UNIVERSIDADE FEDERAL DO PARÁ

INSTITUTO DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Um Software de Reconhecimento de Voz para Português Brasileiro

Carlos Patrick Alves da Silva

DM - 14/2010

UFPA/ITEC/PPGEE

Campus Universitário do Guamá

Belém-Pará-Brasil

2010

UNIVERSIDADE FEDERAL DO PARÁ

INSTITUTO DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Um Software de Reconhecimento de Voz para Português Brasileiro

Autor(a): Carlos Patrick Alves da Silva

Orientador: Aldebaro Barreto da Rocha Klautau Júnior

Dissertação submetida à Banca Examinadora do Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal do Pará para obtenção do Grau de Mestre em Engenharia Elétrica. Área de concentração: **Engenharia de Telecomunicações**.

UFPA/ITEC/PPGEE

Campus Universitário do Guamá

Belém, PA

2010

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Um Software de Reconhecimento de Voz para Português Brasileiro

AUTOR(A): CARLOS PATRICK ALVES DA SILVA

DISSERTAÇÃO DE MESTRADO SUBMETIDA À AVALIAÇÃO DA BANCA EXAMINADORA APROVADA PELO COLEGIADO DO PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA, DA UNIVERSIDADE FEDERAL DO PARÁ E JULGADA ADEQUADA PARA A OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA ELÉTRICA NA ÁREA DE TELECOMUNICAÇÕES.

Prof. Dr. Aldebaro Barreto da Rocha Klautau Júnior
(Orientador - UFPA)

Prof. Dr. Antônio Marcos de Lima Araújo
(Membro - IESAM)

Prof. Dr. Eloi Favero
(Membro - UFPA)

Prof. Dr. Manoel Ribeiro Filho
(Membro - UFPA)

UFPA/ITEC/PPGEE

Agradecimentos

Primeiramente, agradeço a Deus por ter permitido que eu chegasse até aqui. Minha mãe Josemere, agradeço de coração pelo carinho e preocupação constante. Ao meu pai Gilberto, por ter acreditado e confiado em mim. Agradeço à minha irmã Camilla pelo apoio e incentivo. A minha namorada Thais pelo amor, paciência e suporte. Ao meu irmão Matheus pelos milhares de favores feitos.

Agradeço ao meu amigo e orientador Aldebaro Klautau pela oportunidade de mestrado e pelo reconhecimento. Aos amigos do do Laboratório de Processamento de Sinais (LaPS) Igor, Claudomir, Nelson, Pedro, Carol, Renan, Denise, Muller, Eduardo, Márcio, Bruno, Joarley, Diego, Diogo, Almeida, Jefferson, Yomara, Fabíola e Kelly pela ajuda, favores e encorajamento constante. Aos demais amigos do Laps agradeço pela convivência.

Aos professores membros da banca examinadora Aldebaro, Antônio Marcos, Eloi e Manoel Ribeiro pelas valiosas contribuições. Finalmente agradeço à UFPA, CNPq, CAPES e ao PPGEE pelo suporte durante todo o mestrado.

Dedico este trabalho aos meus pais,

à minha irmã e namorada

que torceram pelo meu sucesso...

Resumo

Este trabalho descreve a implementação de um *software* de reconhecimento de voz para o Português Brasileiro. Dentre os objetivos do trabalho tem-se a construção de um sistema de voz contínua para grandes vocabulários, apto a ser usado em aplicações em tempo-real. São apresentados os principais conceitos e características de tais sistemas, além de todos os passos necessários para construção. Como parte desse trabalho foram produzidos e disponibilizados vários recursos: modelos acústicos e de linguagem, novos *corpora* de voz e texto. O *corpus* de texto vem sendo construído através da extração e formatação automática de textos de jornais na Internet. Além disso, foram produzidos dois *corpora* de voz, um baseado em *audiobooks* e outro produzido especificamente para simular testes em tempo-real. O trabalho também propõe a utilização de técnicas de adaptação de locutor para resolução de ploblemas de descasamento acústico entre *corpora* de voz. Por último, é apresentada uma interface de programação de aplicativos que busca facilitar a utilização do decodificador Julius. Testes de desempenho são apresentados, comparando os sistemas desenvolvidos e um software comercial.

