelo cálculo de outros elementos da mesma diagonal (Seção 5.2.1). Em geral, deixar de calcular a matriz $\mathbf{\Phi}$ previamente não representa um ganho. Entretanto, em qualquer caso é válido calcular a matriz de correlação impulsiva previamente e, de acordo com o necessário, calcular os elementos de $\mathbf{\Phi}'$ dentro do algoritmo de busca.

5.9 Busca Conjunta de Posição e Amplitude

A busca conjunta de posição e amplitude, ou JPAS (*Joint Position and Amplitude Search*), foi proposta por M. A Ramírez e M. Gerken [30, 32] como uma alternativa de algoritmo de busca de baixa complexidade para codificadores algébricos.

Pode-se dizer que a busca conjunta de posição e amplitude surge na verdade da análise geométrica do problema da análise-por-síntese, sendo geralmente dado um enfoque geométrico à apresentação do mesmo. A seguir, entretanto, será feita como alternativa uma apresentação diferente da busca conjunta. A interpretação geométrica do processo é dada em seguida, na Seção 5.9.2.

5.9.1 Funcionamento do Algoritmo JPAS

Diferentemente dos algoritmos de busca apresentados até aqui, o processo de busca conjunta consiste na determinação da amplitude e da posição de cada pulso, um após o outro[4]. O algoritmo JPAS executa N iterações, sendo definido um pulso a cada iteração. Ao fim da última iteração o algoritmo chega ao vetor-código completo, com seus N pulsos.

Dados j pulsos já encontrados, a busca de um novo pulso é feita nas posições restantes do vetor-código, de acordo com o dicionário utilizado. No processo as posições já encontradas $m_{i(0)}, m_{i(1)}, \ldots, m_{i(j-1)}$ são consideradas como fixas, e procura-se maximizar $\tau^{(j)}_{m_{i(j)}}$. Esse caso entretanto difere da busca seqüencial apresentada anteriormente por que as amplitudes dos pulsos já encontrados não são consideradas fixas. Inclusive, essa é uma das características mais importantes da busca conjunta: *enquanto que as posições encontradas são mantidas fixas para a busca do próximo pulso, as amplitudes correspondentes são reajustadas conjuntamente a cada etapa.*

O reajuste conjunto das amplitudes antigas durante a busca de um novo pulso pode ser melhor esclarecido com a ajuda da Figura 5.14, que representa uma iteração do algoritmo JPAS.

Na Figura 5.14 constam os três instantes seguintes:

1. Na iteração $j-1$ já foram definidas j posições para os pulsos. Para cada posição de pulso existe uma amplitude correspondente, constituindo o conjunto de amplitudes $(\alpha^{(j-1)}_{i(0)}, \alpha^{(j-1)}_{i(1)}, \ldots, \alpha^{(j-1)}_{i(j-1)})$, onde $\alpha^{(j-1)}_{i(k)}$ é a amplitude do pulso k na etapa $j-1$.

2. A busca do pulso $i(j)$ é feita nas posições restantes considerando o novo pulso como positivo e testando duas possibilidades: a combinação do novo pulso com os pulsos já encontrados, e a combinação do novo pulso com os pulsos já encontrados com suas amplitudes invertidas[5]. A inversão ou não das amplitudes dos pulsos calculados até a etapa $j-1$ é representada por $\sigma^{(j-1)}$. Mais especificamente, $\sigma^{(j-1)} = -1$ representa a inversão das amplitudes, enquanto que $\sigma^{(j-1)} = 1$ representa a sua manutenção.

3. Encontrado o novo pulso, ajusta-se o sinal do vetor resultante de forma que o ganho correspondente seja positivo (um processo semelhante foi descrito para a busca ótima

[4]Por essa razão o algoritmo de busca conjunta de posição e amplitude será classificado aqui, juntamente com a busca seqüencial de posições, como um algoritmo de busca seqüencial.

[5]Um caso análogo seria o de manter as amplitudes dos pulsos anteriores fixas e testar as amplitudes positiva e negativa para o novo pulso. Entretanto, o método apresentado resulta em equações mais simples.

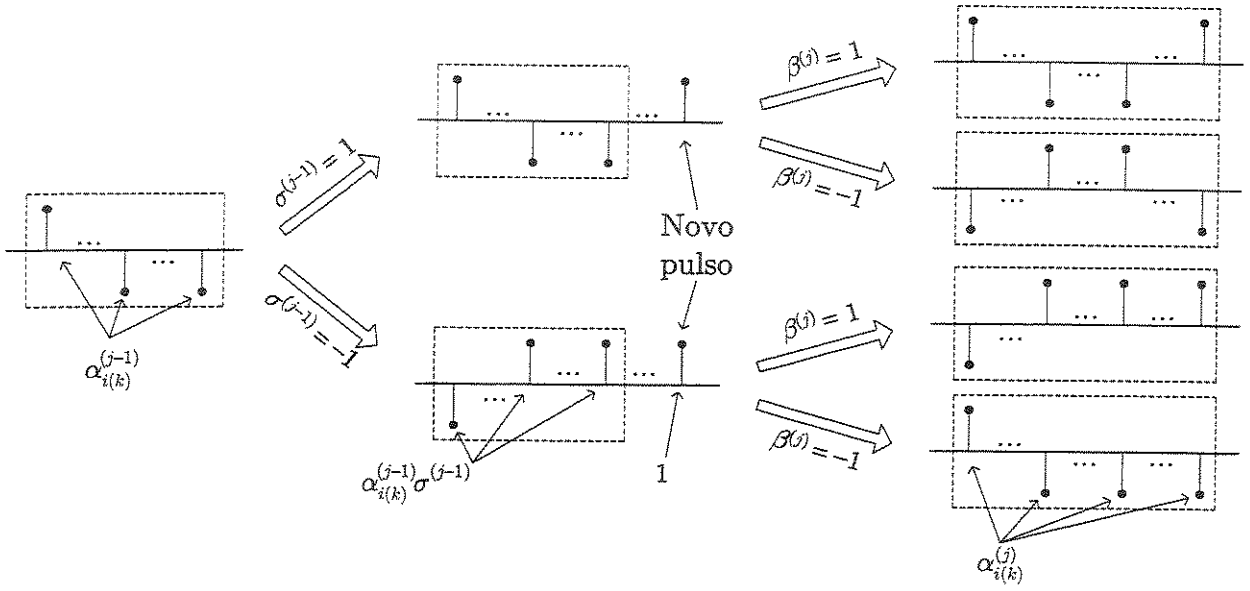


Figura 5.14: Reajuste das amplitudes na busca busca conjunta.

na Seção 5.3). A variável $\beta^{(j)}$ representa a inversão ou não do vetor-código parcial. Caso seja necessária a inversão, $\beta^{(j)} = -1$, caso contrário $\beta^{(j)} = 1$.

Após a aplicação do fator $\beta^{(j)}$, as novas amplitudes $\alpha_{i(k)}^{(j)}$ podem ser dadas por:

$$\alpha_{i(k)}^{(j)} = \begin{cases} \beta^{(j)} & \text{para } k = j \\ \beta^{(j)}\sigma^{(j-1)}\alpha_{i(k)}^{(j-1)} & \text{para } k < j \end{cases} \tag{5.60}$$

Dessa maneira, pode-se dizer que em uma iteração não é encontrada a amplitude do novo pulso, mas sim uma relação entre ela e as amplitudes dos pulsos já encontrados, sendo a amplitude propriamente dita determinada apenas após a última iteração $j = N - 1$, pelos valores $\alpha_{i(k)}^{(N-1)}$.

Feitas essas considerações sobre o reajuste dos sinais dos pulsos, pode-se descrever com maior facilidade o funcionamento geral da busca conjunta. O algoritmo começa procurando o primeiro pulso em todas as posições do vetor-código. A posição escolhida deve satisfazer:

$$m_{i(0)} = \underset{n}{\text{argmax}} \left\{ \frac{(d[n])^2}{\phi(n,n)} \right\}, \tag{5.61}$$

onde n varia de 0 a $L-1$, e o índice $i(0)$ se refere à busca do primeiro pulso.

Encontrado o primeiro pulso determina-se sua amplitude de forma que o ganho correspondente seja positivo, ou seja, $\alpha^{(0)} = \text{sign}\{C^{(0)}_{m_{i(0)}}\}$. Pode-se notar que esse processo de busca do primeiro pulso corresponde na verdade à encontrar a solução ideal para o caso onde existe apenas um pulso por vetor-código ($N = 1$).

Cada pulso seguinte é determinado verificando-se as posições restantes, de acordo com o dicionário adotado, e testando a contribuição de cada uma delas para os valores de $C^{(j)}_{m_{i(j)}}$ e $E^{(j)}_{m_{i(j)}}$. Para testar a contribuição de um novo pulso convém obter formas recursivas semelhantes às representadas por (5.20), que não podem ser utilizadas nesse caso por haver o reajuste conjunto das amplitudes dos pulsos a cada iteração (equação (5.60)).

Como mostra a Figura 5.14, no momento da busca do pulso $i(j)$ as amplitudes dos pulsos antigos são dadas por $\alpha^{(j-1)}_{i(k)}\sigma^{(j-1)}_{i(k)}$, enquanto que a amplitude do novo pulso é 1. Desse modo, da equação (5.16) resulta que:

$$C^{(j)}_{m_{i(j)}} = d[m_{i(j)}] + \sigma^{(j-1)} \sum_{k=0}^{j-1} \alpha^{(j-1)}_{i(k)} d[m_{i(k)}] \tag{5.62}$$

Usando a equação (5.60) para substituir o valor de $\alpha^{(j-1)}_{i(k)}$, chega-se a:

$$C^{(j)}_{m_{i(j)}} = d[m_{i(j)}] + \sigma^{(j-1)}\beta^{(j-1)} \left[d[m_{i(j-1)}] + \sigma^{(j-1)} \sum_{k=0}^{j-2} \alpha^{(j-2)}_{i(k)} d[m_{i(k)}] \right] \tag{5.63}$$

Pode-se notar que o termo entre colchetes na verdade corresponde a $C^{(j-1)}_{m_{i(j-1)}}$. Feita essa substituição tem-se que:

$$C^{(j)}_{m_{i(j)}} = d[m_{i(j)}] + \sigma^{(j-1)}\beta^{(j-1)} C^{(j-1)}_{m_{i(j-1)}} \tag{5.64}$$

O mesmo processo pode ser aplicado a $E^{(j)}_{m_{i(j)}}$. Da equação (5.18) resulta que no momento da busca $E^{(j)}_{m_{i(j)}}$ é dado por:

$$\begin{aligned} E^{(j)}_{m_{i(j)}} = & \sum_{k=0}^{j} \phi[m_{i(k)}, m_{i(k)}] + \\ & 2 \sum_{k=0}^{j-1} \left[\sigma^{(j-1)} \alpha^{(j-1)}_{k} \phi[m_{i(k)}, m_{i(j)}] + \sum_{k \neq \ell = 0}^{j-1} \left(\sigma^{(j-1)}\right)^2 \alpha^{(j-1)}_{i(k)} \alpha^{(j-1)}_{i(\ell)} \phi[m_{i(k)}, m_{i(\ell)}] \right] \end{aligned} \tag{5.65}$$

Utilizando (5.60) para substituir o valor das amplitudes, e considerando que $\left(\sigma^{(j)}\right)^2 = \left(\beta^{(j)}\right)^2 = 1$, pode-se chegar a seguinte fórmula recursiva:

$$E^{(j)}_{m_{i(j)}} = \phi[m_{i(j)}, m_{i(j)}] + 2\sigma^{(j-1)} \sum_{k=0}^{j-1} \alpha^{(j-1)}_{i(k)} \phi[m_{i(k)}, m_{i(j)}] + E^{(j-1)}_{m_{i(j-1)}} \tag{5.66}$$

As recursões (5.64) e (5.66) constituem a base para a escolha de um novo pulso na busca conjunta, no sentido de que na etapa j deve ser escolhida a posição m_j e amplitude $\sigma^{(j-1)}$ que maximizam $(C^{(j)}_{m_{i(j)}})^2/E^{(j)}_{m_{i(j)}}$:

$$\max_{m_{i(j)}, \sigma^{(j-1)}} \left\{ \frac{\left(C^{(j)}_{m_{i(j)}}\right)^2}{E^{(j)}_{m_{i(j)}}} \right\} \tag{5.67}$$

Note que se trata de encontrar dois fatores, a posição do pulso e o valor de $\sigma^{(j-1)}$, que pode ser apenas ± 1, indicando se há ou não inversão dos pulsos antigos. Devem ser feitas, portanto, duas verificações para cada posição, uma referente a $\sigma^{(j-1)}$ positivo, e outra para o valor negativo. É conveniente então definir dois valores a serem maximizados, $\tau^{(j)}_{P,m_{i(j)}}$ e $\tau^{(j)}_{S,m_{i(j)}}$[6], referentes respectivamente aos valores positivo e negativo de $\sigma^{(j-1)}$:

$$\tau^{(j)}_{P,m_{i(j)}} = \frac{\left(C^{(j)}_{P,m_{i(j)}}\right)^2}{E^{(j)}_{P,m_{i(j)}}} \tag{5.68}$$

$$\tau^{(j)}_{S,m_{i(j)}} = \frac{\left(C^{(j)}_{S,m_{i(j)}}\right)^2}{E^{(j)}_{S,m_{i(j)}}} \tag{5.69}$$

onde os valores de $C^{(j)}_{P,m_{i(j)}}$ e $E^{(j)}_{P,m_{i(j)}}$ são determinados a partir de (5.64) e (5.66) fazendo $\sigma^{(j-1)} = 1$:

$$C^{(j)}_{P,m_{i(j)}} = d[m_{i(j)}] + \beta^{(j-1)} C^{(j-1)}_{m_{i(j-1)}} \tag{5.70}$$

$$E^{(j)}_{P,m_{i(j)}} = \phi[m_{i(j)}, m_{i(j)}] + 2\sum_{k=0}^{j-1} \alpha^{(j-1)}_{i(k)} \phi[m_{i(k)}, m_{i(j)}] + E^{(j-1)}_{m_{i(j-1)}} \tag{5.71}$$

[6]Os índices P e S originam-se da análise geométrica da busca conjunta, onde P e S correspondem a duas projeções do vetor-alvo, chamadas de projeção primária e secundária.

Da mesma forma $C^{(j)}_{S,m_{i(j)}}$ e $E^{(j)}_{S,m_{i(j)}}$ são obtidos fazendo $\sigma^{(j-1)} = -1$ nas equações (5.64) e (5.66):

$$C^{(j)}_{S,m_{i(j)}} = d[m_{i(j)}] - \beta^{(j-1)} C^{(j-1)}_{m_{i(j-1)}} \tag{5.72}$$

$$E^{(j)}_{S,m_{i(j)}} = \phi[m_{i(j)}, m_{i(j)}] - 2\sum_{k=0}^{j-1} \alpha^{(j-1)}_{i(k)} \phi[m_{i(k)}, m_{i(j)}] + E^{(j-1)}_{m_{i(j-1)}} \tag{5.73}$$

Ao final da busca, o máximo global entre $\tau^{(j)}_{P,m_{i(j)}}$ e $\tau^{(j)}_{S,m_{i(j)}}$ determina a posição do novo pulso e o sinal de $\sigma^{(j-1)}$ a ser escolhido. Ou seja, caso $\max_{m_{i(j)}}\{\tau^{(j)}_{P,m_{i(j)}}\} > \max_{m_{i(j)}}\{\tau^{(j)}_{S,m_{i(j)}}\}$ o sinal é positivo, caso contrário é negativo.

Após a determinação do novo pulso é feita uma verificação com o objetivo de manter o vetor-código parcial com um ganho correspondente positivo. Para isso, como exposto na Seção 5.2.2, basta analisar o sinal da correlação $C^{(j)}_{m_{i(j)}}$, ou seja:

$$\beta^{(j)} = \text{sign}\left(C^{(j)}_{m_{i(j)}}\right) \tag{5.74}$$

Todas as amplitudes encontradas até o momento, incluindo a do novo pulso, são então reajustadas pelo fator $\beta^{(j)}$ (equação (5.60)). Caso a inovação não esteja completa o algoritmo inicia então a busca por um outro pulso.

A Figura 5.15 apresenta o pseudo-código referente à uma implementação do algoritmo de busca conjunta para o dicionário algébrico do codificador G.729, representado pela Tabela 5.2. Cada iteração do laço maior corresponde à busca de um pulso. Para fazer a busca apenas nas posições restantes é utilizado um vetor chamado $\varphi_{\texttt{OCCUPIED}}$, que possui um elemento para cada trilha, sendo atribuído o valor `FALSE` para as trilhas livres e `TRUE` para trilhas que já possuem pulso. Para cada posição `mj` restante são calculados recursivamente os valores de $C^{(j)}_{P,m_{i(j)}}$ e $E^{(j)}_{P,m_{i(j)}}$, e $C^{(j)}_{S,m_{i(j)}}$ e $E^{(j)}_{S,m_{i(j)}}$, que são representados respectivamente pelas variáveis $\texttt{C}_\texttt{P}$ e $\texttt{E}_\texttt{P}$, e $\texttt{C}_\texttt{S}$ e $\texttt{E}_\texttt{S}$. Após a determinação do pulso, o algoritmo ainda faz algumas tarefas necessárias antes de passar à busca de um outro pulso. As novas amplitudes são calculadas de acordo com a equação (5.60), e as trilhas devidas são marcadas como ocupadas, de acordo com o dicionário adotado. Além disso, é feita a atualização dos valores `E0` e `C0`, utilizados para o cálculo recursivo das novas correlações e energias. Ao final de todo o processo, as amplitudes e posições dos pulsos encontrados estarão armazenadas na ordem cronológica respectivamente nos vetores `alpha(k)` e `m(k)`, `k` = 0, 1, 2, 3.

```
For (j = 1, j < 4, j += 1) {
    Csq_max = 0; E_max = 1; σ = 0;
    /* Para cada trilha */
    For (φ = 0, φ < 5, φ += 1) {
        /* Verifica se a trilha está livre */
        If (φ_OCCUPIED(φ) == FALSE) {
            /* Testa cada posição da trilha */
            For (mj = φ, mj < 40, mj += STEP) {
                dE = 2∑_{k=0}^{j-1} α(k)*φ(m(k),mj);
                E_P = E0 + φ(mj,mj)+dE;
                E_S = E0 + φ(mj,mj)-dE;
                C_P = d(mj) + C0;
                C_P = d(mj) - C0;
                Csq_P = C_P*C_P;
                If (Csq_P*E_max > Csq_max*E_P) {
                    Csq_max = Csq_P; E_max = E_P; mj_max = mj; φ_max = φ; β = sign(C_P); σ = +1;
                }
                Csq_S = C_S*C_S;
                If (Csq_S*E_max > Csq_max*E_S) {
                    Csq_max = Csq_S; E_max = E_S; mj_max = mj; φ_max = φ; β = sign(C_S); σ = -1;
                }
            }
        }
    }
    E0 = E_max;
    C0 = β*(d(mj_max)+σ*C0);
    /* Registra a nova posição e atualiza as amplitudes */
    m(j) = mj_max;
    α(j) = β;
    For (k = 0, k < j, k += 1) {
        α(k) = σ*β*α(k)
    }
    /* Marca a trilha como ocupada */
    If (φ_max < 3) {
        φ_OCCUPIED(φ_max) = TRUE;
    }
    else {
        φ_OCCUPIED(3) = TRUE;
        φ_OCCUPIED(4) = TRUE;
    }
}
```

Figura 5.15: Pseudo-código referente à uma implementação do algoritmo de busca conjunta de posição e amplitude para o codificador G.729, que possui o dicionário algébrico apresentado na Tabela 5.2 (Adaptado de [31]).

Deve-se ressaltar ainda que o pseudo-código apresentado na Figura 5.15 não inclui a busca do primeiro pulso, que corresponde à uma versão degenerada do laço apresentado.

5.9.2 Interpretação Geométrica da Busca Conjunta

Nessa seção procura-se apresentar a fundamentação geométrica do algoritmo de busca conjunta de posição e amplitude. Nos desenvolvimentos apresentados a seguir foi seguida a linha adotada em [33].

Como demonstrado na Seção 4.8.2, a solução do problema da análise-por-síntese corresponde a determinar qual dos vetores-código $\mathbf{c}_i$, $i = 0, 1, \ldots, M-1$ produz a maior projeção do vetor-alvo $\mathbf{u}$ no vetor-código filtrado correspondente $\mathbf{q}_i$.

De acordo com a equação (4.39) o vetor-código filtrado é dado pela multiplicação da matriz de resposta impulsiva $\mathbf{H}$ pelo vetor-código $\mathbf{c}_i$. Para vetores-código esparsos, com pulsos nas posições $m_{i(k)}$, $k = 0, 1, \ldots, N-1$, a equação (4.39) pode ser reescrita da seguinte maneira:

$$\mathbf{q}_i = \sum_{k=0}^{N-1} \alpha_k \mathbf{H}(:, m_{i(k)}), \tag{5.75}$$

onde $\mathbf{H}(:, m_{i(k)})$ representa a coluna $m_{i(k)}$ da matriz de resposta impulsiva $\mathbf{H}$.

A equação (5.75) mostra que $\mathbf{q}_i$ na verdade é composto de N respostas impulsivas deslocadas, que quando somadas definem uma direção que deverá se aproximar o máximo possível da direção do vetor-alvo. A idéia da busca conjunta é justamente escolher esses deslocamentos um após o outro, procurando no processo maximizar as projeções parciais do vetor-alvo no vetor-código filtrado.

Seja $\mathbf{q}^{(j-1)}_{m_{i(j-1)}}$ o vetor-código filtrado na etapa $j-1$:

$$\mathbf{q}^{(j-1)}_{m_{i(j-1)}} = \sum_{k=0}^{j-1} \alpha^{(j-1)}_{i(k)} \mathbf{H}(:, m_{i(k)}), \tag{5.76}$$

onde $m_{i(j-1)}$ é utilizado como índice em $\mathbf{q}^{(j)}_{m_{i(j-1)}}$ para enfatizar a busca pulso a pulso, como foi feito na seção anterior.