Abstract

This work describes a speech recognition software for Brazilian Portuguese. The main objective is to build a system for large vocabulary continuous speech recognition, able to be used in real-time applications. The concepts, characteristics and all steps needed for the construction of such systems are presented. Several resources were produced and made available: acoustic and language models, new voice and text corpora. The text corpus has been built through the extraction and automatic formatting of text from newspapers on the Internet. In addition, two voice corpora were produced, one based on audiobooks and another specifically developed to simulate real-time tests. This work also proposes the use of speaker adaptation techniques for solving the acoustic mismatch problem between speech corpora. Finally, an application programming interface is presented in order to facilitate using the open-source Julius speech decoder. Performance tests are also presented, comparing the developed system with a commercial software.

Sumário

Lista de Figuras **iv**

Lista de Tabelas **1**

1 Introdução **2**

2 Reconhecimento Automático de Voz **5**

2.1 Histórico da Tecnologia RAV 5

2.1.1 O Estado da Arte 6

2.2 Características de Sistemas RAV 7

2.2.1 Dependência de Locutor 7

2.2.2 Tipo de Fala 7

2.2.3 Tamanho do Vocabulário 8

2.2.4 Tamanho do Corpus para Treino 8

2.2.5 Dificuldades na Construção de Sistemas RAV 9

2.3 Dicionário Fonético 10

2.4 Blocos Básicos de um Sistema RAV 11

2.4.1 Front-End 12

2.4.2 Modelo Acústico 14

2.4.2.1 Cadeias Ocultas de Markov 14

2.4.2.2 Modelagem Contínua de Trifones 16

2.4.2.3 Vínculo de Estados de Modelos Trifones 16

2.4.3 Modelagem da Língua 18

2.4.3.1 Gramáticas 19

2.4.3.2 Modelo de Linguagem 19

2.4.3.3 Modelos de Linguagem N-Grama 20
2.4.3.4 Perplexidade de um Modelo de Linguagem 21
2.4.4 Decodificador 23
2.5 Ferramentas 26
2.5.1 HTK - *The Hidden Markov Model Toolkit* 26
2.5.2 SRILM - *SRI Language Modeling Toolkit* 27
2.5.3 Julius 28

3 Principais Contribuições **30**
3.1 Corpora 30
3.1.1 LapsStory 30
3.1.2 LapsBenchmark 31
3.1.3 Spoltech 32
3.1.4 O CETENFolha 33
3.1.5 LapsNews 34
3.2 Treinamento do Sistema RAV 34
3.2.1 Treino do Modelo de Linguagem 34
3.2.2 Treino do Modelo Acústico 36
3.2.2.1 Preparação dos Dados 36
3.2.2.2 Criação de Modelos Monofones 38
3.2.2.3 Criação de Modelos Trifones 41
3.2.2.4 Vínculo de Estados 42
3.2.2.5 Mistura de Gaussianas 43
3.3 Adaptação de Locutor 44
3.3.1 MLLR 46
3.3.2 MAP 47
3.4 Interface de Programação de Aplicativos 48
3.4.1 PPTController 51
3.4.2 Versão para Linux 53
3.4.3 Coruja 54

4 Resultados **56**

4.1 Adaptação de Locutor 56
4.2 Testes utilizando Gramáticas 59
4.3 Ajuste do *Beam* nos Decodificadores 60
4.4 Comparação entre Sistemas de Reconhecimento de Voz 62

5 Considerações Finais **65**
5.1 Publicações Geradas 66
5.2 Trabalhos Futuros 67

Referências Bibliográficas **67**

Lista de Figuras

2.1 Principais blocos de um sistema de reconhecimento de voz. 12
2.2 Processo de segmentação do sinal de voz e extração de parâmetros. 13
2.3 Processo de cálculo dos MFCCs. 13
2.4 Modelo HMM. 15
2.5 União de estados utilizando o método *data-driven*. 18
2.6 *Tied-States* utilizando árvore de decisão fonética. 19
2.7 Exemplo de uma rede para reconhecimento de palavras isoladas. 24
2.8 Modelos HMMs concatenados formando palavras e frases. 25
2.9 Principais estágios na criação de sistemas RAV com o HTK. 27