O vetor $\mathbf{q}^{(j-1)}_{m_{i(j-1)}}$ representa uma direção parcial de projeção do vetor-alvo, que com a inclusão de um novo pulso será alterada. Como a amplitude do novo pulso é ± 1, existem 4 possibilidades para o novo vetor-código filtrado:

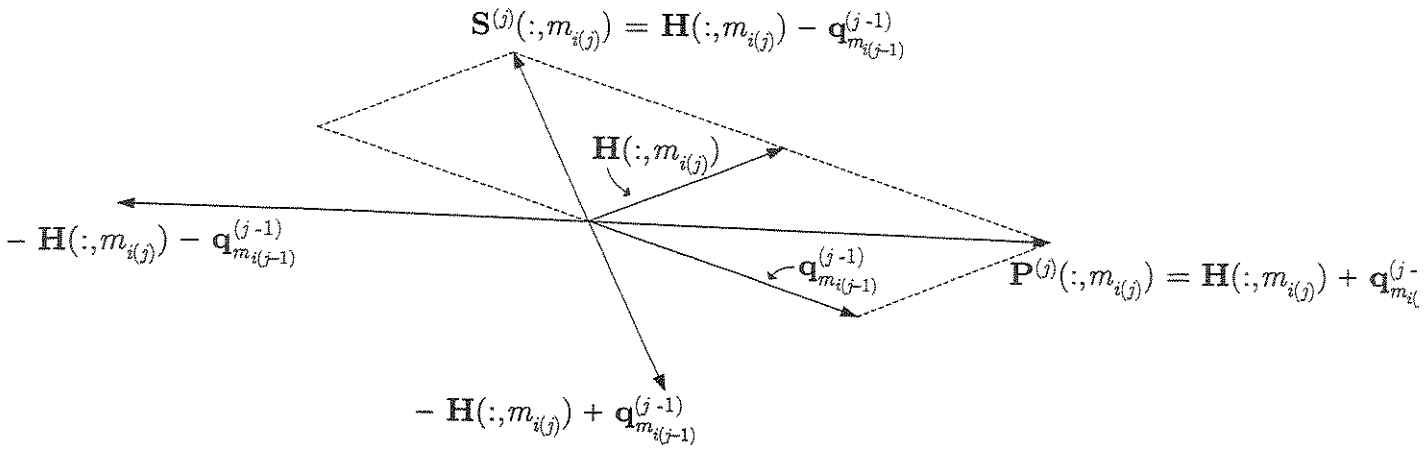


Figura 5.16: As duas direções possíveis para $\mathbf{q}^{(j)}_{m_{i(j)}}$ com a inclusão do pulso j.

$$\mathbf{q}^{(j)}_{m_{i(j)}} = \pm(\pm\mathbf{H}(:,m_{i(j)}) + \mathbf{q}^{(j-1)}_{m_{i(j-1)}}) \tag{5.77}$$

Essas quatro possibilidades, entretanto, definem na verdade apenas duas direções, como mostra a Figura 5.16. Na busca conjunta essas duas direções são representadas por[7]:

$$\mathbf{P}^{(j)}(:,m_{i(j)}) = \mathbf{H}(:,m_{i(j)}) + \mathbf{q}^{(j-1)}_{m_{i(j-1)}}, \tag{5.78}$$

que é chamada de direção de projeção parcial primária, e

$$\mathbf{S}^{(j)}(:,m_{i(j)}) = \mathbf{H}(:,m_{i(j)}) - \mathbf{q}^{(j-1)}_{m_{i(j-1)}}, \tag{5.79}$$

que é a direção de projeção parcial secundária.

Os termos $\mathbf{P}^{(j)}(:,m_{i(j)})$ e $\mathbf{S}^{(j)}(:,m_{i(j)})$ se referem às matrizes $\mathbf{P}^{(j)}$ e $\mathbf{S}^{(j)}$, que possuem como colunas respectivamente todas direções de projeção parcial primárias e secundárias na iteração j.

Seja $\mathbf{f}^{(j)}_{m_{i(j)}}$ a projeção escolhida na etapa j e relativa à posição $m_{i(j)}$:

$$\mathbf{f}^{(j)}_{m_{i(j)}} = \mathbf{H}(:,m_{i(j)}) + \sigma^{(j-1)}\mathbf{q}^{(j-1)}_{m_{i(j-1)}}, \tag{5.80}$$

onde o valor $\sigma^{(j-1)}$ nesse caso representa a direção de projeção parcial escolhida. Ou seja,

[7]A escolha desses sentidos para representar as direções de projeção está associada com o reajuste das amplitudes dos pulsos já encontrados durante a busca de um novo pulso.

$\sigma^{(j-1)} = 1$ corresponde a direção primária, e $\sigma^{(j-1)} = -1$ corresponde à direção secundária.

O novo vetor código filtrado $\mathbf{q}^{(j)}_{m_{i(j)}}$ é dado por:

$$\mathbf{q}^{(j)}_{m_{i(j)}} = \beta^{(j)}\mathbf{f}^{(j)}_{m_{i(j)}}, \tag{5.81}$$

onde $\beta^{(j)}$ é um fator utilizado para manter positivo o ganho da projeção do vetor-alvo na direção escolhida, podendo assumir os valores ± 1.

A correlação entre o vetor-alvo e a direção de projeção escolhida é dada por:

$$C^{j}_{m_{i(j)}} = \mathbf{u}^T\mathbf{f}^{(j)}_{m_{i(j)}} = d[m_{i(j)}] + \sigma^{(j-1)}\beta^{(j-1)}C^{(j-1)}_{m_{i(j-1)}}, \tag{5.82}$$

o que corresponde à definição de $C^{j}_{m_{i(j)}}$ apresentada anteriormente (5.64).

Para a determinação da melhor direção de projeção parcial, é conveniente definir as seguintes matrizes:

$$\boldsymbol{\mathcal{P}}^{(j)} = (\mathbf{P}^{(j)})^T\mathbf{P}^{(j)}, \tag{5.83}$$

e

$$\boldsymbol{\mathcal{S}}^{(j)} = (\mathbf{S}^{(j)})^T\mathbf{S}^{(j)}, \tag{5.84}$$

que são as matrizes de auto-correlação primária e secundária.

As normas quadradas das projeções do vetor-alvo nas direções de projeção primária e secundária podem ser dadas então por:

$$\tau^{(j)}_{P,m_{i(j)}} = \frac{\left(\mathbf{u}^T\mathbf{P}^{(j)}(:, m_{i(j)})\right)^2}{\mathcal{P}^{(j)}(m_{i(j)}, m_{i(j)})} \tag{5.85}$$

$$\tau^{(j)}_{S,m_{i(j)}} = \frac{\left(\mathbf{u}^T\mathbf{S}^{(j)}(:, m_{i(j)})\right)^2}{\mathcal{S}^{(j)}(m_{i(j)}, m_{i(j)})} \tag{5.86}$$

Pode-se mostrar que essas definições para $\tau^{(j)}_{P,m_{i(j)}}$ e $\tau^{(j)}_{S,m_{i(j)}}$ correspondem às definições apresentadas anteriormente pelas equações (5.68) e (5.69). Através de (5.78), (5.79) e (5.81), pode-se chegar nas seguintes expressões para as correlações das projeções primária e secundária utilizadas em (5.85) e (5.86):

$$C^{(j)}_{P,m_{i(j)}} = \mathbf{u}^T \mathbf{P}^{(j)}(:, m_{i(j)}) = d[m_{i(j)}] + \beta^{(j-1)} C^{(j-1)}_{m_{i(j-1)}}, \tag{5.87}$$

$$C^{(j)}_{S,m_{i(j)}} = \mathbf{u}^T \mathbf{S}^{(j)}(:, m_{i(j)}) = d[m_{i(j)}] - \beta^{(j-1)} C^{(j-1)}_{m_{i(j-1)}}, \tag{5.88}$$

que são as mesmas apresentadas em (5.70) e (5.72).

A partir de (5.83) e (5.84), e usando ainda (5.78), (5.79) e (5.76), também pode-se mostrar, que $\mathcal{P}^{(j)}(m_{i(j)}, m_{i(j)})$ e $\mathcal{S}^{(j)}(m_{i(j)}, m_{i(j)})$ correspondem respectivamente aos valores $E^{(j)}_{P,m_{i(j)}}$ e $E^{(j)}_{S,m_{i(j)}}$ apresentados pelas equações (5.71) e (5.73) [30].

Capítulo 6

DESEMPENHO DOS ALGORITMOS DE BUSCA

Neste capítulo são apresentados e analisados alguns resultados de medidas de complexidade para os algoritmos de busca de inovações em dicionários algébricos. São analisados principalmente os algoritmos de busca apresentados no Capítulo 5, considerando-se inserções dos mesmos nos codificadores G.729 e GSM-AMR, que pertencem respectivamente aos órgãos de padronização ITU-T e ETSI. As medidas de complexidade baseiam-se em implementações dos dois codificadores citados no processador digital de sinais de ponto-fixo (DSP - *Digital Signal Processor*) TMS320C6202 da Texas Instruments^{TM}. Além de medidas de complexidade foram extraídas também medidas de qualidade de voz, sendo considerada para esse fim a medida perceptual de qualidade PSQM (*Perceptual Speech Quality Measure*). Vale ressaltar ainda que ao longo deste capítulo procura-se dar maior ênfase ao desempenho dos algoritmos de busca seqüencial dos pulsos: a busca seqüencial de posições e a busca conjunta de posições e amplitudes.

6.1 A Medida Perceptual de Qualidade de Voz

Pode-se dizer que a qualidade da voz constitui-se, juntamente com a complexidade computacional e a taxa de compressão, como uma das características mais importantes de um codificador. As medidas mais confiáveis de qualidade de voz que se pode obter são as extraídas através de testes subjetivos, onde um grupo de pessoas julga um conjunto de sinais codificados, fazendo uma comparação com o conjunto de sinais originais. Existem

inclusive algumas medidas padronizadas de qualidade subjetiva, como por exemplo o MOS (Mean Opinion Score), que atribui uma nota de 1 (ruim) a 5 (excelente) ao conjunto de sinais analisados. Tais medidas subjetivas, entretanto, são trabalhosas e demandam bastante tempo para serem extraídas, dada a necessidade da participação um grupo relativamente grande de pessoas analisando os sinais um a um.

As dificuldades de implementação de experimentos subjetivos levaram ao desenvolvimento de algoritmos que simulam os testes de julgamento da qualidade de voz por ouvintes humanos, e entre esses algoritmos um dos mais usados e conhecidos é o chamado PSQM [18], medida perceptual de qualidade da voz (*Perceptual Speech Quality Measure*), que foi adotado nesse trabalho para avaliar o desempenho dos algoritmos de busca de inovações. O PSQM faz uso de um modelo psico-acústico do sistema auditivo humano para avaliar o sinal codificado em comparação com o sinal original, levando em consideração efeitos perceptivos da audição. Ao final do processo de avaliação, o PSQM atribui uma nota ao sinal codificado que varia de 0 (melhor caso) a 6 (pior caso).

Em geral o processo de codificação introduz um atraso no sinal de voz, e tal atraso deve ser considerado para a correta aplicação do PSQM. O método adotado aqui para o alinhamento temporal foi a simples repetição do cálculo do PSQM para um intervalo considerável em torno do atraso esperado (geralmente igual à um sub-bloco do codificador). O valor final para o PSQM corresponde então ao melhor valor encontrado durante a análise do intervalo de atrasos considerado. Vale ressaltar ainda que para as medidas de PSQM os sinais foram escalonados para -26dBov, como especificado em [15] e [18].

6.2 A Base de Dados NTIMIT

A base de dados NTIMIT [19] é constituída por um conjunto de arquivos de sinais de voz, gravados após transmissão em linhas telefônicas reais. A base é dividida em 8 grupos (DR1-DR8), onde cada grupo corresponde a uma região de gravação diferente.

Todos os arquivos estão gravados com taxa de amostragem de 16kHz e quantização de 16 bits por amostra. Como os codificadores de voz considerados trabalham com sinais de 8kHz, toda a base de dados foi filtrada e dizimada para essa freqüência. A filtragem foi realizada com um filtro FIR passa-baixas de 97 coeficientes projetado no software MATLAB^{TM}. Tal filtro possui uma freqüência de corte em torno de 3,4Hz, e 64dB de atenuação em 4kHz. Após essa filtragem, verifica-se que a menor relação sinal ruído entre o sinal original e o sinal

Tabela 6.1: Principais características dos codificadores utilizados do ponto de vista da busca no dicionário fixo.

Codificador	Blocos	Sub-blocos	Número de pulsos	Algoritmo de busca
G.729 (8kbps)	10ms	5ms	4	Focalizada ($\lambda = 0$)
G.729A (8kbps)	10ms	5ms	4	DFTS ($\lambda = 0$)
GSM-AMR 12.2kbps (GSM-EFR)	20ms	5ms	10	DFTS ($\lambda = 0.5$)
GSM-AMR 10.2kbps	20ms	5ms	8	DFTS ($\lambda = 0.5$)
GSM-AMR 7.95kbps	20ms	5ms	4	DFTS ($\lambda = 0$)
GSM-AMR 7.40kbps (IS-641)	20ms	5ms		
GSM-AMR 6.70kbps	20ms	5ms	3	DFTS ($\lambda = 0$)
GSM-AMR 5.90kbps	20ms	5ms	2	Exaustiva ($\lambda = 0$)
GSM-AMR 5.15kbps	20ms	5ms	2	Exaustiva ($\lambda = 0$)
GSM-AMR 4.75kbps	20ms	5ms		

filtrado é de cerca de 30dB, o que confirma que a energia do sinal realmente está armazenada na faixa de freqüências do canal telefônico.

6.3 Os Codificadores Utilizados

Para os testes dos algoritmos de busca foram utilizados dois padrões de codificação ACELP, o G.729 [17] e o GSM-AMR [10] (*Adaptive Multi-Rate*), que pertencem respectivamente à ITU-T e ao ETSI. É importante ressaltar que o codificador G.729 trabalha na taxa de transmissão fixa de 8kbps, enquanto que o GSM-AMR pode operar em 8 taxas diferentes, que variam de 4.75 a 12.2kbps. Nas taxas de 7.40 e 12.2 kbps, o GSM-AMR é idêntico respectivamente ao IS-641 [12] do TIA/EIA , e ao GSM-EFR [9] do ETSI.

A Tabela 6.1 mostra as principais características dos codificadores G.729 e GSM-AMR, no que diz respeito à busca no dicionário fixo. Pode-se notar que também está incluído na Tabela 6.1 o codificador G.729A, uma versão mais rápida do G.729.

Como mostra a Tabela 6.1, todos os codificadores considerados possuem sub-blocos de mesmo comprimento (5ms, ou 40 amostras). No G.729 é utilizado um algoritmo de busca focalizada, enquanto que na sua versão mais eficiente (G.729A) é utilizado um algoritmo DFTS. Em geral, o GSM-AMR adota um algoritmo de busca e um dicionário algébrico diferente para cada taxa de transmissão. Nas taxas mais altas existem mais pulsos, e são

usados algoritmos do tipo DFTS, enquanto que nas taxas baixas existem poucos pulsos, sendo adotadas buscas exaustivas de posições. Vale ressaltar ainda que as taxas de 7.40 e 7.95kbps são equivalentes do ponto de vista da busca no dicionário fixo, já que para ambas são utilizados o mesmo dicionário algébrico e o mesmo algoritmo de busca. O mesmo também ocorre para as taxas de 4.75 e 5.15kbps.

A Tabela 6.1 mostra ainda o fator λ utilizado por cada codificador para fazer a pré-associação de amplitudes às posições. Em geral é utilizado $\lambda = 0$, com exceção dos casos das taxas de 12.2 e 10.2kbps do GSM-AMR, onde se define $\lambda = 1/2$.

O Apêndice A apresenta maiores detalhes sobre os codificadores G.729 e GSM-AMR, dando ênfase aos algoritmos de busca e aos dicionários algébricos utilizados pelos codificadores.

6.4 Implementações no Codificador G.729

Nessa seção são apresentados resultados de complexidade computacional e qualidade de voz para os diferentes tipos de algoritmos de busca apresentados no Capítulo 5.

As medidas foram extraídas através da implementação, no codificador G.729, dos seguintes algoritmos de busca:

- Um algoritmo de busca seqüencial de posições implementado de acordo com o apresentado na Seção 5.8, e semelhante ao pseudo-código da Figura 5.13;
- Um algoritmo de busca conjunta de posição e amplitude implementado de acordo com o apresentado na Seção 5.9, e semelhante ao pseudo-código da Figura 5.15;
- O algoritmo DFTS do codificador G.729A (descrito na Seção A.1.1);
- O algoritmo de busca focalizada nativo do G.729 (descrito na Seção A.1);
- Um algoritmo de busca exaustiva de posições, implementado de acordo com o apresentado na Seção 5.5, e semelhante ao pseudo-código da Figura 5.7;

Para as buscas focalizada, exaustiva e seqüencial de posições, assim como para o algoritmo DFTS, na pré-associação de amplitudes às posições é adotado o fator $\lambda = 0$, como especificado no padrão G.729. Vale lembrar que o algoritmo JPAS dispensa tal processo, já que nesse caso é executada conjuntamente a busca pelas amplitudes e posições dos pulsos.

6.4.1 Análises de Qualidade de Voz

A Tabela 6.2 mostra os resultados de uma análise PSQM dos algoritmos inseridos no codificador G.729, onde foram considerados todos os 1680 arquivos de voz da partição de teste da base NTIMIT[1].

Pode-se notar na Tabela 6.2 que as medidas perceptuais de qualidade de voz indicam que os dois algoritmos que buscam os pulsos seqüencialmente, a busca seqüencial de posições e a busca conjunta de posição e amplitude, apresentam apenas uma pequena degradação na qualidade de voz quando comparadas com os algoritmos padronizados de busca focalizada e DFTS. A substituição do algoritmo de busca focalizada do codificador por um processo de busca seqüencial de posições resultaria em um aumento de apenas 0.08 na medida de PSQM. No caso da busca conjunta essa diferença seria ainda menor, cerca de 0.06.

Tabela 6.2: Desempenho dos algoritmos de busca sobre a base de dados NTIMIT.

Algoritmo	PSQM
Busca exaustiva de posições	1.974
Busca focalizada	1.991
Busca em árvore por profundidade	2.013
Busca conjunta de posição e amplitude	2.054
Busca seqüencial de posições	2.070

A Tabela 6.3 mostra a quantidade de vezes em que foram escolhidas 0, 1, 2, 3 ou 4 posições corretas para os pulsos durante uma execução da busca, numa comparação com o resultado obtido pela busca ótima. As medidas foram extraídas da codificação de 16 arquivos da base NTIMIT (2 de cada região), o que corresponde a aproximadamente 10000 execuções dos algoritmos. A coluna mais à direita da Tabela 6.3 mostra ainda a porcentagem total de acerto da posição dos pulsos (novamente em comparação com a busca ótima).

Pode-se notar que a relação entre o desempenho dos algoritmos no que diz respeito à quantidade de posições corretas encontradas é aproximadamente a mesma relação obtida pelas medidas de PSQM. A busca exaustiva de posições apresenta o melhor resultado, aparecendo em seguida respectivamente as buscas focalizada e DFTS. Os algoritmos que implementam buscas seqüenciais dos pulsos aparecem em último aproximadamente empatados.

[1] A base NTIMIT é dividida em duas grandes partições, chamadas de partição de teste e partição de treinamento.

Tabela 6.3: Porcentagem de posições corretas dos pulsos (em comparação com a busca ótima).

Busca	Número de posições corretas					Total
	0	1	2	3	4	
Exaustiva	0.46%	1.20%	1.89%	1.22%	95.22%	97.38%
Focalizada	1.01%	2.87%	3.78%	2.39%	89.94%	94.35%
DFTS	2.32%	6.55%	9.61%	9.38%	72.13%	85.61%
JPAS	4.46%	13.12%	23.16%	13.10%	46.15%	70.84%
Seqüencial	4.48%	12.91%	23.23%	14.44%	44.93%	70.61%

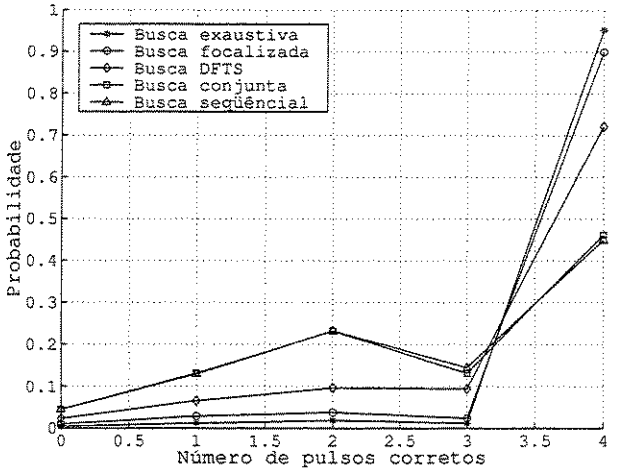


Figura 6.1: Curva de distribuição de densidade de probabilidade do número de posições corretas para cada execução dos algoritmos de busca no dicionário algébrico do G.729.

Os dados da Tabela 6.3 também estão representados em forma de gráfico na Figura 6.1, cujas curvas descrevem um esboço das distribuições de probabilidade do número de acertos das posições a cada execução dos algoritmos.

Tanto na Tabela 6.3 quanto na Tabela 6.2 fica claro que, para o dicionário algébrico do G.729, o algoritmo JPAS apresenta resultados ligeiramente melhores do que os obtidos pela simples busca seqüencial de posições.

6.4.2 Análises de Complexidade Computacional

Além das medidas de qualidade de voz, foram extraídas também medidas da complexidade computacional dos algoritmos inseridos no codificador G.729. Para analisar o esforço computacional demandado por cada algoritmo, foi feita a implementação do codificador G.729 no processador digital de sinais de ponto-fixo TMS320C6202 [37] da Texas Instruments^{TM}, e os resultados foram medidos em termos de número de ciclos gastos por cada algoritmo. As implementações seguiram uma aritmética de 16 bits, e foram feitas na linguagem de programação C. Além disso, embora o processador considerado suporte um alto nível de paralelismo, a execução em paralelo de instruções foi desabilitada durante os testes para reduzir ao máximo a influência de especificidades de hardware nos resultados. O Apêndice B apresenta maiores detalhes sobre o DSP (*Digital Signal Processor*) TMS320C6202, assim como sobre os outros processadores da família TMS320C62x.

A Tabela 6.4 mostra o número de ciclos de processamento gastos para uma execução de cada um dos cinco algoritmos de busca inseridos no codificador G.729. Os dados da Tabela 6.4 são referentes à codificação de 16 arquivos de voz da base NTIMIT (2 de cada região), num total de mais de 50 segundos de voz, o que corresponde a cerca de 10000 execuções dos algoritmos. As medidas incluem o tempo gasto na execução das seguintes tarefas:

- Pré-associação de amplitudes às posições (quando necessário);
- Cálculo prévio da matriz $\mathbf{\Phi}'$, a partir de $\mathbf{\Phi}$ (quando necessário);
- Execução do algoritmo de busca.

A matriz $\mathbf{\Phi}'$ só não é calculada previamente para a busca conjunta, que dispensa a pré-associação de amplitudes às posições, e para a busca seqüencial de posições, onde na implementação adotada os elementos necessários de $\mathbf{\Phi}'$ são calculados dentro do algoritmo de busca, conforme necessário.

A implementação dos algoritmos no DSP de ponto-fixo mostra que a pequena degradação em qualidade de voz apresentada pelos dois algoritmos de busca seqüencial dos pulsos é compensada por uma grande redução na complexidade computacional do processo de busca no dicionário fixo. Na Tabela 6.4 pode-se notar que a busca conjunta e a busca seqüencial de posições demandam menos da metade do processamento necessário para o

Tabela 6.4: Medidas de complexidade para os algoritmos de busca inseridos no G.729. As medidas de complexidade são dadas em termos do número de ciclos de processamento gastos em cada execução do processo de busca.

Algoritmo	Número de ciclos($\times 10^3$)		
	Mínimo	Médio	Máximo
Busca exaustiva de posições	768.1	768.3	768.5
Busca focalizada	63.3	144.7	186.5
Busca em árvore por profundidade	35.2	35.6	36.1
Busca conjunta de posição e amplitude	14.5	16.9	21.0
Busca seqüencial de posições	12.4	14.5	17.4

algoritmo DFTS do codificador G.729A. Além disso, ambas requerem apenas cerca de um décimo do número médio de ciclos exigido pela busca focalizada nativa do padrão G.729.

A Tabela 6.4 também mostra que a busca seqüencial de posições apresenta uma complexidade cerca de 15% menor do a que complexidade do processo de busca conjunta de posição e amplitude.

De um maneira geral, o que as tabelas 6.4 e 6.2 mostram é que existe um relação de compromisso entre a complexidade computacional do algoritmo e a qualidade da voz codificada. Ou seja, quanto menor o caráter ótimo do algoritmo, menor é o esforço computacional exigido pelo processo, e maior é a degradação no que diz respeito à qualidade da voz. Essa relação de compromisso pode ser representada por uma curva definida por pontos que correspondem às relações custo-benefício dos algoritmos. A Figura 6.2 apresenta essa curva para o caso dos algoritmos de busca considerados.

Na Figura 6.2 pode-se notar que a busca seqüencial de posições define um ponto extremo da curva de "qualidade x complexidade", representando o algoritmo de menor complexidade. Vale ressaltar ainda que as diferenças nos valores de PSQM da Figura 6.2 são pequenas, enquanto que as diferenças no que diz respeito ao número de ciclos exigidos para cada algoritmo são consideráveis.

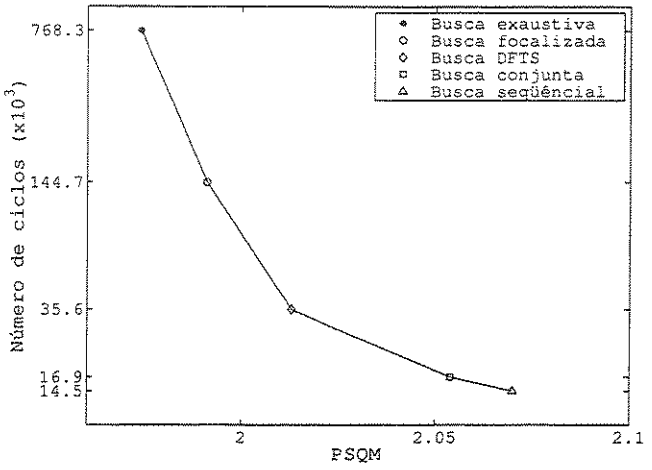


Figura 6.2: Relação entre complexidade computacional e qualidade de voz dos algoritmos de busca em dicionários algébricos. (Para melhor visualização é utilizada a escala logarítmica no eixo vertical)

6.5 Implementações no Codificador GSM-AMR

Como apresentado na Seção 6.3, o codificador GSM-AMR funciona em 8 diferentes taxas de transmissão, que variam de 4.75 a 12.2kbps. As taxas de 5.90, 6.70, 10.2 e 12.2kbps possuem cada qual um dicionário algébrico e um algoritmo de busca específico. Já as taxas de 4.57 e 5.14kbps compartilham o mesmo dicionário e o mesmo algoritmo de busca, o que também ocorre com as taxas de 7.40 e 7.95kbps. Portanto, existem no total seis dicionários algébricos diferentes, cada qual com seu algoritmo de busca associado. Esses algoritmos estão representados na Tabela 6.1 e são descritos em detalhes na Seção A.2.

Os testes com o codificador GSM-AMR tiveram como objetivo principal verificar o desempenho dos algoritmos de busca seqüencial dos pulsos em uma grande variedade de dicionários algébricos. Para isso foram feitas implementações do algoritmo JPAS e do algoritmo de busca seqüencial de posições para cada um dos dicionários algébricos do GSM-AMR. Portanto, contando com os processos de busca originais do GSM-AMR, foram analisados um total de 18 algoritmos:

- Os seis algoritmos de busca originais do codificador GSM-AMR;
- Seis algoritmos de busca seqüencial de posições, implementados para cada um dos seis dicionários algébricos do GSM-AMR;

- Seis algoritmos de busca de conjunta de posição e amplitude, implementados para cada um dos seis dicionários algébricos do GSM-AMR.

Em todas as implementações tanto o algoritmo JPAS como a busca seqüencial de posições seguem as seguintes etapas:

1. A busca da posição (ou posição e amplitude) de um pulso é feita nas posições das trilhas livres;

2. Marca-se ou não a(s) trilha(s) devida(s) como ocupada(s), de acordo com a estrutura do dicionário e com os pulsos já encontrados.

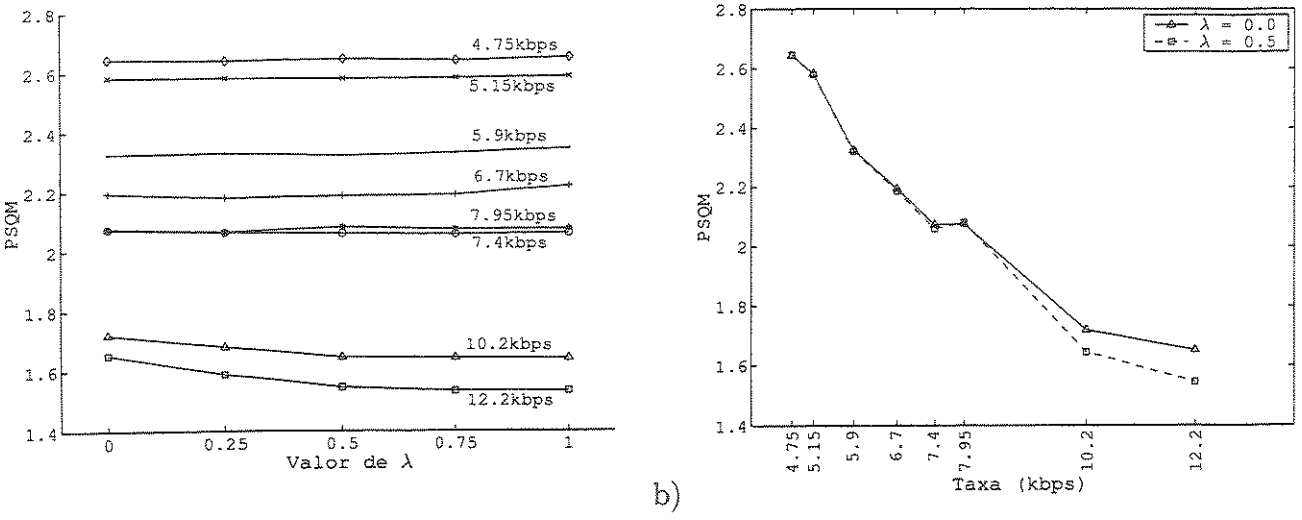
A etapa 1 é comum à todas as implementações, enquanto que a etapa 2 é a responsável pela adequação dos algoritmos às diferentes estruturas dos dicionários. Nessa etapa deve-se levar em consideração que para alguns dicionários uma trilha pode possuir mais de um pulso, e que em outros dicionários duas ou mais trilhas abrigam um único pulso. Dessa maneira, nas taxas de 10.2 e 12.2kbps, por exemplo, cada trilha possui 2 pulsos; devendo uma trilha ser marcada como ocupada somente após a determinação de seu segundo pulso. Nas taxas de 7.40 e 7.95kbps, por outro lado, a determinação de um pulso nas trilhas 3 ou 4 já as torna ocupadas.

Vale ressaltar ainda que nas taxas de 4.75 e 5.15kbps assume-se que a definição do primeiro pulso determina o subconjunto a ser escolhido[2].

6.5.1 Considerações sobre a Pré-Associação de Amplitudes às Posiçõe

Como mostra a Seção 5.4, a técnica de pré-associação de amplitudes às posições é utilizada para simplificar o processo de busca, de forma a reduzi-lo apenas à busca das posições dos pulsos. No processo, a associação de amplitudes é feita com base na quantização de um vetor de estimação de amplitudes $\mathbf{b}$, que de acordo com a equação (5.37), é composto pela soma dos valores normalizados do vetor-alvo filtrado regressivamente $\mathbf{d}$ e do resíduo LTP $\mathbf{u}'$, havendo ainda uma ponderação controlada pelo fator de peso λ. Quanto maior o valor de λ, maior a influência do resíduo LTP na composição do vetor de estimação de amplitudes.

[2]O dicionário adotado nessas taxas considera a divisão das trilhas em subconjuntos, como mostra a Tabela A.9.



a) b)

Figura 6.3: Influência do fator λ, que controla a composição do vetor de estimação de amplitudes.

O valor de λ a ser utilizado é uma característica do codificador, e deve ser escolhido de acordo com o dicionário algébrico adotado. No caso do codificador GSM-AMR, por exemplo, a adoção de diferentes valores para λ nos diferentes algoritmos de busca utilizados, produz efeitos consideráveis na qualidade da voz codificada. A Figura 6.3(a) mostra medidas de PSQM obtidas para os algoritmos de busca originais do GSM-AMR, com a pré-associação de amplitudes às posições feita com o fator λ igual a 0, 0.25, 0.5, 0.75 e 1. As medidas foram extraídas da análise de 16 arquivos da base NTIMIT, 2 de cada região.

Na Figura 6.3(a) pode-se notar que a utilização de valores não nulos para λ não representa nenhuma vantagem para as taxas de transmissão de 4.75 a 7.95kbps, casos onde os dicionários algébricos possuem de 2 a 4 pulsos por vetor-código, ou seja, uma esparsidade de 5 a 10%. Para as taxas de 10.2 e 12.2kbps, onde a esparsidade é de mais de 20%, a utilização de valores maiores para λ representa de fato um ganho no que diz respeito à qualidade da voz. Pode-se observar na Figura 6.3(a) que nesses casos a qualidade da voz aumenta juntamente com o aumento do valor do fator λ até o ponto onde $\lambda = 1/2$, havendo a partir daí uma certa estabilidade na curva de qualidade de voz.

A Tabela 6.5 mostra em particular as medidas de PSQM obtidas para os casos onde $\lambda = 0$ e $\lambda = 1/2$. Os mesmos valores também estão representados em forma de gráfico na Figura 6.3(b), onde fica claro o ganho obtido no que diz respeito à qualidade de voz com a utilização de $\lambda = 1/2$ para as taxas de 10.2 e 12.2kbps. Como mostra a Tabela 6.1,

Tabela 6.5: Medidas de PSQM para o codificador GSM-AMR com $\lambda = 0$ e $\lambda = 1/2$.

Taxa	PSQM	
(kbps)	$\lambda = 0$	$\lambda = 1/2$
4.75	2.644	2.646
5.15	2.583	2.580
5.90	2.325	2.321
6.70	2.194	2.186
7.40	2.073	2.060
7.95	2.075	2.081
10.20	1.720	1.645
12.20	1.652	1.545

apresentada na Seção 6.3, o codificador GSM-AMR adota justamente o fator $\lambda = 0$ para as taxas mais baixas, de 4.75 a 7.95kbps, e o fator $\lambda = 1/2$ para as taxas mais altas, de 10.2 e 12.2kbps.

6.5.2 Análises de Qualidade de Voz

A Tabela 6.6 mostra resultados de medidas perceptuais objetivas obtidas de uma análise PSQM feita sobre os 110 arquivos de voz da região DR1 da base NTIMIT. Para a busca seqüencial de posições foi adotado um esquema de pré-associação de amplitudes às posições semelhante ao dos algoritmos originais do GSM-AMR, ou seja, feito com o fator $\lambda = 0$ para as taxas de 4.75 a 7.95kbps, e com o fator $\lambda = 1/2$ para as taxas de 10.2 e 12.2kbps.

A partir dos dados da Tabela 6.6, pode-se notar que a busca seqüencial de posições apresenta valores de PSQM ligeiramente superiores aos obtidos pelos algoritmos de busca originais do GSM-AMR. As diferenças de PSQM entre esses algoritmos aumentam com a taxa de transmissão, variando de 0.04 a 0.12.

A busca seqüencial de posições e a busca conjunta de posição e amplitude apresentam resultados similares paras as taxas mais baixas, de 4.75 a 6.70kbps. Já nas taxas de 7.40 e 7.95kbps o algoritmo JPAS apresenta um desempenho um pouco melhor. Vale ressaltar que nessas taxas é utilizado um dicionário algébrico idêntico ao adotado pelo codificador G.729, e que portanto esses resultados confirmam os resultados apresentados anteriormente na Tabela 6.2, obtidos a partir das implementações feitas no G.729.

Ainda na Tabela 6.6, pode-se notar que nas taxas mais altas, de 10.2 e 12.2kbps,

Tabela 6.6: Desempenho dos algoritmos de busca inseridos no GSM-AMR numa análise de PSQM sobre a base NTIMIT (região DR1).

Taxa (kbps)	PSQM		
	Busca padronizada do GSM-AMR	Busca seqüencial de posições	Busca conjunta
4.75	2.531	2.572	2.578
5.15	2.497	2.524	2.525
5.90	2.276	2.321	2.321
6.70	2.126	2.184	2.182
7.40	1.998	2.066	2.059
7.95	2.009	2.074	2.064
10.20	1.549	1.656	1.730
12.20	1.481	1.602	1.747

Tabela 6.7: Medidas de PSQM da análise da base NTIMIT para os algoritmos de busca na taxa de 12.2kbps.

Algoritmo	PSQM
Busca padronizada do GSM-AMR (12.2kbps)	1.494
Busca seqüencial de posições ($\lambda = 1/2$)	1.619
Busca seqüencial de posições ($\lambda = 0$)	1.685
Busca conjunta de posição e amplitude	1.754

a busca seqüencial de posições apresenta resultados consideravelmente melhores do que os obtidos pelo algoritmo JPAS. Esse fato também pode ser verificado na Figura 6.4, que mostra em forma de gráfico as medidas de PSQM da Tabela 6.6. Esses resultados indicam que em dicionários algébricos com muitos pulsos, e onde existem mais de um pulso por trilha, a execução de uma busca seqüencial apenas das posições apresenta resultados melhores do que a execução de uma busca conjunta de posição e amplitude.

A Tabela 6.7 mostra outras medidas de PSQM para os algoritmos de busca inseridos no GSM-AMR. Os dados foram obtidos através da análise da base NTIMIT completa, e se referem apenas a taxa de 12.2kbps, onde o GSM-AMR é idêntico ao codificador GSM-EFR. São analisadas duas versões da busca seqüencial de posições, uma com o fator $\lambda = 0$, e outra com o fator $\lambda = 1/2$. De acordo com a Tabela 6.7, na taxa de 12.2kbps a utilização do

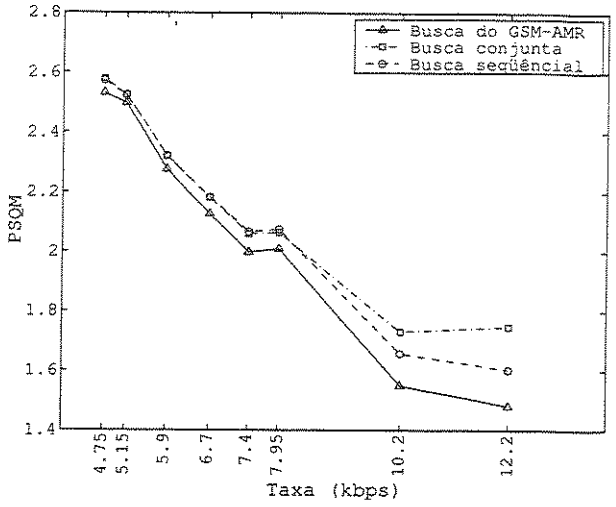


Figura 6.4: Medidas de PSQM dos algoritmos inseridos no codificador GSM-AMR. Os dados foram obtidos através da análise dos arquivos de voz da região DR1 da base NTIMIT.

fator $\lambda = 1/2$ para a busca seqüencial de posições produz uma melhoria na qualidade de voz. Na verdade, esse resultado é esperado, já que o mesmo ocorre para a busca padronizada do GSM-AMR, como foi mostrado na Seção 6.5.1. A Tabela 6.7 também mostra que a busca seqüencial de posições apresenta resultados melhores do que os obtidos pela busca conjunta mesmo quando a pré-associação de amplitudes às posições é feita com o fator $\lambda = 0$.

6.5.3 Análises de Complexidade Computacional

Para analisar o esforço computacional requerido pelos algoritmos inseridos no codificador GSM-AMR, foi utilizado novamente o processador digital de sinais de ponto-fixo TMS320C6202 [37] da Texas Instruments^{TM}. O codificador GSM-AMR foi implementado no DSP, e os resultados foram medidos em termos de número de ciclos gastos por cada algoritmo. As implementações seguiram uma aritmética de 16 bits, e foram feitas na linguagem de programação C. Como feito para os testes com o G.729, a execução em paralelo de instruções no DSP foi desabilitada durante os testes, para reduzir ao máximo a influência de especificidades de hardware nos resultados.

A Tabela 6.8 mostra o número de ciclos de processamento gastos para uma execução dos algoritmos de busca em cada taxa de transmissão. Os dados da Tabela 6.8 são referentes à codificação de 16 arquivos de voz da base NTIMIT (2 de cada região), num total de cerca de 50 segundos de voz, o que corresponde a cerca de 10000 execuções dos algoritmos.

Tabela 6.8: Medidas de complexidade para os algoritmos de busca inseridos no GSM-AMR. As medidas de complexidade são dadas em termos do número de ciclos de processamento gastos em cada execução do processo de busca.

Algoritmo	Número de ciclos ($\times 10^3$)		
4.75 e 5.15kbps	Mínimo	Médio	Máximo
Busca original do GSM-AMR (Exaustiva)	19.0	19.3	19.5
Busca conjunta de posição e amplitude	3.4	3.5	3.6
Busca seqüencial de posições	3.2	3.5	3.6
5.90kbps	Mínimo	Médio	Máximo
Busca original do GSM-AMR (Exaustiva)	45.5	46.0	46.3
Busca conjunta de posição e amplitude	5.6	7.0	10.1
Busca seqüencial de posições	5.2	6.0	8.3
6.70kbps	Mínimo	Médio	Máximo
Busca original do GSM-AMR (DFTS)	95.9	96.9	97.5
Busca conjunta de posição e amplitude	9.0	10.1	12.4
Busca seqüencial de posições	7.5	8.2	10.0
7.40 (IS-641) e 7.95kbps	Mínimo	Médio	Máximo
Busca original do GSM-AMR (DFTS)	72.6	73.4	73.7
Busca conjunta de posição e amplitude	13.4	15.0	18.9
Busca seqüencial de posições	10.6	12.2	14.8
10.2kbps	Mínimo	Médio	Máximo
Busca original do GSM-AMR (DFTS)	80.8	81.3	81.5
Busca conjunta de posição e amplitude	43.4	54.7	64.5
Busca seqüencial de posições	39.8	47.0	54.9
12.2kbps (GSM-EFR)	Mínimo	Médio	Máximo
Busca original do GSM-AMR (DFTS)	104.6	105.4	105.8
Busca conjunta de posição e amplitude	61.1	81.9	92.4
Busca seqüencial de posições	54.5	62.2	77.9

As medidas apresentadas na Tabela 6.8 incluem o tempo gasto na execução das seguintes tarefas:

- Pré-associação de amplitudes às posições (para a busca seqüencial e para os algoritmos originais do GSM-AMR);
- Cálculo prévio da matriz Φ' a partir de Φ (para os algoritmos originais do GSM-AMR);
- Execução do algoritmo de busca.

Vale ressaltar que o cálculo prévio da matriz Φ' é feito apenas para os algoritmos originais do GSM-AMR, porque o algoritmo JPAS dispensa a pré-associação de amplitudes às posições, e porque a busca a seqüencial de posições é implementada de forma a calcular os elementos de Φ' dentro do próprio algoritmo de busca.

Os dados da Tabela 6.8 mostram que tanto a busca conjunta quanto a busca seqüencial de posições requerem um esforço computacional bem menor do que o requerido pelos procedimentos de busca padronizados do GSM-AMR. Nas taxas de 4.75 a 6.70kbps, por exemplo, o tempo gasto para a execução desses algoritmos é inferior a 20% do tempo gasto para a execução dos algoritmos de busca originais do GSM-AMR. Nas taxas de 7.40 e 7.95kbps, a busca seqüencial de posições e a busca conjunta requerem respectivamente apenas cerca 16% e 20% do número de ciclos demandados pelo algoritmo DFTS padrão do GSM-AMR, enquanto que nas taxas de 10.2 e 12.2kbps esses porcentagens são de cerca de 59% e 78%. É interessante também analisar as reduções de complexidade em termos de ciclos executados por segundo. Para 12.2kbps, por exemplo, onde o codificador corresponde ao GSM-EFR, a busca seqüencial de posições e a busca conjunta economizam respectivamente cerca de 23 e 43 mil ciclos por execução. Como a busca é executada a cada 5ms, essas reduções de complexidade representariam uma economia de 4.6 e 8.6 MCPS (Milhões de Ciclos Por Segundo) respectivamente. Na taxa de 6.70kbps, onde ocorre o maior ganho em valor absoluto, a economia é de mais de 17 MCPS, tanto para a busca conjunta quanto para a busca seqüencial de posições.

Os dados da Tabela 6.8 também estão representados em forma de gráfico na Figura 6.5, onde se pode notar claramente que a busca conjunta e a busca seqüencial de posições estão em um nível de complexidade bem inferior ao dos algoritmos de busca do GSM-AMR. Além disso, pode-se notar, tanto na Tabela 6.8 quanto na Figura 6.5, que a

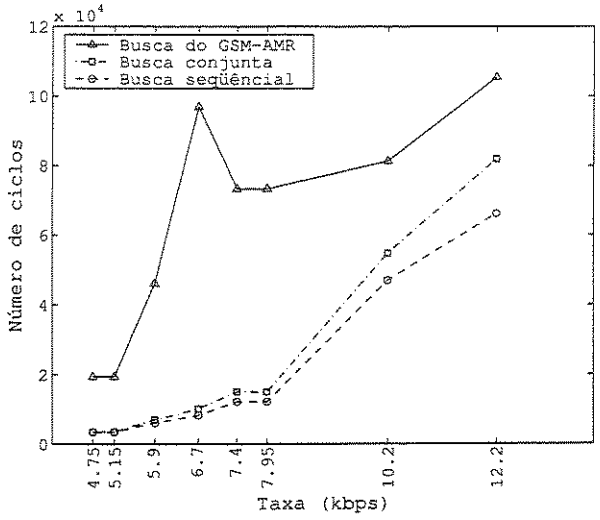


Figura 6.5: Medidas (médias) de complexidade dos algoritmos inseridos no codificador GSM-AMR.

busca seqüencial de posições apresenta em todos os casos complexidade menor do que a busca conjunta. Nas taxas mais baixas, de 4.75 e 5.15kbps, ambas apresentam resultados semelhantes. Porém, com o aumento da taxa de transmissão e o conseqüente aumento do número de pulsos a serem procurados, a diferença entre a busca seqüencial de posições e a busca conjunta se torna considerável. Na taxa de 10.2kbps, por exemplo, a complexidade da busca seqüencial de posições é cerca de 16% maior, enquanto que na taxa de 12.2kbps essa diferença é de cerca de 31%. Ressalva-se ainda que nessas taxas (10.2 e 12.2kbps) a menor complexidade da busca seqüencial de posições ainda vem acompanhada de um ganho no que diz respeito à qualidade de voz, como foi apresentado na Seção 6.5.2.

Capítulo 7

IMPLEMENTAÇÃO OTIMIZADA DE ALGORITMOS DE BUSCA

Como foi apresentado no Capítulo 6, a complexidade da busca no dicionário fixo pode ser reduzida com a utilização de algoritmos de busca sub-ótimos. Dessa forma, para diminuir a complexidade do processo de codificação da voz pode-se considerar a substituição de um algoritmo de busca por um outro mais simples, o que implica geralmente em uma ligeira degradação na qualidade de voz[1].

Existe, entretanto, um outro caminho que pode ser considerado para a redução do tempo gasto no processo de busca, e que consiste simplesmente na tentativa de se obter implementações mais eficientes dos algoritmos de busca. Durante a implementação de um padrão de codificação em um certo processador pode-se empregar algum esforço no desenvolvimento de uma versão do algoritmo de busca adotado específica para o processador considerado, onde se procura tirar a maior vantagem possível dos recursos de hardware disponíveis. Por ser uma das tarefas mais complexas da codificação ACELP, uma melhor implementação da busca no dicionário fixo pode trazer grandes vantagens no desempenho do codificador como um todo.

Nesse sentido, esse capítulo procura apresentar rapidamente a idéia da implementação eficiente de algoritmos, e analisar a sua influência no desempenho dos algoritmos de busca e do codificador em que estão inseridos.

[1] É importante ressaltar que a substituição do algoritmo de busca implica na perda da compatibilidade bit-a-bit do padrão de codificação.

7.1 Códigos Otimizados

Nas aplicações práticas, os padrões de codificação de voz são geralmente implementados em processadores digitais de sinais, e é principalmente nesses casos que se exige um bom desempenho dos codificadores. Os processadores de sinais atuais, entretanto, possuem recursos avançados e bastante específicos, que devem ser bem explorados para uma execução mais eficiente do algoritmo considerado. Nesse sentido, chama-se de algoritmo otimizado uma implementação eficiente do algoritmo, onde os recursos do processador considerado são utilizados da melhor forma possível. Ou seja, a otimização de um certo algoritmo é feita no sentido de criar uma versão dele específica para um certo processador. A versão otimizada geralmente é feita em uma linguagem de programação de nível inferior, mas deve manter uma correspondência bit-a-bit com o algoritmo original.

Geralmente os organismos de padronização disponibilizam para cada padrão de codificação de voz uma implementação de referência[2] correspondente na linguagem C. Dessa forma, a otimização dos módulos de um codificador de voz pode ser resumida como a transformação de um código de referência em C, em um código otimizado em assembly (ou em uma linguagem de nível intermediário) para o processador de sinais considerado.

7.2 Compiladores com Otimização

A maioria dos processadores de sinais atuais possuem compiladores da linguagem C, ou mesmo C++, que são capazes de gerar códigos bem otimizados. Nesses casos, os compiladores são então os responsáveis pela transformação do código em C original em um código de mais baixo nível que utiliza bem os recursos do processador. Geralmente, para que o código gerado pelo compilador seja eficiente, é necessário principalmente que o programa original em C esteja preparado para a otimização do compilador, sendo essa preparação dependente do código e do processador considerados.

A seguir procura-se apresentar de uma maneira geral os passos necessários para a preparação de códigos para a otimização gerada por compiladores. Aborda-se principalmente a preparação de códigos de referência originários dos organismos de padronização

[2]Além da própria especificação, os organismos de padronização também costumam disponibilizar uma implementação de referência para os padrões de codificação. Essas implementações servem tanto para ajudar a definir o padrão quanto como um ponto de partida para o desenvolvimento de outras implementações do padrão de codificação considerado.

mais conhecidos, como ETSI e ITU-T.

Extração de contadores. Algumas implementações de referência, especialmente as do ETSI, possuem dispositivos de análises de complexidade inseridos dentro do código. Uma operação de soma, por exemplo, incrementa um contador que faz parte de um estrutura; ao fim do processo analisado o contador poderá informar quantas operações de soma foram realizadas. Da mesma forma, os outros contadores da estrutura conterão o número de operações de subtração, multiplicação, comparação e outras. Geralmente esses contadores, mesmo quando desabilitados, interferem no processamento realizado pelo compilador, de forma que indica-se que eles sejam extraídos.

Inserção de diretivas. Muitos compiladores C dos processadores de sinais atuais apresentam um desempenho melhor quando são alimentados com informações sobre o programa considerado, o que é feito através de diretivas inseridas dentro do código. Tais diretivas informam, por exemplo, o número mínimo de execuções de um laço, se uma variável é ou não modificada dentro de uma certa função, ou outras coisas do gênero.

Utilização de funções intrínsecas. Em geral as implementações de referência disponibilizadas pelos organismos de padronização possuem um arquivo onde estão reunidas todas as operações aritméticas básicas, como soma e subtração com saturação, extração de valor absoluto, operações de multiplicação e acumulação (MAC *Multiply-and-Accumulate*), deslocamentos a direita ou esquerda, com ou sem saturação, e assim por diante. Todas essas operações aritméticas básicas são implementadas como funções em C para facilitar a portabilidade do código. A Figura 7.1 mostra como exemplo uma função que implementa uma soma com saturação, que consta tanto nos padrões de codificação do ETSI como nos padrões da ITU-T.

Um grande passo para adequar o código ao processador considerado é a implementação das operações aritméticas básicas através do uso de funções intrínsecas do processador. As funções intrínsecas de um processador de sinais são funções mapeadas diretamente às instruções de baixo nível do processador. Ou seja, existe uma correspondência direta entre uma função intrínseca e uma instrução em assembly. Quando uma função intrínseca é chamada, o compilador simplesmente a substitui pela instrução equivalente em assembly. Além de serem bem mais eficientes, as funções

```
/* Soma com saturação em 32 bits */
Word32 L_add (Word32 a, Word32 b)
{
    Word32 result;
    result = a + b;
    if (((a^b) & 0x80000000) == 0)
    {
        if ((result ^ a) & 0x80000000)
        {
            result = (a < 0) ?  0x800000000 :  0x7fffffff;
        }
    }
    return (result);
}
```

Figura 7.1: Função em C que implementa uma soma com saturação. As entradas são de 32 bits e a saída é saturada em 32 bits.

```
/* Soma com saturação em 32 bits */
#define sadd(a,d) _sadd(a,b);
```

Figura 7.2: Exemplo de implementação da soma com saturação da figura 7.1 através de uma macro que utiliza a função intrínseca `_sadd(x, y)` do processador TMS320C6202.

intrínsecas ainda eliminam o overhead necessário para a chamada de uma função e possibilitam que o compilador gere um código bem mais otimizado.

No caso do processador de sinais TMS320C6202, por exemplo, todos os procedimentos descritos pela função em C da Figura 7.1 podem ser executados por uma única instrução em assembly, que possui uma função intrínseca correspondente em C chamada `_sadd(x, y)`. A Figura 7.2 mostra como a função em C que implementa a soma saturada pode ser substituída por uma macro equivalente que utiliza uma função intrínseca do processador.

Adaptação da Estrutura do Código em C. A própria estrutura do código original em C pode ser alterada em determinados pontos para indicar ao compilador como realizar a otimização no trecho considerado. Essa etapa é feita de acordo com o processador considerado e é geralmente chamada de otimização a nível de C (*C-level optimization*). A Figura 7.3 mostra como exemplo um trecho em C com um laço onde um certo vetor

c é construído através da multiplicação dos elementos de dois outros vetores[3]. No caso do DSP TMS320C6202, a otimização do laço considerado é feita calculando-se dois elementos do vetor c por vez, através da utilização dos recursos de paralelismo do processador. Para ajudar a indicar ao compilador que isso deve ser feito é interessante alterar a estrutura do laço em C passando-se a calcular dois elementos de c a cada iteração, como mostra a Figura 7.4.

```
for(i = 0; i < N; i++)
{
    c[i] = (a[i] * b[i]);
}
```

Figura 7.3: Trecho de código em C. Os vetores a[.], b[.] e c[.] possuem comprimento N e elementos de 16 bits.

```
for(i = 0; i < (N>>1); i+=2)
{
    c[i] = (a[i] * b[i]);
    c[i+1] = (a[i+1] * b[i+1]);
}
```

Figura 7.4: Implementação do trecho de código da figura 7.3 voltada para a otimização do compilador C do processador TMS320C6202.

Feitas essas modificações o código em C torna-se bem mais preparado para ser processado pelo compilador. A compilação transforma o código em C em um código em assembly otimizado, que em seguida passa pelo *assembler* e pelo *linker*, resultando então em um arquivo executável que pode ser carregado e executado no processador.

7.3 Otimização Manual

Em alguns casos o código gerado pelo compilador não é suficientemente eficiente para se alcançar o objetivo desejado. Nesses casos, uma alternativa possível é a otimização manual (*hand-optimization*), ou seja, a transformação manual do código em C em uma seqüencia de instruções de mais baixo nível, geralmente assembly. A figura 7.5 mostra o processo em

[3]Um trecho similar de código em C é utilizado como exemplo da otimização manual na Seção B.6

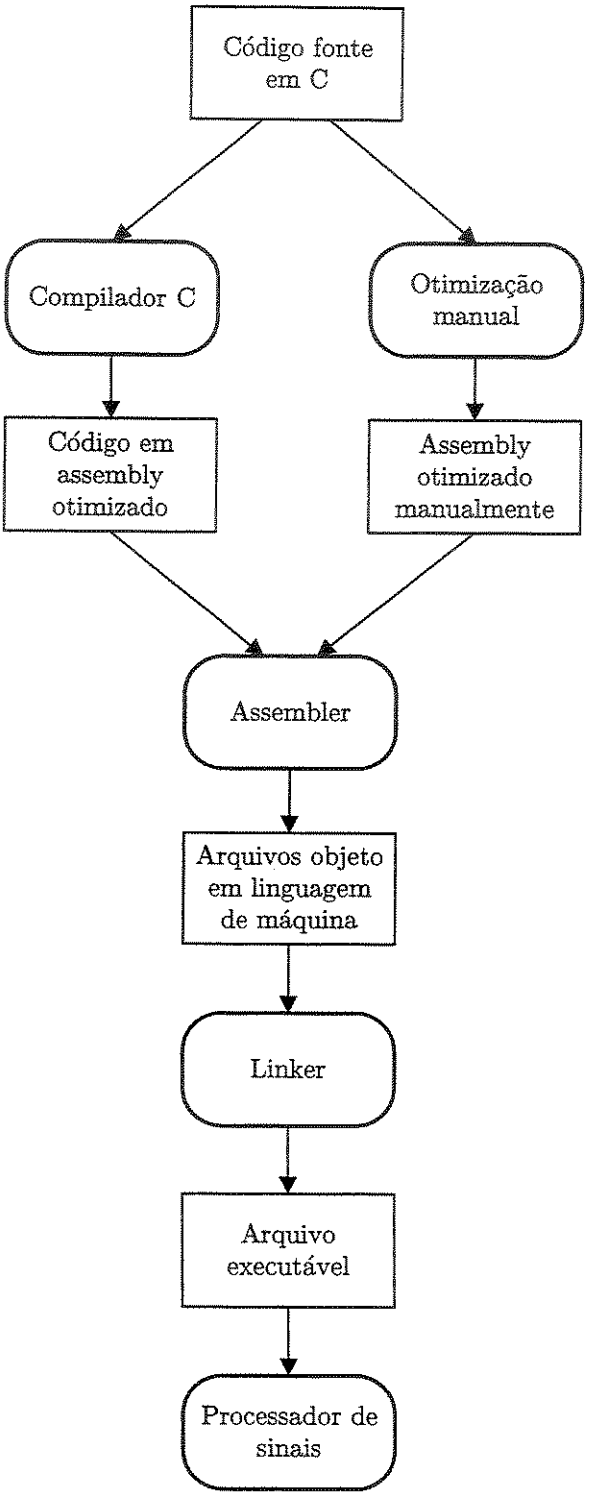


Figura 7.5: Diagrama de blocos representando a geração de um arquivo executável a partir de um código fonte em C.

questão. Apesar de se tratar de um processo lento e trabalhoso, a otimização manual, quando bem feita, apresenta resultados bem melhores do que os obtidos através do compilador.

É interessante aplicar a otimização manual apenas nos trechos de código onde se identifica que pode haver algum ganho sobre o resultado obtido com o compilador. Uma abordagem comum consiste na extração desses trechos e na implementação dos mesmos como sub-rotinas otimizadas da função original. Em outros casos, pode-se trabalhar com a função original completa e otimizá-la por inteiro, criando uma versão otimizada manualmente da função original. A otimização manual dos trechos críticos do programa pode prosseguir até que o programa esteja praticamente completamente otimizado, ou como é comum, apenas até que se tenha atingido o desempenho desejado.

A Seção B.6 do Apêndice B descreve o processo de otimização manual de códigos para o caso particular do processador TMS320C6202, e mostra um exemplo onde esse tipo de otimização diminui em cerca de 16 vezes o número de ciclos necessários para a execução de um trecho de código em C.

7.4 Implementação Otimizada do Algoritmo DFTS do G.729A

Para analisar o ganho decorrente da otimização proporcionada pelo compilador, e principalmente o impacto causado pela otimização manual, foram estudadas diferentes implementações do codificador G.729A [16] no processador digital de sinais de ponto-fixo TMS320C6202 [37] da Texas Instruments^{TM}.

São consideradas três versões do algoritmo de busca DFTS do codificador G.729A:

1. Versão sem otimização. Essa é a mesma versão utilizada no Capítulo 6. Essa versão foi gerada a partir do código de referência do G.729, tendo sido feita apenas a preparação inicial do código, ou seja, a troca das funções de operações básicas por macros que utilizam funções intrínsecas do processador. Nesse caso não existiam contadores de complexidade e nem foi necessária a inclusão de nenhuma diretiva de compilação. Nessa versão a otimização do compilador está desabilitada;

2. Versão com a otimização máxima do compilador. Essa versão é semelhante à versão sem otimização, porém nesse caso é habilitada a otimização máxima do compilador;

3. Versão otimizada manualmente. Essa é uma versão em assembly do algoritmo de busca do G.729A desenvolvida para processadores da família TMS320C62x.

A otimização manual do processo de busca no dicionário fixo foi feita criando-se três sub-funções. As duas primeiras são responsáveis por procedimentos preliminares, enquanto que a terceira executa a busca propriamente dita. As sub-funções criadas foram:

1. Uma função responsável pela pré-associação de amplitudes às posições. No G.729A é utilizado o fator $\lambda = 0$, portanto essa função simplesmente verifica os sinais dos elementos do vetor-alvo filtrado regressivamente $\mathbf{d}$;

2. Uma função responsável pelo cálculo da matriz Φ' a partir da matriz Φ. Trata-se da multiplicação dos elementos da matriz de correlação impulsiva pelos sinais encontrados na pré-associação de amplitudes às posições, de acordo com a equação (5.43);

3. A função que implementa o algoritmo DFTS adotado pelo codificador G.729A.

A tabela 7.1 mostra o número médio de ciclos gastos durante uma execução do processo de busca para cada versão do algoritmo DFTS do codificador G.729A. Pode-se notar que a versão sem otimização exige cerca de 35 mil ciclos para cada execução da busca, enquanto que na versão otimizada pelo compilador esse número se reduz para aproximadamente um terço, ou seja, para cerca de 12 mil ciclos. Por outro lado, o código gerado pela otimização manual produz um resultado ainda melhor. Nesse caso o processo de busca exige apenas cerca de 11% do número médio de ciclos exigidos pela versão sem otimização. Além disso, com a otimização manual o tempo necessário para execução do processo de busca é de apenas um terço do tempo exigido pela melhor implementação gerada pelo compilador.

Tabela 7.1: Análise do algoritmo de busca DFTS do codificador G.729A em diferentes níveis de otimização.

Característica do alg. de busca	Nível de otimização		
	Nenhuma	Otimiz. do compilador	Otimização manual
Número médio de ciclos	35613	11819	4034
Tamanho do código (bytes)	3832	7012	7968

Também são mostrados na Tabela 7.1 os tamanhos das três versões do algoritmo de busca em termos do número de bytes. Pode-se notar que uma das desvantagens da

otimização é o crescimento do tamanho do programa. Em geral, pode-se dizer que quanto mais otimizado um código, maior o seu tamanho. De acordo com a Tabela 7.1, apesar de ser executado bem mais rapidamente do que a versão sem otimização, o código otimizado manualmente possui aproximadamente o dobro do seu tamanho.

A tabela 7.2 mostra a relação entre o número médio de ciclos necessários para o processo completo de codificação de um bloco de 10ms de sinal de voz, e o número médio de ciclos gastos apenas no processo de busca no dicionário fixo. Vale ressaltar que nos dados da tabela 7.2 considera-se um implementação multi-canal do codificador G.729A onde, além da otimização máxima do compilador, foram selecionados alguns módulos para otimização manual. De acordo com os resultados quando é utilizada a versão otimizada pelo compilador cerca de 30% do tempo necessário para a codificação de um bloco é gasto na processo de busca no dicionário. Com a otimização manual o tempo gasto no processo de busca é reduzido para cerca de 13% do tempo total.

Tabela 7.2: Impacto da otimização manual do processo de codificação como um todo.

Tarefa	Nível de otimização do algoritmo de busca	
	Otimiz. do compilador	Otimização manual
Ciclos gastos na codificação de um bloco	78305	62735
Ciclos gastos na busca no dic. fixo	23638	8068
Tempo relativo gasto na busca	30.2%	12.9%

A Tabela 7.3 mostra mais resultados de ganhos obtidos com a otimização manual do algoritmo de busca. Os dados mostram que o esforço empenhado no desenvolvimento de uma versão do algoritmo de busca otimizada manualmente é recompensado com uma redução de cerca de 1.6 MCPS na carga do processador durante a codificação. Essa redução representa um aumento de 6 canais de codificação considerando-se a operação do processor de sinais na freqüência de 200MHz, valor típico para o TMS320C6202. A Tabela 7.3 ainda mostra que com a otimização manual o processador pode executar conjuntamente a codificação e decodificação em 25 canais, 4 a mais do que seria possível apenas com a otimização do compilador.

Por fim, vale ressaltar que embora apresente resultados bem melhores do que os obtidos pelo uso do compilador, a otimização manual se constitui como uma tarefa extremamente trabalhosa e minuciosa. O código otimizado manualmente para o processo de busca do G.729A possui mais de 2500 linhas de programação em assembly apenas para a sub-função que implementa o algoritmo DFTS. As outras duas sub-funções juntas contém mais cerca

Tabela 7.3: Ganho em termos de MCPS e número de canais decorrente da otimização manual do processo de busca no dicionário fixo.

Nível de otimização do algoritmo de busca	Carga do processador (MCPS)		Número de canais a 200MHz	
	Codificação	Decodificação	Codificador	Cod.+Decod.
Otimiz. do compilador	7.83	1.43	25	21
Otimização manual	6.27	1.43	31	25

700 linhas de programação em assembly. Para o desenvolvimento da otimização manual do processo de busca completo foram requeridas cerca de 200 horas-homem, enquanto que a versão do compilador é gerada em apenas alguns minutos.

Capítulo 8

CONCLUSÕES

Os algoritmos de busca de inovações possuem um papel fundamental para os codificadores algébricos. É a adoção de algoritmos sub-ótimos eficientes que possibilita que esses codificadores possam ser implementados em tempo-real e que portanto possam ser utilizados em aplicações práticas. Além disso, algoritmos mais rápidos possibilitam que sejam adotados dicionários com um maior número de pulsos.

As duas características principais de um algoritmo de busca são a sua complexidade e a qualidade de voz resultante do processo de codificação. Em geral, pode-se dizer que quanto menos complexo o algoritmo, pior a qualidade de voz. Dessa forma, o desenvolvimento de algoritmos sub-ótimos eficientes consiste na obtenção de pontos vantajosos na relação de troca entre complexidade e qualidade, ou seja, pontos onde uma grande redução de complexidade tem como custo uma pequena redução de qualidade, ou mesmo situações onde a complexidade é reduzida sendo obtido um certo ganho na qualidade.

Para reduzir a complexidade do processo de busca, a maioria dos algoritmos executa inicialmente uma pré-associação de amplitudes às posições, o que simplifica o problema da busca no dicionário apenas à determinação das posições dos pulsos. A pré-associação é feita através da quantização de um vetor de estimação de amplitudes, composto por um resíduo de predição de longo-prazo e pelo vetor-alvo filtrado regressivamente. Quanto maior a esparsidade do dicionário, maior deve ser o peso dado ao vetor-alvo filtrado. Em casos onde a esparsidade é de 15% ou menos, vetor-alvo filtrado pode ser utilizado como vetor de estimação de amplitudes. Após a pré-associação de amplitudes, cada algoritmo de busca executa, à sua maneira, a busca das posições dos pulsos, sendo essa a grande responsável pela complexidade de cada algoritmo.

Entre os algoritmos analisados, as alternativas de menor complexidade para a execução do processo de busca no dicionário fixo são representadas pelos algoritmos de busca seqüenciais dos pulsos: o algoritmo JPAS e a busca seqüencial de posições, proposta neste trabalho. O algoritmo JPAS, que dispensa a pré-associação de amplitudes, procura a posição e a amplitude de cada pulso seqüencialmente, ou seja, um após o outro. Na busca seqüencial de posições a idéia é a mesma, porém são procuradas apenas as posições dos pulsos.

Por procurar apenas as posições dos pulsos, a busca seqüencial de posições apresenta uma complexidade um pouco menor do que a apresentada pelo algoritmo JPAS. Entretanto, em ambos os casos, o esforço computacional exigido para a busca é bem inferior ao exigido pelos algoritmos de busca adotados nos padrões atuais de codificação ACELP. No que diz respeito à qualidade de voz, a busca seqüencial de posições apresenta, na maioria dos casos, resultados semelhantes ou ligeiramente piores do que os apresentados pela busca conjunta de posição e amplitude. Porém, a implementação dos algorimtos de busca no codificador GSM-AMR mostra que nos casos de dicionários algébricos com muitos pulsos, e onde existem mais de um pulso por trilha, a busca seqüencial de posições apresenta resultados consideravelmente melhores tanto em complexidade como em qualidade de voz.

Além da análise da adoção de algoritmos mais rápidos, no âmbito da redução do tempo gasto no processo de busca, também deve-se considerar a questão da otimização dos algoritmos de busca, que surge durante o porte de codificadores para plataformas avançadas de processamento de sinais. Na implementação de padrões de codificação de voz em processadores digitais de sinais modernos, especialmente nos que contam com recursos de paralelismo, o desenvolvimento de uma implementação bem otimizada do algoritmo de busca para o processador considerado pode gerar ganhos equivalentes aos que seriam obtidos com a substituição do algoritmo por um outro de menor complexidade.

Apêndice A

OS PADRÕES DE CODIFICAÇÃO G.729 e GSM-AMR

Nesse apêndice são apresentados de maneira rápida os padrões de codificação ACELP G.729 [17] e GSM-AMR [10], que pertencem respectivamente aos organismos de padronização ITU-T e ETSI. Nas apresentações procura-se dar ênfase principalmente aos dicionários algébricos e aos procedimentos de busca adotados por cada padrão. Vale ressaltar que também são descritos os procedimentos de busca do G.729A [16], uma versão de baixa complexidade do G.729, e dos padrões GSM-EFR [9] e IS-641 [12], que são englobados pelo padrão GSM-AMR e pertencem respectivamente ao ETSI e TIA/EIA.

A.1 O Codificador G.729

O padrão de codificação de voz G.729 [17] foi estabelecido em 1995 pelo ITU-T, e destina-se principalmente a sistemas de comunicações pessoais, sistemas digitais via satélite, ou outras aplicações como por exemplo a transmissão de voz sobre redes de pacotes. O G.729 trabalha com a taxa de compressão de 8kb/s e baseia-se no modelo CS-ACELP (Conjugate-Structure ACELP), tendo sido projetado para ser equivalente ao codificador ADPCM (G.726 [6]) de 32kb/s na maioria das condições de operação.

A Figura A.1(a) mostra o princípio básico de operação do G.729. O codificador tem como entrada um sinal de voz amostrado a 8kHz e com 16bits/amostra (PCM). O sinal de entrada é segmentado em blocos de 10ms, e a predição é feita com janelas de 30ms, que excedem 5ms para a frente e 15ms para trás do bloco correspondente na linha temporal.

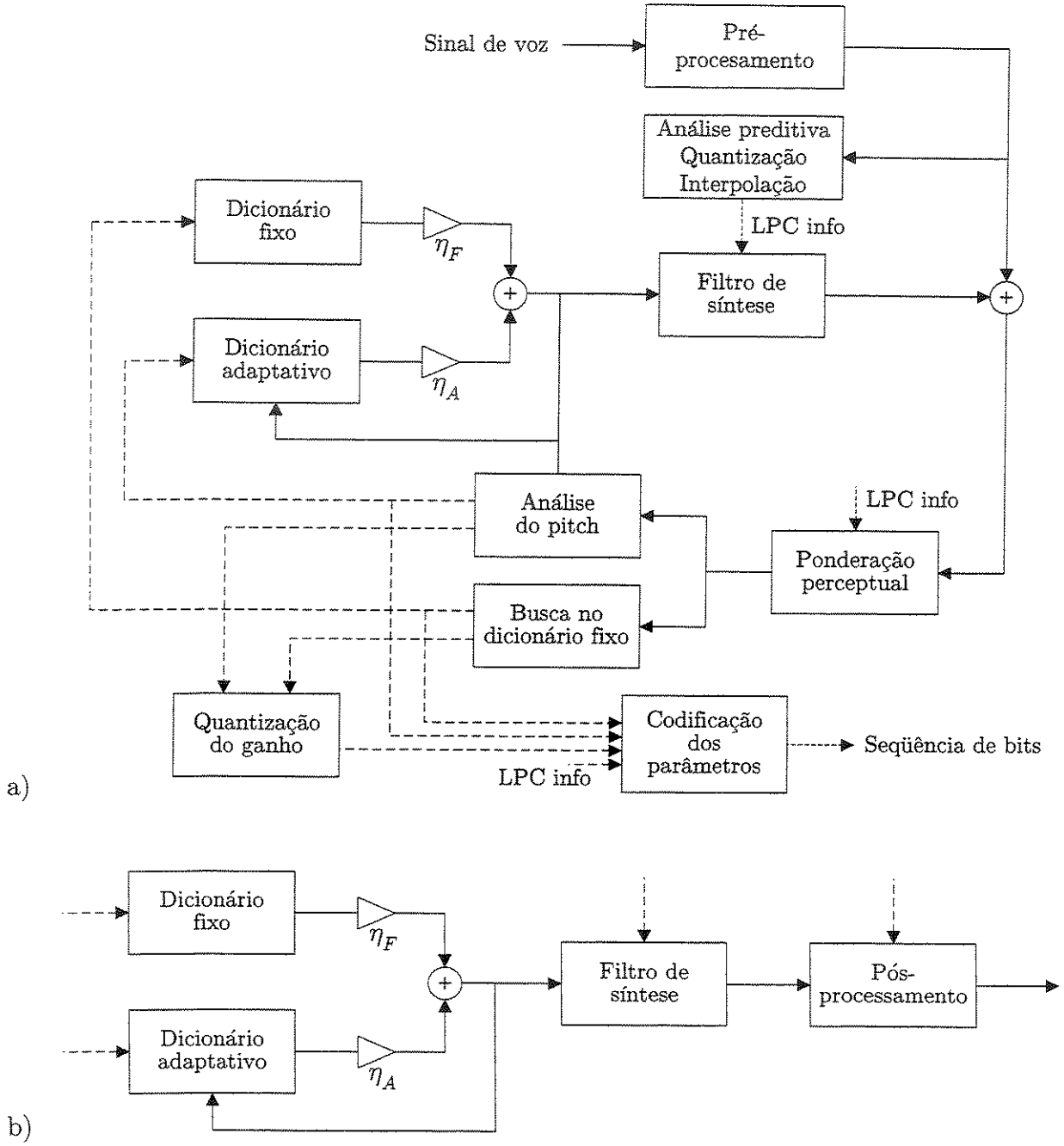


Figura A.1: (a) Princípio de codificação e (b) de decodificação ACELP. (Adaptado de [17])

No pré-processamento o sinal é submetido à um filtro passa-altas com freqüência de corte de 140Hz, e que simultaneamente divide o sinal pelo fator 2, para reduzir a possibilidade de estouro nas implementações em ponto fixo. Logo após é feita uma análise preditiva de décima ordem com o algoritmo de Levinson-Durbin, e os parâmetros encontrados são quantizados no domínio dos pares de linhas espectrais [13] (LSPs *Line Spectral Pairs*). O sinal pré-filtrado também é dividido em dois sub-blocos de 5ms cada. A busca no dicionário adaptativo é feita a cada sub-bloco, num processo onde a excitação adaptativa é encontrada com uma resolução de 1/3 do período de amostragem [22]. Como apresentado na Seção 4.7, a busca no dicionário fixo é feita nos sub-blocos após a subtração da contribuição da excitação adaptativa.

O dicionário algébrico adotado pelo G.729 já foi apresentado anteriormente, e está representado pela Tabela 5.2. Para o processo de busca é utilizado um algoritmo focalizado com fator limitante K igual a 0.4. Para o primeiro sub-bloco é imposto o limite de 105 buscas do último pulso, enquanto que para o segundo sub-bloco o limite consiste em que o total de buscas do bloco não seja maior que 180. O algoritmo utilizado á semelhante ao pseudo-código apresentado pela Figura 5.8. Antes do processo de busca é feita a pré-associação de amplitudes às posição considerando-se $\lambda = 0$.

Após a busca no dicionário fixo, a codificação do bloco é finalizada com a quantização conjunta dos ganhos dos dicionários fixo e adaptativo através do uso de um dicionário de estrutura conjugada (*Conjugate-Structure*) [20].

A Figura A.1(b) mostra o processo que ocorre na decodificação. Pode-se notar que esse processo é bem mais simples, tratando-se basicamente da decodificação dos parâmetros transmitidos e da utilização dos mesmos para a síntese da voz. Fato novo na Figura A.1(b) é apenas a presença do pós-filtro, um mecanismo de melhoria final da qualidade da voz, que elimina certos efeitos desagradáveis da codificação CELP [7].

A.1.1 O G.729 Anexo A

O codificador G.729 anexo A [16], ou simplesmente G729A, foi desenvolvido tendo como base o padrão G.729, e sendo feitas algumas mudanças para tornar mais rápida a execução dos processos de codificação e decodificação. Dessa forma, pode-se dizer que o G.729A representa uma versão de baixa complexidade do codificador G.729.

A seguir constam algumas das mudanças principais adotadas pelo G.729A para a redução da complexidade dos processos de codificação e decodificação [34]:

- O filtro perceptual passa a usar um fator γ fixo (no G.729 esse fator é adaptativo);
- A análise do pitch em malha aberta é simplificada pelo uso de uma dizimação no cálculo das correlações do sinal de voz ponderado [34];
- A busca no dicionário adaptativo foi simplificada. No G.729A a busca procura apenas maximizar a correlação entre a excitação passada e o vetor-alvo filtrado regressivamente, sendo desconsiderada a energia da excitação passada filtrada;
- A busca no dicionário fixo é simplificada pela adoção de uma algoritmo de busca DFTS, em lugar do algoritmo de busca focalizada utilizado pelo G.729;
- No decodificador a pós-filtragem é simplificada pela consideração apenas de atrasos inteiros [7, 34].

Vale ressaltar que a simples substituição do algoritmo de busca focalizada por um algoritmo DFTS é a responsável por cerca de 50% da redução de complexidade alcançada pelo G.729A [34].

No algoritmo de busca em árvore adotado pelo G.729A os 4 pulsos são divididos em 2 grupos de 2 pulsos cada. Inicialmente é feita a associação do pulso $i(0)$ com a trilha 2, $i(1)$ com a trilha 3, $i(2)$ com a trilha 0, e $i(3)$ com a trilha 1. A busca começa determinando a posição dos dois primeiros pulsos ($i(0)$ e $i(1)$), o que é feito testando-se o critério $\tau_i^{(1)}$ para 16 (2x8) combinações de posições. Para o pulso $i(0)$ são testadas as duas posições da trilha que maximizam o valor absoluto do vetor de estimação de amplitudes $b[n]$, enquanto que para o pulso $i(1)$ são testadas as 8 posições da trilha associada a ele. Em seguida, os outros dois pulsos são encontrados testando-se todas as 64 (8x8) possíveis combinações de posições. O procedimento descrito ainda é repetido modificando-se ciclicamente a associação de fases feita para os pulsos, ou seja, $i(0)$ passa a ser associado com a trilha 3, $i(1)$ com a trilha 0, $i(2)$ com a trilha 1, e $i(3)$ com a trilha 2. Em seguida, o processo completo ainda é repetido trocando-se a trilha 3 pela 4.

A.2 O Codificador GSM-AMR

O padrão de codificação GSM-AMR (*Global System Mobile - Adaptive Multi-Rate codec*) [10], foi lançado em 1999 pelo organismo de padronização ETSI (*European Telecommunications Standards Institute*), e tornou-se o codificador padrão para os sistemas de

Tabela A.1: Principais características dos codificadores G.729 e GSM-AMR.

Característica	G.729/A	GSM-AMR
Tipo	ACELP	Multirate-ACELP
Taxas de transmissão	8kbps	4.74, 5.15, 5.9, 6.7, 7.4, 7.95, 10.2 e 12.2kbps
Taxa de amostragem	8kHz	8kHz
Blocos	10ms	20ms
Sub-blocos	2 de 5ms cada	4 de 5ms cada
Atraso de codificação	15ms	25ms
Ordem da análise LPC	10	10

telefonia de terceira geração GSM 2+ e WCDMA. O AMR foi desenvolvido para manter a qualidade da voz sob uma grande variedade de circunstâncias, e para isso ele trabalha com 8 diferentes taxas de transmissão (modos de operação), que variam de 4.75 a 12.2kbps. Além disso, o codificador é capaz de mudar de taxa de transmissão a cada bloco sem perda de qualidade de voz, e a escolha da taxa a ser usada pode ser feita de acordo com as condições do canal, com o tráfego do sistema, ou de acordo com o próprio sinal de voz. A Tabela A.1 mostra um resumo das características do codificador GSM-AMR, onde é feita uma breve comparação com o codificador G.729. Vale ressaltar ainda que nas taxas de 12.2kbps e 7.4kbps, o AMR é, respectivamente, exatamente idêntico aos codificadores GSM-EFR [9] (Enhanced Full Rate) do ETSI e IS-641 (DAMPS EFR) [12] da TIA/EIA.

O GSM-AMR baseia-se no modelo ACELP, e também pode ser representado pela Figura A.1. Pode-se dizer que o funcionamento do codificador em cada uma das 8 taxas de transmissão é basicamente o mesmo, sendo as maiores diferenças devidas ao número de bits utilizados para a quantização dos parâmetros, e ao número de pulsos do dicionário fixo.

A entrada do codificador é um sinal de voz amostrado a 8kHz e com 16bits/amostra (PCM), que é segmentado em blocos de 20ms (160 amostras). No pré-processamento o sinal é submetido à um filtro passa-altas com freqüência de corte de 80Hz, e que simultaneamente escalona o sinal pelo fator 2, para reduzir a possibilidade de estouro nas implementações em ponto fixo. A análise preditiva que se segue é feita duas vezes por bloco para a taxa de 12.2kbps, e apenas uma vez para as outras taxas. Em ambos os casos os parâmetros de predição encontrados são convertidos em pares de linhas espectrais e quantizados.

Cada bloco ainda é dividido em quatro sub-blocos de 5ms cada (40 amostras), e para cada sub-bloco é feita a busca no dicionário adaptativo e, em seguida, a busca no dicionário fixo. Ao fim dos processos de busca os ganhos correspondentes são calculados e quantizados

e a codificação do bloco está completa.

O decodificador pode ser representado pela Figura A.1(b). O processo resume-se à decodificação dos parâmetros transmitidos e à utilização dos mesmos para a síntese da voz. A excitação é construída através da adição dos vetores-código dos dicionários fixo e adaptativo multiplicados pelos respectivos ganhos. Em seguida a voz é reconstruída com a passagem da excitação pelo filtro de síntese. Ainda é utilizado um pós-filtro para melhoria da qualidade final da voz.

A.2.1 Algoritmos de Busca Adotados pelo GSM-AMR

A seguir são descritos os dicionários e os algoritmos de busca usados em cada taxa de transmissão do codificador GSM-AMR.

Busca na Taxa de 12.2kbps (GSM-EFR)

No modo de operação de 12kbps, o codificador GSM-AMR usa o dicionário algébrico representado pela Tabela A.2. As 40 posições são divididas em 5 trilhas e para cada trilha são alocados 2 pulsos.

Tabela A.2: Dicionário algébrico do GSM-AMR na taxa de transmissão de 12.2kbps.

Fases	Pulsos	Posições							
0	i_0, i_5	0	5	10	15	20	25	30	35
1	i_1, i_6	1	6	11	16	21	26	31	36
2	i_2, i_7	2	7	12	17	22	27	32	37
3	i_3, i_8	3	8	13	18	23	28	33	38
4	i_4, i_9	4	9	14	19	24	29	34	39

Para a busca no dicionário é feita a pré-associação de amplitudes às posições com fator $\lambda = 1/2$, e é adotado um algoritmo DFTS bem parecido com o do exemplo apresentado na Seção 5.7. Os 10 pulsos são divididos em grupos de 2 pulsos cada, resultando em um processo de busca de 5 níveis. A Tabela A.3 mostra um resumo do algoritmo DFTS utilizado pelo GSM-AMR para a taxa de 12.2kbps.

As regras de seleção dos pulsos são:

Tabela A.3: Busca DFTS utilizada no GSM-AMR na taxa de transmissão de 12.2kbps.

Nível (p)	Número de pulsos (N_p)	Caminhos candidatos	Regra	Critério de seleção
0	2	4	R1	$\mathbf{b}$
1	2	1	R2	$\tau_i^{(3)}$
2	2	1	R2	$\tau_i^{(5)}$
3	2	1	R2	$\tau_i^{(7)}$
4	2	1	R2	$\tau_i^{(9)}$

R1. Para cada uma das trilhas é encontrada a posição que maximiza o valor absoluto de $b[n]$. Dos valores encontrados é extraído o máximo global de $|b[n]|$, cuja posição correspondente já é determinada em definitivo como a posição do pulso $i(0)$.

Os outros valores máximos de $|b[n]|$ encontrados definem então 4 possibilidades de posição para o pulso $i(1)$. Dessa forma, a combinação de $i(0)$ com cada uma das 4 posições para $i(1)$ gera os 4 caminhos candidatos resultantes da busca no nível zero.

R2. Para os níveis seguintes os dois pulsos de cada grupo são determinados através de uma busca exaustiva. Ou seja, para cada nível p são verificadas todas as 256 (8x8) combinações de posições para os dois pulsos em questão, $i(2p)$ e $i(2p+1)$. Os dois pulsos que maximizam $\tau_i^{(2p+1)}$ são escolhidos e anexados ao caminho considerado, de forma que de cada caminho candidato analisado resulta apenas um caminho candidato estendido.

Vale ressaltar que a determinação da ordem em que os pulsos restantes serão procurados segue um princípio de permutação cíclica, que consiste em dispor todos os pulsos restantes em um círculo e escolher os pares a serem procurados no sentido horário a partir do número que estaria à direita de $i(1)$ (ver Seção 5.7).

Busca na Taxa de 10.2kbps

Nessa taxa o codificador GSM-AMR utiliza o dicionário apresentado anteriormente pela Tabela 5.3. As 40 posições são divididas em 4 trilhas, sendo alocados dois pulsos em cada trilha. Para a busca no dicionário é feita a pré-associação de amplitudes às posições com o fator $\lambda = 1/2$, e é adotado um algoritmo DFTS apresentado pela Tabela A.4. O

algoritmo é semelhante ao utilizado na taxa de 12.2kbps, com a exceção de que nesse caso o número total de pulsos é 8, e portanto existem apenas 4 grupos de 2 pulsos cada. As regras R1 e R2 também são semelhantes às utilizadas para a taxa de 12.2kbps, sendo necessário somente ressaltar que nesse caso resultam apenas 3 caminhos candidatos do nível 1, já que o número de trilhas total é 4.

Tabela A.4: Busca DFTS utilizada no GSM-AMR na taxa de transmissão de 10.2kbps. As regras R1 e R2 são as mesmas utilizadas para a taxa de 12.2kbps

Nível (p)	Número de pulsos (N_p)	Caminhos candidatos	Regra	Critério de seleção
0	2	3	R1	$\mathbf{b}$
1	2	1	R2	$\tau_i^{(3)}$
2	2	1	R2	$\tau_i^{(5)}$
3	2	1	R2	$\tau_i^{(7)}$

Busca nas Taxas de 7.40 (IS-641) e 7.95kbps

Nas taxas de 7.40 e 7.95kbps o codificador GSM-AMR utiliza o mesmo dicionário algébrico adotado pelo G.729 (Tabela 5.2), onde as 40 posições são divididas em 5 trilhas de 8 posições cada. É alocado 1 pulso para cada uma das 3 primeiras trilhas, enquanto que um quarto pulso pode ocupar qualquer uma das duas últimas trilhas.

Além do mesmo dicionário também é utilizado o mesmo processo de busca para as duas taxas consideradas, 7.40 e 7.95kbps. A pré-associação de amplitudes às posições é feita com o fator $\lambda = 0$, e algoritmo utilizado é um tipo de busca DFTS que pode ser resumida pela Tabela A.5.

Pode-se notar na Tabela A.5 que os 4 pulsos são divididos em 4 grupos de 1 pulso cada. O processo de busca em árvore é feito duas vezes, uma para o conjunto de trilhas (0,1,2,3) e outra para o conjunto (0,1,2,4).

As regras utilizadas nesse caso são:

R3. Considera-se 16 possibilidades para o primeiro pulso $i(0)$, que correspondem às 4 posições com os maiores valores absolutos de $b[n]$ para cada uma das 4 trilhas conside-

Tabela A.5: Busca DFTS utilizada no GSM-AMR nas taxas de transmissão de 7.40 e 7.95kbps. O processo é executado duas vezes, uma para o pulso i_3 na trilha 3, e outra para o pulso i_3 na trilha 4.

Nível (p)	Número de pulsos (N_p)	Caminhos candidatos	Regra	Critério de seleção
0	1	16	R3	$\mathbf{b}$
1	1	1	R4	$\tau_i^{(1)}$
2	1	1	R4	$\tau_i^{(2)}$
3	1	1	R4	$\tau_i^{(3)}$

radas. A seqüencia das fases dos pulsos seguintes obedece ao princípio de permutação cíclica.

R4. Os pulsos seguintes são procurados seqüencialmente, verificando-se as 8 posições da trilha devida e selecionando aquela que maximiza $\tau_i^{(p)}$. Note que essa etapa difere das etapas da busca seqüencial de posições porque nesse caso o pulso é procurado apenas nas posições da trilha considerada, e não em todas as posições restantes.

Busca na Taxa de 6.70kbps

Na taxa de 6.70kbps o codificador GSM-AMR utiliza o dicionário algébrico representado pela Tabela A.6. As 40 posições são divididas em 5 trilhas, e são considerados 3 pulsos. O pulso i_0 está obrigatoriamente na trilha 0, enquanto que i_1 pode estar nas trilhas 1 ou 3, e i_2 pode estar nas trilhas 2 ou 4.

Tabela A.6: Possíveis posições dos pulsos no dicionário algébrico adotado pelo padrão GSM-AMR na taxa de transmissão de 6.70kbps.

Fases	Pulsos	Posições							
0	i_0	0	5	10	15	20	25	30	35
1	i_1	1	6	11	16	21	26	31	36
3		3	8	13	18	23	28	33	38
2	i_2	2	7	12	17	22	27	32	37
4		4	9	14	19	24	29	34	39

Para a busca no dicionário da Tabela A.6 é feita uma pré-associação de amplitudes aos pulsos com o fator $\lambda = 0$, e é adotado um algoritmo DFTS que se assemelha ao utilizado para as taxas de 7.40 e 7.95kbps. A Tabela A.7 resume o procedimento de busca da taxa de 6.70kbps.

Tabela A.7: Busca DFTS utilizada no GSM-AMR na taxa de 6.70kbps. O processo é repetido 4 vezes,

Nível (p)	Número de pulsos (N_p)	Caminhos candidatos	Regra	Critério de seleção
0	1	18	R5	$\mathbf{b}$
1	1	1	R6	$\tau_i^{(1)}$
2	1	1	R6	$\tau_i^{(2)}$

Pode-se notar na Tabela A.7 que os 3 pulsos são divididos em 3 grupos de 1 pulso cada. O processo de busca em árvore é feito quatro vezes, para os quatro possíveis conjuntos de trilhas: (0,1,2), (0,1,4), (0,3,2), (0,3,4).

As regras R5 e R6 são as seguintes:

R5. Considera-se 18 possibilidades para o primeiro pulso $i(0)$, que correspondem as 6 posições com os maiores valores absolutos de $b[n]$ para cada uma das 3 trilhas consideradas. A seqüencia das fases dos pulsos seguintes segue o princípio de permutação cíclica.

R6. Semelhante à regra R4 utilizada para as taxas de 7.40 e 7.95kbps.

Busca nas Taxas de 5.90, 5.15 e 4.75kbps

Para as taxas de 5.90, 5.15 e 4.75kbps o procedimento de busca é o mesmo. É feita a pré-associação de amplitudes às posições com o fator $\lambda = 0$, e em seguida é utilizado um algoritmo de busca exaustiva.

Na taxa de 5.90kbps é adotado o dicionário algébrico representado pela Tabela A.8. As 40 posições são divididas em 5 trilhas e são considerados apenas 2 pulsos. O pulso i_0 pode estar na trilha 1 ou na trilha 3, enquanto que o pulso i_1 pode estar em qualquer uma das trilhas 0, 1, 2, ou 4.

Tabela A.8: Possíveis posições dos pulsos no dicionário algébrico adotado pelo padrão GSM-AMR na taxa de transmissão de 5.90kbps.

Fases	Pulsos	Posições							
1	i_0	1	6	11	16	21	26	31	36
3		3	8	13	18	23	28	33	38
0	i_1	0	5	10	15	20	25	30	35
1		1	6	11	16	21	26	31	36
2		2	7	12	17	22	27	32	37
4		4	9	14	19	24	29	34	39

Para as taxas de 5.15 e 4.75kbps é utilizado o dicionário representado pela Tabela A.9. Existem dois pulsos, e as 40 posições são divididas em 5 trilhas. Nesse caso em particular, as trilhas onde podem estar os pulsos i_0 e i_1 dependem do sub-bloco considerado. Para cada sub-bloco são definidos os subconjuntos 1 e 2 contendo cada qual 2 trilhas, onde devem ser alocados os pulsos i_0 e i_1. A cada sub-bloco deve ser escolhido o melhor subconjunto e as melhores posições para os pulsos.

Tabela A.9: Possíveis posições dos pulsos no dicionário algébrico adotado pelo padrão GSM-AMR nas taxas de transmissão de 4.75 e 5.15kbps.

Sub-bloco	Subconjunto	Fases	Pulsos	Posições							
1	1	0	i_0	0	5	10	15	20	25	30	35
		2		2	7	12	17	22	27	32	37
	2	1	i_1	1	6	11	16	21	26	31	36
		3		3	8	13	18	23	28	33	38
2	1	0	i_0	0	5	10	15	20	25	30	35
		3		3	8	13	18	23	28	33	38
	2	2	i_1	2	7	12	17	22	27	32	37
		4		4	9	14	19	24	29	34	39
3	1	0	i_0	0	5	10	15	20	25	30	35
		2		2	7	12	17	22	27	32	37
	2	1	i_1	1	6	11	16	21	26	31	36
		4		4	9	14	19	24	29	34	39
4	1	0	i_0	0	5	10	15	20	25	30	35
		3		3	8	13	18	23	28	33	38
	2	1	i_1	1	6	11	16	21	26	31	36
		4		4	9	14	19	24	29	34	39

Apêndice B

O PROCESSADOR DIGITAL DE SINAIS TMS320C6202

Neste apêndice são apresentados a arquitetura e o funcionamento básico do processador digital de sinais de ponto-fixo TMS320C6202 [37] da Texas Instruments™.

O DSP (*Digital Signal Processor*) TMS320C6202 faz parte da família de processadores de ponto-fixo chamada TMS320C62x, diferindo dos outros membros apenas pela capacidade de memória e freqüência máxima de operação. Dessa maneira, embora o foco principal deste apêndice seja o DSP TMS320C6202, a quase totalidade das considerações apresentadas são válidas para toda a família de processadores de ponto-fixo TMS320C62x.

Vale ressaltar que a apresentação a seguir tem como objetivo expor as características mais importantes do processador do ponto de vista do programador, e é suposto que o leitor detenha algum conhecimento básico sobre a arquitetura de microprocessadores, assim como sobre programação em baixo nível.

B.1 Características Principais

Os DSPs da família 'C62x[1] possuem uma arquitetura voltada para aplicações em tempo-real, caracterizando-se por apresentarem um alto nível de paralelismo e uma estrutura complexa de *pipeline*. Do ponto de vista do programador, pode-se citar como características principais dos processadores da família 'C62x os seguintes ítens:

[1]Por simplicidade serão utilizadas as notações 'C62x e 'C6202 para referenciar respectivamente os processadores da família TMS320C62x e o DSP TMS320C6202 em particular.

- Operação em freqüências de até 200MHz ('C6202);
- Possibilidade de execução de até 8 instruções por ciclo;
- Velocidade de pico de 1600 MIPS (Milhões de Instruções Por Segundo);
- Existência de 8 unidades funcionais: 2 multiplicadores de 16 bits, e 6 unidades lógicas aritméticas de 32 bits;
- Suporte para aritmética de 40 bits em várias operações;
- Disponibilidade de 32 registradores de uso geral de 32 bits cada;
- Possibilidade de execução condicional de qualquer instrução;
- Acesso a memória em 8, 16, ou 32 bits;
- Anexação de saturação ou normalização em algumas instruções sem detrimento de perda de velocidade;

Além das características citadas é importante ressaltar que os DSP 'C6202 possui 128k de memória interna de dados e 256k de memória interna de programa.

B.2 Arquitetura

A Figura B.1 mostra um diagrama de blocos simplificado dos processadores 'C62x. O trecho destacado em cinza corresponde ao núcleo (*core*), ou CPU (*Central Processing Unit*) do processador, enquanto que os blocos que o circundam representam os periféricos e as memórias internas do DSP.

No núcleo do processador são considerados dois caminhos de dados, chamados de caminho A (*data path A*) e caminho B (*data path B*). Para cada caminho é associado um conjunto de registradores de uso geral e um conjunto de unidades funcionais. Dessa maneira as 8 unidades funcionais, chamadas de .L1, .L2, .S1, .S2, .M1, .M2, .D1 e .D2, estão divididas em dois grupos similares de 8 unidades cada, sendo .L1, .S1, .M1 e .D1 associadas ao caminho A, e .L2, .S2, .M2 e .D2 associadas ao caminho B. Além disso os 32 registradores de uso geral também são divididos em dois grupos, que são chamados de arquivo de registradores A (*register file A*) e arquivo de registradores B (*register file B*), cada grupo contendo 16 registradores.

Ainda na Figura B.1 pode-se notar que a memória de programa é endereçada em 32 bits, e acessada em pacotes de 256 bits. Esses pacotes recebem o nome de pacotes de

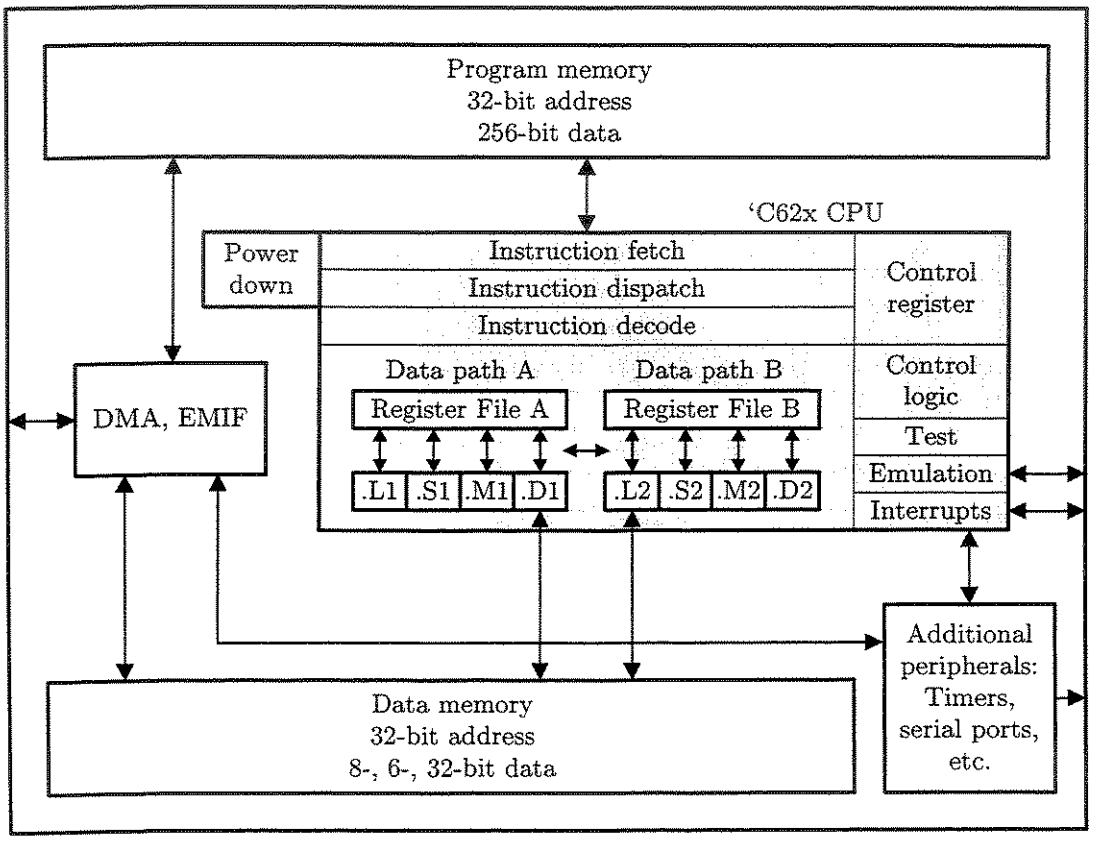


Figura B.1: Arquitetura dos processadores da família TMS320C62x. (Adaptado de [37])

aquisição (*fetch packets*) e na verdade cada um deles contém um conjunto de 8 instruções de 32 bits. Após ser trazido da memória de programa cada conjunto de instruções é então analisado e em seguida cada uma de suas instruções é destinada à unidade funcional devida. Dessa forma, a cada ciclo podem ser despachadas até 8 instruções para o processamento, sendo 4 para cada caminho.

Além de efetuar o processamento dos dados, o núcleo ainda controla todos os periféricos, o que é feito através de registradores específicos, chamados registradores de controle (*control registers*).

O módulo de acesso direto a memória, ou DMA (*Direct Memory Access*) é um dos periféricos mais importantes. Pode-se notar na Figura B.1 que tanto a memória de programa quanto a memória de dados conectam-se ao DMA. Esse módulo possui 4 canais que funcionam de maneira independente, e que podem transferir dados entre as memórias internas e externas sem interferir no funcionamento do núcleo do processador. Além do DMA pode-se notar a presença de outros periféricos na Figura B.1, como por exemplo o módulo de interface de memória externa (EMIF - *External Memory Interface*), responsável pelo gerenciamento da comunicação do processador com qualquer memória externa conectada, seja ela de programa ou dados.

Existem no DSP dois *timers* de uso geral, que podem servir por exemplo para agendar ou contar eventos, gerar pulsos, interromper a CPU, ou ainda sincronizar transferências atuando junto com o DMA. O DSP 'C62x ainda possui três portas seriais multi-canais, chamadas de McBSPs (*Multichannel Buffered Serial Ports*), responsáveis pela comunicação do processador com o mundo externo. As portas seriais comunicam-se com o núcleo do processador através de interrupções, e podem acessar dados tanto da memória interna como da memória externa diretamente através do DMA.

B.3 Registradores de Uso Geral

Cada um dos arquivos de registro do DSP 'C62x é composto de 16 registradores de uso geral. O arquivo de registro A conta com os registradores A0-A15, enquanto que o arquivo de registro B possui os registradores B0-B15. Cada registrador pode ser usado para armazenar dados ou ponteiros, ou para qualquer outro fim.

Todos os registradores suportam dados de até 32 bits. Pode-se usar um registrador, portanto, para armazenar 4 valores de 8 bits, 2 de 16, ou ainda um único de 32 bits. Ainda

é possível lidar com valores de 40 bits, bastando para isso utilizar um par de registradores. Nesse caso os 32 bits menos significantes (LSBs - *Least Significant Bits*) do valor são armazenados em um registrador de número par, e os 8 bits restantes são armazenados no registrador de número ímpar logo acima. O par A1:A0, por exemplo, pode ser usado para armazenar um valor de 40 bits, sendo utilizados 8 bits de A1 e 32 bits de A0.

Tabela B.1: Esquema de armazenamento de dados de 40 bits no DSP 'C62x

Caminho A	Caminho B
A1:A0	B1:B0
A3:A2	B3:B2
A5:A4	B5:B4
A7:A0	B7:B6
A9:A0	B9:B8
A11:A0	B11:B10
A13:A0	B13:B12
A15:A0	B15:B14

B.4 Unidades Funcionais

Cada unidade funcional em um caminho de dados é praticamente idêntica à unidade correspondente no outro caminho, e cada um dos quatro tipos de unidade funcional (.L, .S, .D, .M) pode executar um conjunto limitado de instruções. A seguir são citadas algumas das principais funções de cada tipo de unidade funcional

- Unidades .L1 e .L2: Operações aritméticas e de comparação em 32 ou 40 bits, operações lógicas de 32 bits, e normalização em 32 ou 40 bits;
- Unidades .S1 e .S2: Operações aritméticas e lógicas de 32 bits, deslocamentos em 32 ou 40 bits, operações de *bit-field* em 32 bits, *branches*, transferências de/para os registradores de controle (somente .S2);
- Unidades .M1 e .M2: Operações de multiplicação (16 x 16 bits);
- Unidades .D1 e .D2: Soma e subtração em 32 bits, cálculo de endereços lineares e circulares, leituras e gravações com *offset* de 5 bits, leituras e gravações com *offset* de 15 bits (somente .D2)

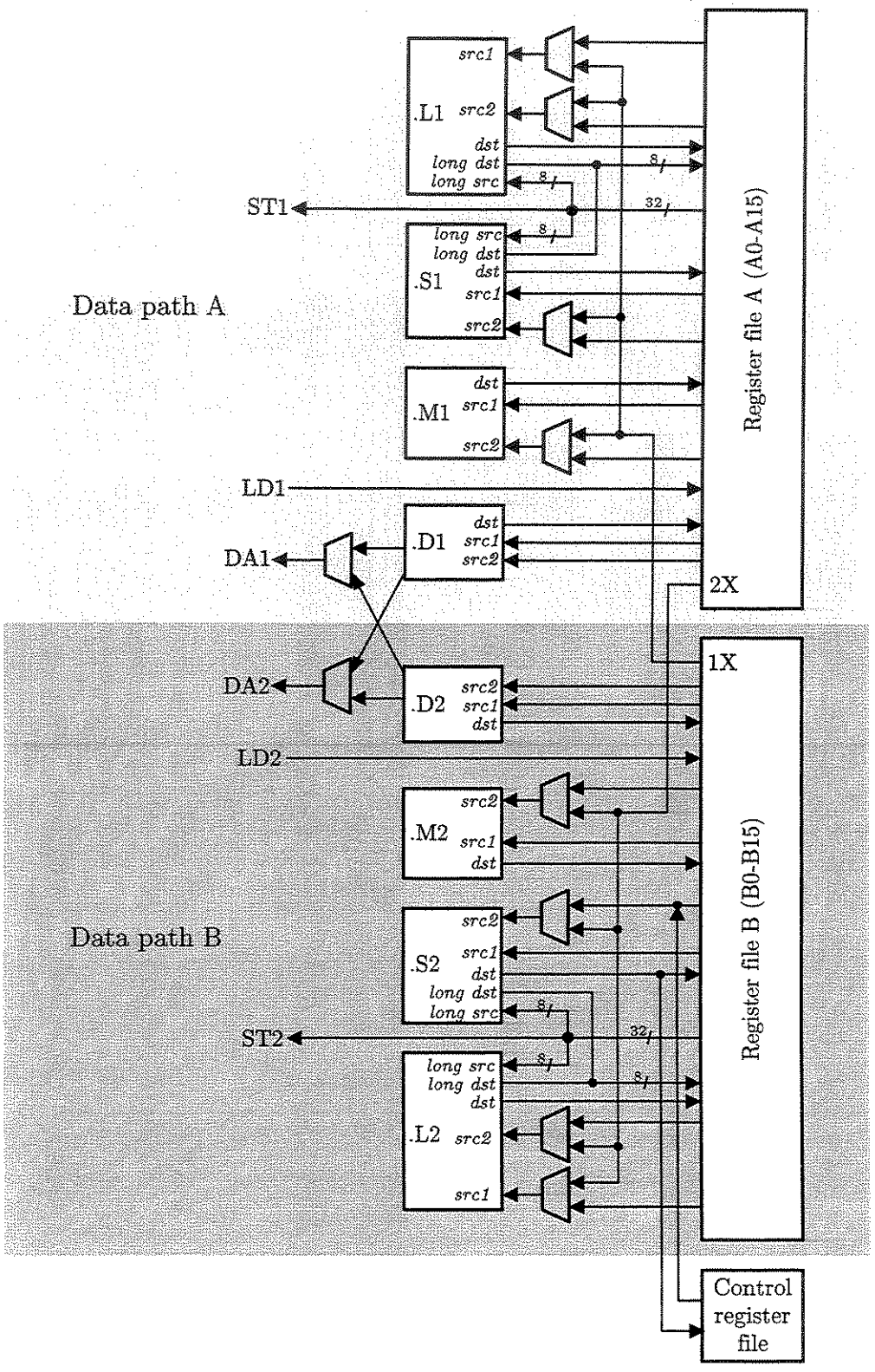


Figura B.2: Caminhos de dados A e B no núcleo no processador 'C62x.

A Figura B.2 mostra em maiores detalhes os caminhos A e B na CPU do DSP 'C62x. As oito unidades funcionais e os dois arquivos de registro são representados por blocos, enquanto que as setas correspondem à caminhos de dados, que em geral são de 32 bits.

Cada unidade funcional tem dois caminhos de dados de 32 bits para leitura de operandos do arquivo de registro correspondente. Da mesma maneira, de cada unidade sai um caminho de dados para escrita de resultados no arquivo de registro correspondente. Para as operações em 40 bits quatro unidades (.L1, .L2, .S1, e .S2) possuem um caminho extra de entrada de 8 bits, assim como um caminho extra de escrita de 8 bits.

Ainda na Figura B.2 pode-se notar a presença dos caminhos cruzados (*cross paths*), que são representados por 1X e 2X. A função desses caminhos cruzados é possibilitar que os registradores de um caminho possam ter como operandos registradores do arquivo de registro do caminho oposto. Dessa forma, o caminho cruzado 1X possibilita que uma unidade funcional do caminho A tenha como operando um registrador do caminho B, enquanto que o caminho cruzado 2X tem a função oposta. As unidades .M e .S podem ter no máximo um de seus operandos oriundos do caminho cruzado, enquanto que as unidades .L podem ter ambos. Não é permitida a utilização de dados de caminhos cruzados como entradas nas unidades .D.

Os caminhos LD1 e LD2, ambos de 32 bits, destinam-se respectivamente à leitura de dados da memória para os arquivos de registro A e B. Também existem dois caminhos de 32 bits para a escrita de dados na memória, LD1 e LD2, que são oriundos respectivamente dos registradores dos arquivos A e B. Os endereços de escrita e leitura são transmitidos ou recebidos através dos caminhos de endereço de dados (*data address paths*) DA1 e DA2, que estão ambos conectados às duas unidades .D. Isso permite que o endereço seja gerado em um lado, para ser utilizado em operações de leitura ou escrita no lado oposto.

B.5 Pipeline

A execução de qualquer instrução no DSP 'C62x requer a passagem da mesma por cada um dos três estágios do *pipeline*:

- Estágio de aquisição (*fetch*);
- Estágio de decodificação (*decode*);
- Estágio de execução (*execute*).

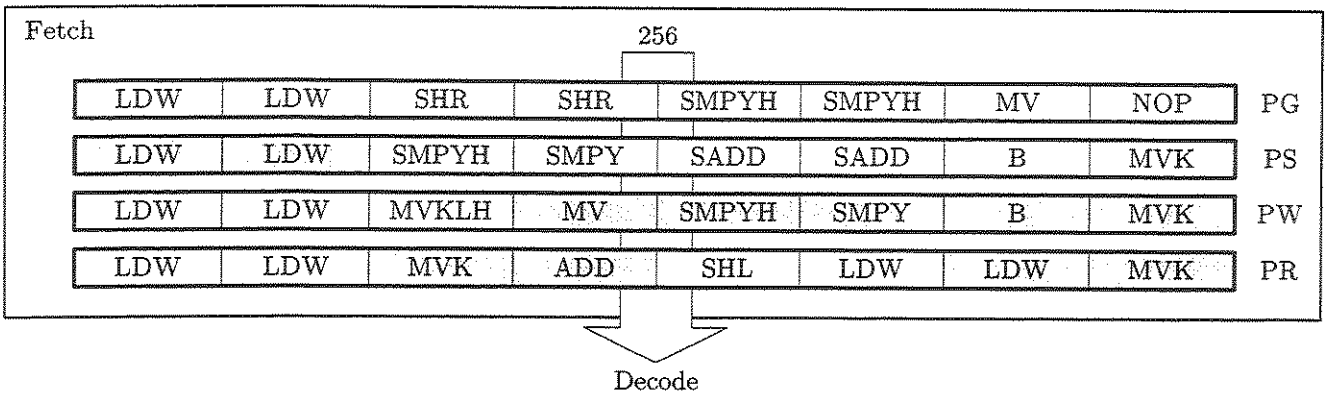


Figura B.3: Fases do estágio de aquisição do *pipeline* do DSP 'C62x. No caso apresentado o pacote de aquisição na fase PR é composto de 4 pacotes de execução (LDW e LDW, MVK e ADD, SHL e LDW, e LDW e MVK). Os 2 pacotes de aquisição seguintes, nas fases PW e PS contém 2 pacotes de execução cada, enquanto que o último pacote de aquisição (PG) corresponde à um único pacote de execução de 8 instruções. (Adaptado de [37])

Cada estágio do *pipeline* ainda se divide em um certo número de fases, cada fase sendo executada geralmente em um ciclo. Todas as instruções requerem o mesmo número de fases do *pipeline* para a passagem pelos estágios de aquisição e decodificação, porém o número de fases do estágio de execução é variável e depende o tipo de instrução sendo executada.

B.5.1 Estágio de Aquisição

Como apresentado anteriormente na Seção B.2, o núcleo do processador 'C62x traz da memória de programa pacotes de 256 bits, contendo 8 instruções cada, os chamados pacotes de aquisição. O estágio de aquisição corresponde justamente aos procedimentos necessários para a transferência do pacote de aquisição da memória de programa para o núcleo do processador. Tais procedimentos constituem 4 fases distintas:

- PG: Geração do endereço (*program address generate*);
- PS: Envio do endereço (*program address send*);
- PW: Espera pelo acesso à memória de programa (*program access ready wait*);
- PR: Recebimento do pacote de aquisição (*program fetch packet receive*).

Na primeira fase é feita a geração do endereço de programa da instrução pela CPU, e na fase seguinte (PS) o endereço gerado é enviado para a memória. Na fase PW ocorre a

leitura na memória, e na fase PR, finalmente, a instrução chega na CPU. Todas as 8 instruções do pacote fluem através das fases do estágio de aquisição simultaneamente. A Figura B.3 mostra a passagem dos pacotes de aquisição através das quatro fases desse estágio.

B.5.2 Estágio de Decodificação

Apesar dos pacotes de aquisição trazidos da memória de programa conterem 8 instruções cada, nem todas as 8 instruções serão executadas em paralelo. Os conjuntos de instruções de um pacote de aquisição que devem ser executadas em paralelo constituem os chamados pacotes de execução (*execute packets*). Dessa forma, cada pacote de aquisição contém um ou mais pacotes de execução, que por sua vez contém ou uma única instrução, ou de 2 a 8 instruções em paralelo. A Figura B.3 mostra a divisão de um pacote de aquisição em pacotes de execução.

É no estágio do *pipeline* de decodificação que são feitas a identificação das instruções que devem ser executadas em paralelo, e a conseqüente separação dos pacotes de aquisição em pacotes de execução. Também é nesse estágio que se verifica quais os registradores e caminhos utilizados por cada instrução. Essas duas tarefas constituem respectivamente as duas fases do estágio de decodificação:

- DP: Entrega das instruções (*instruction dispatch*);
- DC: Decodificação das instruções (*instruction decode*).

A fase DP, portanto, é a responsável pelo desmonte dos pacotes de aquisição em pacotes de execução, e pelo envio das instruções para as unidades funcionais especificadas. A cada ciclo de CPU é extraído um pacote de execução para a fase seguinte (DC), onde é feita a decodificação dos caminhos e dos registradores de origem e destino a serem utilizados por cada instrução.

A Figura B.4 mostra o processamento feito no estágio de decodificação para um pacote de aquisição que contém dois pacotes de execução. O primeiro pacote de execução está destacado com uma cor escurecida, e contém apenas duas instruções de multiplicação (MPY). As duas instruções já passaram em paralelo pela fase DP, e agora estão na fase de decodificação (DC), a um ciclo da execução. O outro pacote de execução é composto por 6 instruções, e está ainda na fase DP. As setas mostram para qual unidade funcional será entregue cada instrução. Vale ressaltar que a instrução NOP (*No OPeration*) não é entregue a nenhuma unidade funcional.

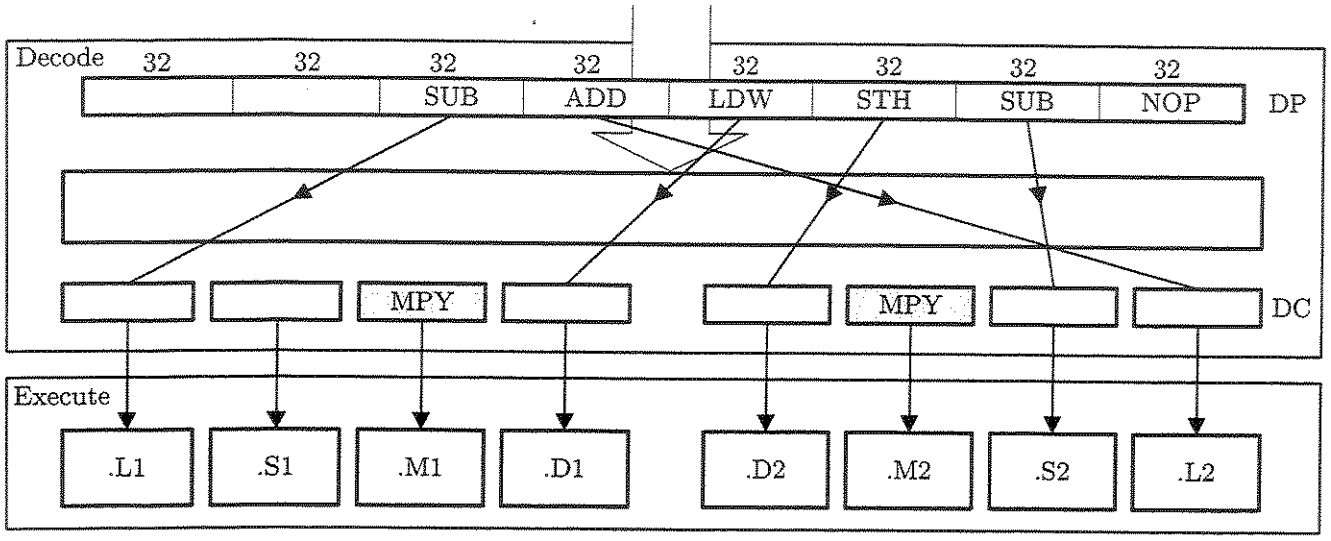


Figura B.4: Fases do estágio de decodificação do *pipeline* do DSP 'C62x. (Adaptado de [37])

É interessante notar que esse estágio do *pipeline* age como um funil para o fluxo de instruções. Caso todos os pacotes de aquisição contivessem 8 instruções em paralelo, o fluxo de pacotes de aquisição pelo *pipeline* seria igual ao fluxo de pacotes de execução, e o DSP estaria sendo usado com 100% de eficiência, atingindo o pico de 1600MIPS (para uma freqüência de operação de 200MHz). Entretanto, em geral, as instruções de um programa não estão todas em paralelo, e nesses casos a eficiência do uso do DSP é menor, podendo chegar ao mínimo de 12,5% em um caso onde o programa contenha apenas instruções seqüenciais (não executadas em paralelo), e portanto cada pacote de aquisição contenha 8 pacotes de execução.

B.5.3 Estágio de Execução

O estágio de execução do *pipeline* do DSP 'C62x é composto de 5 fases, E1-E5 (ver Figura B.5), não sendo entretanto necessária a passagem por todas as cinco fases para o processamento de uma instrução. Ou seja, cada tipo de instrução requer um certo número de fases do estágio de execução para ser processada.

A seguir constam as principais tarefas executadas em cada fase do estágio de execução:

- Fase E1: Essa fase é obrigatória para todas as instruções, e nela é feita a leitura dos

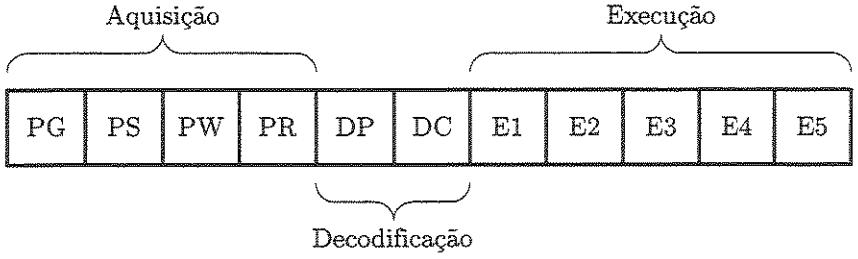


Figura B.5: Fases do *pipeline* do DSP 'C62x. (Adaptado de [37])

operandos e a verificação da condição da instrução, caso exista alguma[2]. Caso se trate de um instrução de leitura ou escrita, o endereço correspondente é gerado. Muitas das instruções do 'C62x necessitam apenas dessa fase do estágio de execução, sendo chamadas de instruções de ciclo único. Para estas instruções o resultado é escrito no registrador devido já nessa primeira fase.

- Fase E2: Nessa fase são completadas as instruções de multiplicação, sendo aqui seus resultados escritos nos registradores devidos. Além disso, para o caso de instruções de leitura, é enviado para a memória o endereço gerado na fase anterior, e para o caso de instruções de escrita é enviado o endereço e o valor a ser escrito. Também é nessa fase que as instruções de ciclo único que trabalham com saturação informam se houve ou não a saturação do resultado obtido[3].

- Fase E3: As instruções de escrita são completadas (i.e. o valor devido é escrito na memória). As instruções de multiplicação com saturação informam se houve ou não a saturação do resultado obtido.

- Fase E4: Essa fase, assim como a próxima, é utilizada apenas para instruções de leitura. Nesse ponto os dados lidos da memória chegam ao núcleo do processador.

- Fase E5: Os dados lidos da memória são escritos no registrador devido (instruções de leitura).

[2]Qualquer instrução pode vir acompanhada de uma certa condição. Na fase E1 essa condição é testada e caso seja inválida a instrução não é executada.

[3]Caso tenha havido saturação é setado um bit chamado SAT do registrador de controle.

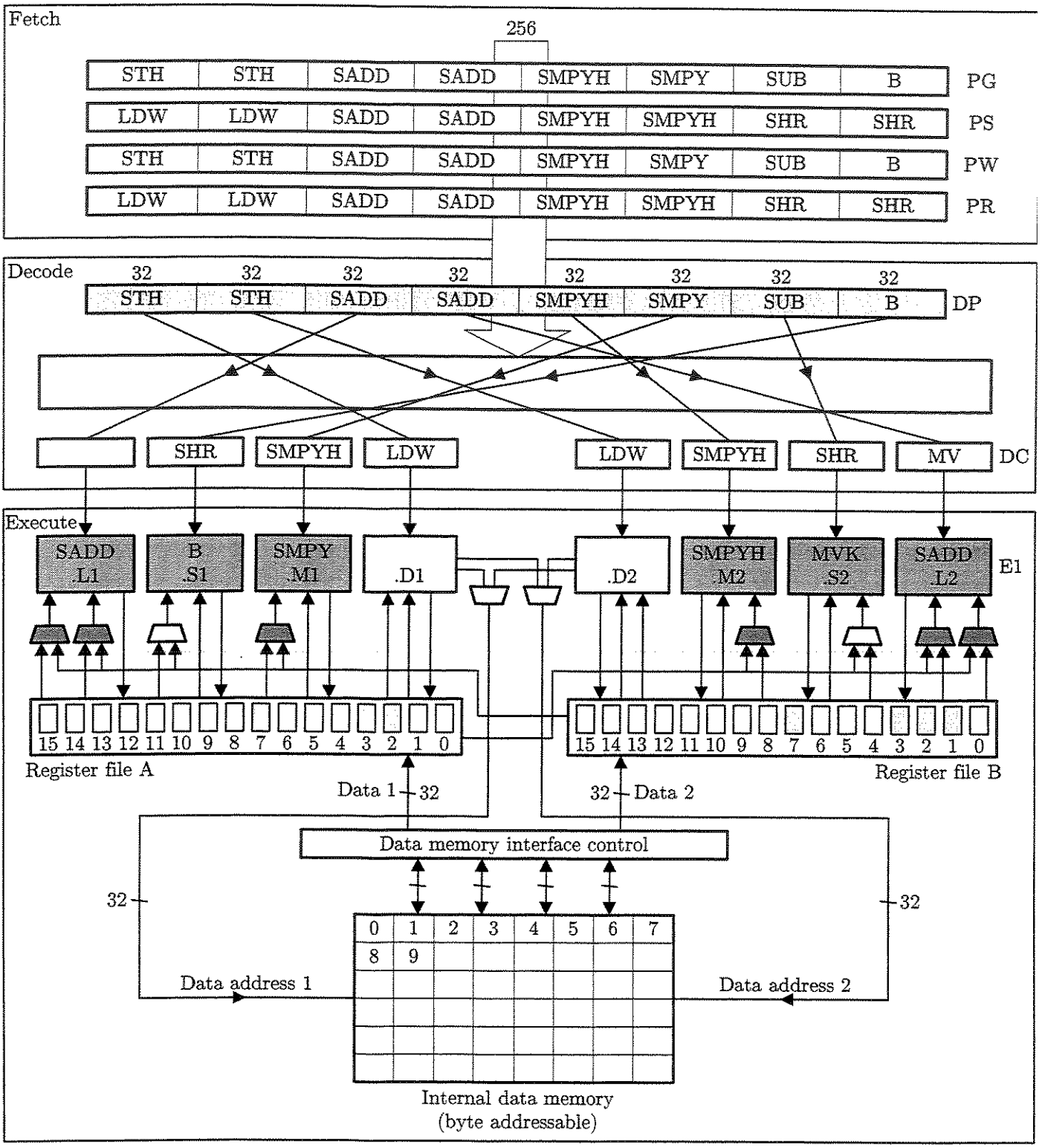


Figura B.6: Diagrama de blocos do DSP 'C62x, enfatizando as fases do *pipeline*. (Adaptado de [37])

B.5.4 Influência do Pipeline na Programação

Durante a programação no 'C62x [39] deve-se estar atento principalmente às diferenças quanto ao número de fases gasto no estágio de execução, visto que todas as instruções gastam o mesmo número de fases nos dois primeiros estágios do *pipeline*. Um fato importante, por exemplo, é que a leitura dos operandos de uma instrução é feita sempre na fase E1, enquanto que a escrita dos resultados só é feita na fase E1 para instruções de ciclo único. Dessa forma, deve-se tomar cuidado para que o resultado de uma primeira instrução já tenha sido escrito no registrador de destino no momento em que uma segunda instrução, que utiliza o resultado da primeira, chegue na fase E1. Esse fato pode ser melhor compreendido pelo trecho de código em *assembly* a seguir, que trata da utilização de um resultado trazido da memória:

```
LDH .D1   *A4, A6   ;Carrega o valor endereçado por A4 para o registrador A6.
NOP       4         ;Espera 4 ciclos para que o valor lido chegue em A6.
SHR .S1   A6,2,A7   ;Usa o valor de A6 (para um deslocamento para a direita)
```

Figura B.7: Influência do atraso das instruções na programação do processador 'C62x.

A primeira instrução, LDH, traz um valor de 16 bits da memória para o registrador A6, utilizando a unidade funcional .D1. A instrução "NOP 4" significa o mesmo que quatro instruções NOP seguidas, e tem como objetivo fazer com que o programa aguarde a chegada no registrador A6 do valor lido da memória pela instrução LDH. A instrução "SHR .S1 A6,2,A7" desloca o valor de A6 dois bits para a esquerda e armazena o resultado em A7, utilizando para isso a unidade funcional .S1. Nesse exemplo é interessante notar que caso os NOPs não houvessem sido utilizados, o valor de A6 usado como entrada pela instrução SHR não seria ainda o valor trazido da memória, causando um resultado indesejado em A7.

Como após uma instrução de leitura é necessário aguardar 4 ciclos para utilizar o valor lido, diz-se que essa instrução tem atraso 4, ou 4 *delay slots*. O atraso de uma instrução equivale portanto ao número de ciclos requeridos para que o resultado dela esteja disponível, contando-se a partir do momento da leitura dos operandos. Dessa maneira, o atraso de uma instrução constitui uma de suas características mais importantes, e que deve ser considerada com muita atenção durante a programação dos DSPs 'C62x. Vale lembrar que as instruções de ciclo único tem atraso zero, e as instruções de multiplicação atraso 2.

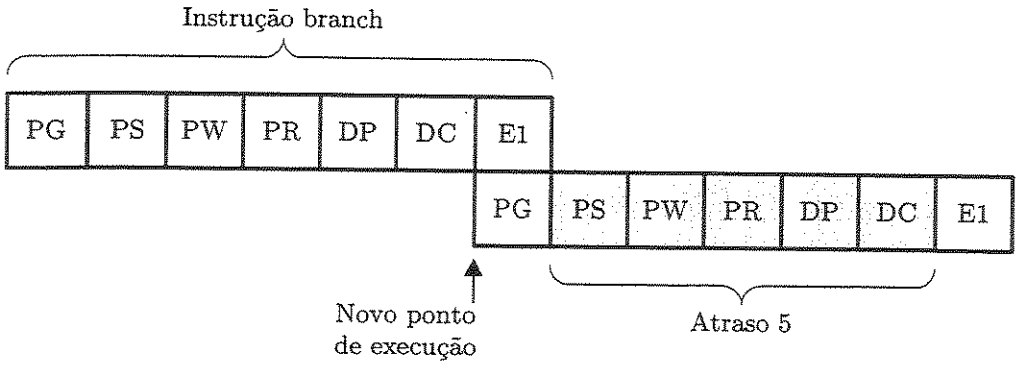


Figura B.8: Atraso existente entre a execução de uma instrução *branch* e a execução da primeira instrução do ponto para onde o programa saltou.

Instrução *Branch*

A instrução *branch*[4], representada pela letra B, solicita um salto na execução do programa. Ou seja, a execução de uma instrução *branch* altera o ponteiro de programa utilizado para trazer os pacotes de aquisição da memória de programa, fazendo com que o código passe a executado a partir de um novo ponto, especificado pela própria instrução.

Embora seja executado na fase E1, o *branch* possui atraso 5. Isso pode ser esclarecido com a ajuda da Figura B.8, que mostra que enquanto uma instrução *branch* está na fase E1 do estágio de execução, o pacote de aquisição está na fase PG do estágio de aquisição, ou seja, está tendo seu endereço gerado. O atraso 5 corresponde portanto ao período entre a execução do *branch* e a execução da primeira instrução no novo ponto de execução.

Do ponto de vista do programador é interessante notar principalmente que após a ocorrência de uma instrução *branch* as cinco instruções (ou conjunto de instruções em paralelo) seguintes ainda serão executadas antes que o programa de fato passe a ser executado a partir de novo ponto.

[4]A instrução *branch* corresponde às instruções GOTO ou JUMP, existentes em muitas linguagens de programação, e que fazem com que a execução do programa salte para um certo ponto especificado.

B.6 Otimização Manual no TMS320C62x

Existe para os processadores a família 'C62x um compilador C [38] capaz de gerar códigos em *assembly* razoavelmente bem otimizados, ou seja, que fazem bom uso dos recursos do DSP, principalmente do paralelismo. O compilador C oferece a possibilidade de escolha do nível de otimização a ser utilizado. Quanto maior o nível de otimização, mais rápido é o programa, mas por outro lado maior é o seu tamanho. Em alguns casos, entretanto, o código gerado pelo compilador, mesmo no maior nível de otimização, não é rápido o suficiente para se alcançar os objetivos desejados. Nesses casos, se o problema for identificado realmente como ineficiência do compilador, pode-se partir para a otimização manual [39], uma abordagem mais trabalhosa, mas por outro lado mais eficiente.

Em geral a otimização manual é aplicada apenas nos trechos de código onde se identifica que pode haver algum ganho sobre o resultado obtido com o compilador. Pode-se, por exemplo, extrair tais trechos e implementá-los como sub-funções otimizadas da função original. Também pode-se procurar otimizar a função completa, gerando um versão otimizada manualmente da função original.

Para o caso do processador TMS320C62x, executar manualmente a otimização pode gerar implementações bem mais rápidas do que as geradas pelo compilador. Entretanto, o ganho alcançado depende do tempo gasto na otimização e do próprio trecho de código considerado.

B.6.1 Exemplo de Otimização Manual

A melhor maneira de explicar o processo de otimização manual no DSP 'C62x é através de exemplos. Será considerado aqui portanto o seguinte exemplo de um trecho de código em C:

```
for(i = 0; i < N; i++)
{
    c[i] = (a[i] * b[i]) >> Qn;
}
```

Figura B.9: Exemplo simples de trecho de código em C facilmente otimizável.

onde Qn é constante, e os vetores a[.], b[.] e c[.] possuem cada um N elementos de 16 bits.

Primeiro Passo

Inicialmente deve-se escrever o código equivalente em assembly para o DSP. Os atrasos de cada função não são considerados por enquanto:

```
loop:
            LDH   .D1   *A4++,A2     ; carrega a[i]
            LDH   .D2   *B4++,B2     ; carrega b[i]
            MPY   .M1   A2,B2,A5     ; a[i] * b[i]
            SHR   .S1   A5,A8,A7     ; c[i] = (a[i] * b[i]) >> Qn
            STH   .D1   A7,*A6++     ; salva c[i]
            SUB   .S1   A1,1,A1      ; decrementa o contador
      [A1]  B     .S1   loop         ; vai para loop (se A1!=0)
```

Figura B.10: Implementação do laço da Figura B.9 em assembly, não sendo considerados os atrasos das instruções.

No trecho de código apresentado os registradores A4, B4 e A6 são ponteiros respectivamente para os vetores a[.], b[.] e c[.]. O registrador A1 é um contador que que deve conter inicialmente o valor N. Além disso A8 deve conter o valor de Qn.

As duas primeiras instruções correspondem ao carregamento dos elementos a[i] e b[i] respectivamente para os registradores A2 e B2. Juntamente com essas operações os registradores usados como ponteiros, A4 e B4, são pós-incrementados[5] para apontarem para o próximo elemento de 16 bits da memória. Em seguida, A2 é multiplicado por B2 e o resultado armazenado em A5. O resultado em A5 por sua vez é deslocado para a direita por Qn (A8) bits e colocado em A7. A instrução STH salva A7 na memória usando A6 como ponteiro, que é também pós-incrementado para apontar para o próximo elemento do vetor c[.]. Na penúltima instrução o contador (A1) é decrementado, e na última instrução é feito um *branch* para o início do laço. A presença do registrador A1 entre colchetes antes dessa instrução corresponde à uma condição, ou seja, a instrução é executada apenas se A1 for diferente de zero. Dessa forma o laço pode ser executado exatamente N vezes, até que o valor do contador A1 chegue a zero.

[5]Esse incremento deve-se à presença do dois sinais + em *A4++ e *B4++, e corresponde na verdade à soma do valor 2 aos ponteiros, já que o endereçamento à memória é feito por bytes e deseja-se avançar 16 bits.

Segundo Passo

O segundo passo trata da consideração dos atrasos das instruções. Também são feitas algumas melhorias prévias que ajudarão na otimização final.

Inserindo os NOPs necessários o código em assembly se torna:

```
loop:
             LDH   .D1   *A4++,A2     ; carrega a[i]
||           LDH   .D2   *B4++,B2     ; carrega b[i]
             NOP         4            ; espera a chegada de a[i] e b[i] em A2 e B2
             MPY   .M1   A2,B2,A5     ; a[i] * b[i]
             NOP                      ; espera o resultado da multiplicação
             SHR   .S1   A5,A8,A7     ; c[i] = (a[i] * b[i]) >> Qn
             STH   .D1   A7,*A6++     ; salva c[i]
       [A1]  SUB   .S1   A1,1,A1      ; decrementa o contador (se A1!=0)
||     [A1]  B     .S1   loop         ; vai para loop (se A1!=0)
             NOP         5            ; cinco ciclos para que o branch ocorra
```

Figura B.11: Versão em assembly do laço da Figura B.9.

A presença do símbolo "||" antes uma instrução diz que ela deve ser executada em paralelo com a instrução anterior. Dessa forma as duas instruções LDH são executadas em paralelo, trazendo os valores de a[i] e b[i] para os registradores A2 e B2 simultaneamente. Como a leitura de memória possui atraso 4, foram inseridos 4 NOPs para aguardar que os valores lidos cheguem em A2 e B2. Outro NOP foi inserido após a multiplicação entre a[i] e b[i], para que o resultado da mesma já esteja em A7 quando este é deslocado para a direita Qn bits. Outra mudança feita foi a paralelização do decremento do contador e do *branch*. Embora não seja necessário (por enquanto) foi incluída uma condição para o decremento do contador, com o objetivo de evitar que este se torne negativo em algum momento. Pode-se notar que essa mudança faz com que o laço seja executado uma vez a mais. Entretanto isso pode ser corrigido facilmente ajustando o valor inicial do contador.

Ainda foram inseridos 5 NOPs no fim do trecho de código, já que o *branch* demora cinco ciclos para ocorrer de fato.

Terceiro Passo

Para alguns laços é possível fazer um melhor uso do processador dividindo a tarefa a ser executada entre os caminhos A e B. No caso em questão isso pode ser feito processando-se simultaneamente pares de elementos dos vetores a[.] e b[.]. Para isso é conveniente utilizar a

instrução LDW no lugar de LDH. Enquanto a instrução LDH carrega da memória elementos de 16 bits, a instrução LDW trabalha com elementos de 32 bits. Dessa forma LDW pode ser usada para carregar a[i] e a[i+1], e b[i] e b[i+1] simultaneamente, resultando na seguinte implementação para o laço da Figura B.9:

```
loop:
             LDW   .D1    *A4++,A2     ; carrega a[i] e a[i+1] para A2
||           LDW   .D2    *B4++,B2     ; carrega b[i] e b[i+1] para B2
             NOP          4            ; espera as instruções de leitura
             MPY   .M1    A2,B2,A5     ; a[i] * b[i]
||           MPYH  .M2    A2,B2,B5     ; a[i+1] * b[i+1]
             NOP                       ; espera os resultados das multiplicações
             SHR   .S1    A5,A8,A7     ; c[i] = (a[i] * b[i]) >> Qn
||           SHR   .S2    B5,B8,B7     ; c[i+1] = (a[i+1] * b[i+1])>> Qn
             STH   .D1    A7,*A6+[2]   ; salva c[i]
||           STH   .D2    B7,*B6+[2]   ; salva c[i+1]
      [A1]   SUB   .S1    A1,1,A1      ; decrementa o contador (se A1!=0)
||    [A1]   B     .S2    loop         ; vai para loop (se A1!=0)
             NOP          5            ; cinco ciclos para que o branch ocorra
```

Figura B.12: Implementação do laço da Figura B.9 com a utilização em paralelo dos caminhos A e B.

Pode-se notar que dessa maneira o processamento dos elementos pares dos vetores a[i] e b[i] é feito todo no caminho A, enquanto que os elementos ímpares são processados no caminho B. O laço inicia com o carregamento dos elementos a[i] e a[i+1], destinados respectivamente aos 16 bits mais significantes e menos significantes do registrador A2. Em paralelo é feito o carregamento dos elementos b[i] e b[i+1], que se destinam ao registrador B2. Logo que chegam simultaneamente aos registradores A2 e B2, os elementos carregados são utilizados para o cálculo de a[i]*b[i] e a[i+1]*b[i+1]. As duas multiplicações são feitas em paralelo, a primeira por MPY, e a segunda por MPYH, que multiplica os 16 MSB de A2 e B2 e coloca o resultado em B5. Os resultados das duas multiplicações chegam ao mesmo tempo a A5 e B5, e são deslocados Qn bits para a direita (para isso A8 e B8 devem conter o valor Qn). Em seguida c[i] e c[i+1] são salvos na memória também em paralelo. Para isso são usados dois ponteiros para o vetor c[.], A6 e B6, onde B6 aponta para uma posição à frente de A6. Dessa forma c[i] e c[i+1] são salvos com duas instruções STH que pós-incrementam os valores dos ponteiros duas posições, para que o deslocamento entre eles se mantenha.

Nota-se ainda no trecho de código da Figura B.12 que devem ser executadas apenas N/2 iterações do laço, já que as amostras pares e ímpares dos vetores são processadas em

paralelo. A divisão do processamento entre os caminhos A e B impõe portanto a restrição de que o comprimento dos vetores a[.], b[.] e c[.] seja par. Isso, entretanto, não representa maiores problemas, dado que no caso de vetores de comprimento ímpar pode ser feito posteriormente o processamento de uma amostra restante.

Quarto Passo

Apesar de fazerem parte do processo de otimização manual, os três primeiros passos apresentados correspondem somente à ajustes preliminares necessários para a otimização propriamente dita. O grande ganho no que se refere à melhor utilização dos recursos do processador 'C62x ocorre na verdade quando se observa que as diferentes iterações do laço considerado são independentes, ou seja, que não existe uma relação de dependência entre uma certa iteração do laço e as iterações anteriores ou futuras. Esse fato possibilita que uma nova iteração do laço possa ser iniciada antes do término da iteração anterior, num processo de *pipeline* do laço, onde em um certo instante cada uma de diversas iterações do laço está em uma certa fase de execução.

O procedimento a seguir, que resulta finalmente na utilização eficiente das unidades funcionais, pode ser melhor compreendido representando-se a seqüência de instruções do trecho de código da Figura B.12 através de uma tabela onde as colunas correspondem aos diferentes instantes da execução do laço considerado:

Tabela B.2: Tabela referente ao laço da Figura B.12.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
D1	LDW								STH									
D2	LDW								STH									
M1						MPY												
M2						MPYH												
L1																		
L2																		
S1								SHR		SUB								
S2								SHR		B								

Da Tabela B.2 pode-se notar que cada unidade funcional é utilizada no máximo duas vezes em cada iteração do laço. Isso indica que uma nova iteração do laço pode ser iniciada

no mínimo 2 ciclos após o início da primeira. Esse número mínimo é característica de cada laço e é chamado de intervalo mínimo de iteração (*minimum iteration interval*). Na verdade o intervalo mínimo de iteração não depende apenas do número máximo de vezes que uma unidade funcional é utilizada, mas também das relações de dependência entre as iterações. No caso, entretanto, o número mínimo de iteração é 2, e uma nova iteração do laço pode ser iniciada a cada 2 ciclos.

Antes de iniciar a implementação de um esquema de *pipeline* para o laço, deve-se ainda fazer mais um último ajuste na disposição das instruções da Tabela B.2. Como o intervalo mínimo de iteração é 2, convém realocar a posição das instruções STH para o instante 9. Tal procedimento visa evitar o conflito que ocorreria entre as instruções LDH e STH, e será melhor compreendido após a otimização do laço.

O resultado desse reposicionamento está mostrado na Tabela B.3. Vale ressaltar que não existe qualquer problema com esse reposicionamento, visto que as instruções STH usam resultados calculados por SHR que ficam disponíveis no instante 8, e que evidentemente continuam disponíveis no instante 9. Juntamente com as instruções STH, também foram reposicionadas as instruções SUB e B, relativas ao decremento do contador e ao *branch*. Como a instrução *branch* tem atraso 5, é uma vantagem posicioná-la no instante 4, já que dessa maneira o *branch* ocorrerá logo após a execução das instruções do instante 9, evitando assim que o processador prossiga desnecessariamente do instante 10 em diante, como ocorria no esquema da Tabela B.2.

Tabela B.3: Reposicionamento das instruções STH.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
D1	LDW									STH								
D2	LDW									STH								
M1						MPY												
M2						MPYH												
L1																		
L2																		
S1					SUB			SHR										
S2					B			SHR										

A partir da Tabela B.3 pode-se partir finalmente para o processo final de otimização do laço. Trata-se de considerar que uma nova iteração do laço tem início a cada 2 ciclos,

que corresponde ao intervalo mínimo de iteração do caso considerado. A Tabela B.4 mostra o resultado final, e a Figura B.13 ilustra a idéia do procedimento.

Tabela B.4: Otimização do laço da Figura B.9.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	Prólogo								Núcleo		Epílogo							
D1	LDW		LDW		LDW		LDW		LDW	STH		STH		STH		STH		STH
D2	LDW		LDW		LDW		LDW		LDW	STH		STH		STH		STH		STH
M1						MPY		MPY		MPY		MPY		MPY				
M2						MPYH		MPYH		MPYH		MPYH		MPYH				
L1																		
L2																		
S1					SUB		SUB	SHR	SUB	SHR		SHR		SHR		SHR		
S2					B		B	SHR	B	SHR		SHR		SHR		SHR		

Na Tabela B.4 foram repetidas a cada 2 ciclos as instruções da Tabela B.3, e cada repetição está evidenciada com um nível de cinza diferente. Os valores lidos pelas instruções LDW do instante 0 serão utilizados por MPY e MPYH no instante 5. Já os valores lidos por LDW no instante 2 serão utilizados por MPY e MPYH do instante 7, e assim por diante. A repetição das instruções é feita até que se atinja um ponto onde é formado o chamado núcleo do laço (*loop kernel*), que corresponde a uma seqüência de poucas colunas que representa uma iteração completa do laço. Para que o núcleo do laço seja executado repetidas vezes, faz-se com que a instrução *branch* não se destine ao início do laço (instante 0), sim ao início do núcleo do laço (instante 8). Dessa forma, a seqüência temporal de execução da Tabela B.4 é: 0, 1, 2, ..., 7, 8, 9, 8, 9, ..., 8, 9, 10, 11, ..., 17. O trecho inicial, que antecede o núcleo, é chamado de prólogo, e o trecho final, que corresponde ao esvaziamento do *pipeline* do laço, é chamado de epílogo. A Figura B.13 ilustra a idéia do procedimento.

Durante a execução do núcleo, no caso em questão, existem 5 iterações do laço sendo executadas simultaneamente. Dessa maneira, considerando que estão sendo executadas simultaneamente as iterações n, $n+1$,n+2, $n+3$ e $n+4$, pode-se dizer que no núcleo estão sendo feitas as seguintes operações:

- Instante 8:
  - As instruções LDW estão lendo dados correspondentes à iteração $n+4$.

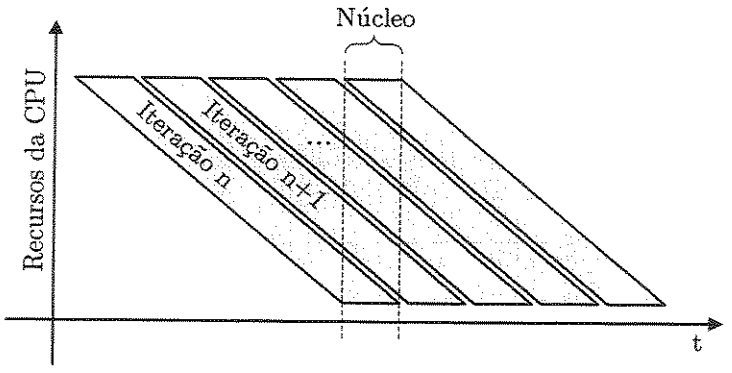


Figura B.13: *Pipeline* relativo às iterações de um laço.

  - A instrução SUB faz o decremento do contador da iteração $n+2$;

- Instante 9:
  - As instruções STH estão gravando dados da iteração n.
  - As instruções MPY e MPYH estão multiplicando dados da iteração $n+2$.
  - As instruções SHR estão aplicando deslocamentos em dados da iteração $n+1$.

A Figura B.14 mostra o trecho de código representado pela Tabela B.4. Pode-se notar que cada bloco de instruções corresponde à uma coluna da tabela considerada.

Ainda na Tabela B.4 pode-se notar a razão pela qual foi feito um ajuste no posicionamento da instruções STH: na posição em que foram colocadas as instruções STH liberam espaço para as instruções LDW no núcleo do laço. A manutenção das instruções STH na sua posição original impossibilitaria que a uma nova iteração do laço fosse iniciada a cada dois ciclos.

É importante ressaltar ainda que a adoção do esquema de *pipeline* para laços impõe como restrição um número mínimo de iterações a serem executadas. No caso em questão, por exemplo, ainda que o núcleo do código representado pela Tabela B.4 seja executado uma única vez, são feitas 5 leituras de 32 bits cada, ou seja, 10 iterações do laço original. Nesse caso, portanto, o número mínimo de iterações é 10.

Outro ponto importante de ser analisado é a redução de complexidade alcançada com a técnica de otimização manual apresentada. A Tabela B.5 mostra algumas medidas de complexidade das implementações consideradas para o laço da Figura B.9. A implementação inicial em *assembly* do laço considerado, representada pela Figura B.11, exigia 15 ciclos para que fosse completada uma iteração, sendo necessários portanto $15 \times N$ ciclos para a execução

completa do laço. Em um segundo momento, com uma implementação que distribui o processamento entre os caminhos A e B (Figura B.12, a execução completa do laço passa a exigir $15 \times N/2$ ciclos. Finalmente, com a implementação do laço em um esquema de *pipeline*, consegue-se uma redução bastante significativa. Da observação da Tabela B.4, ou mesmo do trecho de código da Figura B.14, pode-se notar que no núcleo do laço otimizado é calculado uma média de um elemento do vetor c[.] a cada ciclo. O número total de ciclos para a execução completa do laço nesse caso é $N + 6$, consideradas as influências do prólogo e do epílogo. Por fim, pode-se notar ainda que após todas as otimizações a eficiência de uso dos recursos do DSP no núcleo do *pipeline* do laço é de 62,5%.

Tabela B.5: Complexidade (em número de ciclos) dos esquemas apresentados para a implementação do laço da Figura B.9

Técnica	Complexidade	Restrição
Implementação simples em assembly (fig B.11)	$15 \times N$	Nenhuma
Implementação com o uso dos caminhos A e B (fig B.12)	$15 \times N/2$	N par
Implementação do laço com *pipeline* (fig B.14)	$N + 6$	N par e ≥ 10

```
                LDW   .D1    *A4++,A2     ; carrega a[i] e a[i+1]
||              LDW   .D2    *B4++,B2     ; carrega b[i] e b[i+1]

                NOP

                LDW   .D1    *A4++,A2     ; carrega a[i] e a[i+1]
||              LDW   .D2    *B4++,B2     ; carrega b[i] e b[i+1]

                NOP

                LDW   .D1    *A4++,A2     ; carrega a[i] e a[i+1]
||              LDW   .D2    *B4++,B2     ; carrega b[i] e b[i+1]
||      [A1]    SUB   .S2    A1,1,A1      ; decrementa o contador (se A1!=0)
||      [A1]    B     .S2    loop         ; vai para loop (se A1!=0)

                MPY   .M1    A2,B2,A5     ; a[i] * b[i]
||              MPYH  .M2    A2,B2,B5     ; a[i+1] * b[i+1]

                LDW   .D1    *A4++,A2     ; carrega a[i] e a[i+1]
||              LDW   .D2    *B4++,B2     ; carrega b[i] e b[i+1]
||      [A1]    SUB   .S2    A1,1,A1      ; decrementa o contador (se A1!=0)
||      [A1]    B     .S2    loop         ; vai para loop (se A1!=0)

                MPY   .M1    A2,B2,A5     ; a[i] * b[i]
||              MPYH  .M2    A2,B2,B5     ; a[i+1] * b[i+1]
||              SHR   .S1    A5,A8,A7     ; c[i] = (a[i] * b[i]) >> Qn
||              SHR   .S2    B5,B8,B7     ; c[i+1] = (a[i+1] * b[i+1])>> Qn

;-------------------------------------------------------------
;Loop Kernel
;-------------------------------------------------------------
loop:
                LDW   .D1    *A4++,A2     ; carrega a[i] e a[i+1]
||              LDW   .D2    *B4++,B2     ; carrega b[i] e b[i+1]
||      [A1]    SUB   .S2    A1,1,A1      ; decrementa o contador (se A1!=0)
||      [A1]    B     .S2    loop         ; vai para loop (se A1!=0)

                STH   .D1    A7,*A6+[2]   ; salva c[i]
||              STH   .D2    B7,*B6+[2]   ; salva c[i+1]
||              MPY   .M1    A2,B2,A5     ; a[i] * b[i]
||              MPYH  .M2    A2,B2,B5     ; a[i+1] * b[i+1]
||              SHR   .S1    A5,A8,A7     ; c[i] = (a[i] * b[i]) >> Qn
||              SHR   .S2    B5,B8,B7     ; c[i+1] = (a[i+1] * b[i+1])>> Qn
;-------------------------------------------------------------
                :
```

Figura B.14: Resultado final da otimização manual do trecho de código da Figura B.9 (continua na próxima página).

```
                :

                NOP

                STH   .D1   A7,*A6+[2]   ; salva c[i]
||              STH   .D2   B7,*B6+[2]   ; salva c[i+1]
||              MPY   .M1   A2,B2,A5     ; a[i] * b[i]
||              MPYH  .M2   A2,B2,B5     ; a[i+1] * b[i+1]
||              SHR   .S1   A5,A8,A7     ; c[i] = (a[i] * b[i]) >> Qn
||              SHR   .S2   B5,B8,B7     ; c[i+1] = (a[i+1] * b[i+1])>> Qn

                NOP

                STH   .D1   A7,*A6+[2]   ; salva c[i]
||              STH   .D2   B7,*B6+[2]   ; salva c[i+1]
||              MPY   .M1   A2,B2,A5     ; a[i] * b[i]
||              MPYH  .M2   A2,B2,B5     ; a[i+1] * b[i+1]
||              SHR   .S1   A5,A8,A7     ; c[i] = (a[i] * b[i]) >> Qn
||              SHR   .S2   B5,B8,B7     ; c[i+1] = (a[i+1] * b[i+1])>> Qn

                NOP

                STH   .D1   A7,*A6+[2]   ; salva c[i]
||              STH   .D2   B7,*B6+[2]   ; salva c[i+1]
||              SHR   .S1   A5,A8,A7     ; c[i] = (a[i] * b[i]) >> Qn
||              SHR   .S2   B5,B8,B7     ; c[i+1] = (a[i+1] * b[i+1])>> Qn

                NOP

                STH   .D1   A7,*A6+[2]   ; salva c[i]
||              STH   .D2   B7,*B6+[2]   ; salva c[i+1]
```

Figura B.15: (continuação...) Resultado final da otimização manual do trecho de código da Figura B.9.

Bibliografia

[1] ADOUL, J.-P.; LAFLAMME, C. Algebraic codebook with signal-selected pulse amplitude/position combinations for fast coding of speech. *United States Patent 5,754,976*, 1995.

[2] ADOUL, J.-P.; LAFLAMME, C. Depth-first algebraic-codebook search for fast coding of speech. *United States Patent 5,701,392*, 1995.

[3] ATAL, B.; SCHROEDER, M. Adaptive predictive coding of speech signals. *Conf. Comm. and Proc.*, p. 360–361, 1967.

[4] ATAL, B.; SCHROEDER, M. Linear prediction analysis of speech based on a pole-zero representation. *J. Acoust. Soc. Am.*, v. 64(5), p. 1310, 1978.

[5] BARBOSA, L. M. J.; MELONI, L. G. P. A sequential search algorithm with signal-selected pulse amplitudes. *IEEE International Telecommunications Symposium*, 2002.

[6] CCITT. *Recommendation G.721, 32kb/s adaptive differential pulse code modulation (ADPCM)*. Blue Book, v. III, Fasc. III.3, Outubro 1988.

[7] CHEN, J.-H.; GERSHO, A. Adaptive postfiltering for quality enhancement of coded speech. *IEEE Transactions On Speech and Audio Processing*, v. 3, n. 1, Janeiro 1995.

[8] DELLER, J. R.; PROAKIS, J. G.; HANSEN, J. H. *Discrete-time processing of speech signals*. Macmillan, New York, 1993.

[9] ETSI - European Telecommunications Standards Institute. *Digital cellular telecommunications system (phase 2); enhanced full rate (EFR) speech transcoding*. (GSM 06.60 version 6.0.0) ETSI EN 300 726, 1997.

[10] ETSI - European Telecommunications Standards Institute. *Digital cellular telecommunications system (phase 2+); adaptive multi-rate (AMR) speech transcoding.* (GSM 06.90 version 7.2.1) ETSI EN 301 704, 1998.

[11] FANT, G. *Acoustic theory of speech production.* Mounton and Co., Gravenhage, Holanda, 1960.

[12] HONKANEN, T. et al. Enhanced full rate codec for IS-136 digital cellular system. *Proc. of IEEE International Conference on Acoustics, Speech and Signal Processing*, p. 20–24, Abril 1997.

[13] ITAKURA, F. Line spectrum representation of linear predictive coefficients. *J. Acoust. Soc. Amer.*, v. 57, p. S.35, 1975.

[14] ITAKURA, F.; SAITO, S. Analysis synthesis telephony based on the maximum likelihood method. *Proc. 6th International Congress of Acoustics*, p. C.17–C.20, Agosto 1968.

[15] ITU-T - International Telecommunication Union. *Recommendation P.56, objective measurement of active speech level.* Fevereiro 1993.

[16] ITU-T - International Telecommunication Union. *Recommendation G.729 Annex A: reduced complexity 8 kbit/s CS-ACELP speech codec.* Março 1996.

[17] ITU-T - International Telecommunication Union. *Recommendation G.729, coding of speech at 8kbit/s using conjugate-structure algebraic-code-excited linear prediction (CS-ACELP).* Março 1996.

[18] ITU-T - International Telecommunication Union. *Recommendation P.861, objective quality measurement of telephone-band (300-3400Hz) speech codecs.* Fevereiro 1998.

[19] JANKOWSKI, C. et al. NTIMIT: A phonetically balanced, continuous speech, telephone bandwidth speech database. *Proc. ICASSP*, Albuquerque, Abril 1990.

[20] KATAOKA, A.; IKEDO, J.; HAYASHI, S. LSP and gain quantization for the proposed ITU-T 8 kb/s speech coding standard. *Proc. IEEE Workshop On Speech Coding*, p. 7–8, 1995.

[21] KONDOZ, A. M. *Digital speech: coding for low bit rate communication systems.* John Wiley & Sons, 1999. (Wiley Series in Communication and Distributed Systems).

[22] KROON, P.; ATAL, B. S. On the use pitch predictors with high temporal resolution. *IEEE Trans. Signal Processing*, v. 39, p. 733–735, 1991.

[23] LAFLAMME, C.; AL. et. On reducing the complexity of codebook search in CELP through the use of algebraic codes. *Proc. ICASSP*, p. 177–180, 1990.

[24] MAKHOUL, J. et al. A mixed-source model for speech compression and synthesis. *Acoustical Society of America*, v. 64, p. 1577–1581, Dezembro 1978.

[25] MAKHOUL, J. Linear prediction: A tutorial review. *Proc. IEEE*, v. 63, p. 561–580, 1975.

[26] MARKEL, J.; GRAY, A. J. *Linear prediction of speech.* Springer-Verlag, 1976.

[27] National Communication System - Office Technology and Standards. *Federal standard 1015, telecommunications: Analog to digital conversion of radio voice by 2400 bit/second linear predictive coding.* Novembro 1984.

[28] O'SHAUGHNESSY, D. *Speech communications - human and machine.* 2a. ed. New York: IEEE Press, 2000.

[29] RABINER, L.; SCHAFER, R. W. *Digital processing of speech signals.* Prentice-Hall, 1978.

[30] RAMíREZ, M. A. *Busca de inovações e tratamento de transitórios em codificadores de voz CELP.* Tese (Doutorado) — Universidade de São Paulo, São Paulo, Brazil, 1997.

[31] RAMíREZ, M. A.; GERKEN, M. Joint position and amplitude search of algebraic multipulses. *IEEE Transactions on Speech and Audio Processing*, v. 8, n. 5, Setembro 2000.

[32] RAMíREZ, M. A.; GERKEN, M. Efficient algebraic multipulse search. *SBT/IEEE Int. Telecommunications Symposium*, v. 1, p. 231–236, 1998.

[33] RAMíREZ, M. A.; GERKEN, M. A multistage search of algebraic CELP codebooks. *Proc. IEEE Int. Conf. on Acoust., Speech, Signal Processing*, p. 17–20, 1999.

[34] SALAMI, R. et al. ITU-T G.729 Annex A: Reduced complexity 8 kb/s CS-ACELP codec for digital simultaneous voice and data. *IEEE Communications Magazine*, 1997.

[35] SCHROEDER, M. R.; ATAL, B. Code-excited linear prediction (CELP): High quality speech at very low bit rates. *Proc. ICASSP-85*, Tampa, p. 937, Abril 1985.

[36] STROBACH, P. *Linear prediction theory: A mathematical basis for adaptive systems.* Springer-Verlag, 1990. (Springer Series in Information Sciences).

[37] Texas Instruments Incorporated. *TMS320C6000 CPU and instruction set reference guide (SPRU189f).* Outubro 2000.

[38] Texas Instruments Incorporated. *TMS320C6000 optimizing compiler user's guide (SPRU187i).* Abril 2001.

[39] Texas Instruments Incorporated. *TMS320C6000 programmer's guide (SPRU198f).* Fevereiro 2001.