3.1 Tipos de codificação suportados pela ferramenta *HCopy*. 37
3.2 Funcionamento da ferramenta *HCopy*. 38
3.3 Alinhamento das HMMs com os fones realizados pela ferramenta *HERest*. . . . 40
3.4 Modelo HMM do *short-pause* que compartilha parâmetros com o modelo do silêncio. 41
3.5 Vínculo das matrizes de transição para trifones com o mesmo fone central . . . 42
3.6 Fluxograma do processo de criação do Modelo Acústico. 45
3.7 Modelo de interação com a API. 49

4.1 Variação da WER com o aumento do *beam*. 61
4.2 Variação do xRT com o aumento do *beam*. 61

Lista de Tabelas

2.1 Exemplos de transcrições utilizando trifones. 17
2.2 Relação entre tamanho do vocabulário, perplexidade e WER de acordo com [46]. 22

3.1 Características dos modelos de linguagem utilizados. 36
3.2 Exemplos dos arquivos MLF com transcrições a nível de palavra e fone. 39
3.3 Principais métodos e eventos da API. 50

4.1 Parâmetros utilizados para decodificação. 57
4.2 Resultados obtidos com os modelos acústicos do LapsStory e Spoltech. 57
4.3 Resultados obtidos com as técnicas de adaptação de locutor. 58
4.4 Parâmetros utilizados nos testes com o HDecode. 62
4.5 Parâmetros utilizados nos testes com o Julius. 62
4.6 Comparação dos sistemas utilizando modelos independentes de locutor. 63
4.7 Comparação dos sistemas utilizando modelos dependentes de locutor. 64

Capítulo 1

Introdução

Desde o surgimento dos computadores os pesquisadores buscam formas de tornar os sistemas computacionais mais inteligentes, umas das etapas desse processo é a compreensão da fala, ou seja, máquinas com a capacidade de entender e se comunicar em uma linguagem natural. Como a fala representa o principal meio de comunicação, os sistemas de reconhecimento automático de voz (RAV) e síntese de fala passaram a ser largamente estudados e aprimorados nos últimos anos.

As interfaces homem-máquina vem sendo sendo aperfeiçoadas e utilizadas em diversas aplicações, ganhando cada vez mais espaço no mercado tanto em aplicações pessoais quanto empresariais. Existem diversas aplicações para reconhecimento de voz, tais como sistemas para atendimento automático, ditado, interfaces para computadores pessoais, controle de equipamentos, robôs domésticos, indústrias totalmente à base de robôs inteligentes, etc. Muitas dessa aplicações já são utilizadas hoje, como os celulares com discagem por comandos de voz, as unidades de resposta audível, utilizadas por empresas de *call center* (atendimento) de forma que sejam faladas as opções durante o atendimento eletrônico, sistemas de ditado como o *ViaVoice* [1] da IBM e o *Dragon NaturallySpeaking* [2] da Nuance. Os portadores de necessidades especiais também são beneficiados com tais sistemas, usuários que não podem usar as mãos e deficientes visuais usam essa tecnologia para se expressarem, seja ditando textos ou realizando o controle sobre várias das funções do computador através da voz.

Dependendo da aplicação as características do sistema de reconhecimento de voz podem variar muito, desde sistemas com vocabulário limitado a poucas palavras até sistemas com vocabulário de milhares de palavras, sistemas dependentes ou independentes de locutor, etc. Essas características definem o nível de sofisticação desejada para os sistemas. Nesse contexto,

[1] http://www-01.ibm.com/*software*/pervasive/embedded_viavoice/, Visto em Janeiro, 2010

[2] http://www.nuance.com/naturallyspeaking/products/, Visto em Janeiro, 2010

se torna necessário a utilização de técnicas de modelagens que produzam resultados rápidos e precisos. Dentre as várias técnicas de reconhecimento de voz, destacam-se os Modelos Ocultos de Markov (HMM - *Hidden Markov Models*) [1] e Redes Neurais Artificiais [2], sendo a primeira mais utilizada na maior partes dos trabalhos.

Universidades e indústrias vêm tentando resolver problemas práticos da área, de forma a possibilitar o reconhecimento da fala natural. Sistemas de reconhecimento de voz com suporte a grandes vocabulários são conhecidos na literatura como LVCSR, do inglês *Large Vocabulary Continuos Speech Recognition*. Dentre as dificuldades encontradas na criação de sistemas LVCSR temos a disponibilidade de um *corpus* de voz digitalizada e transcrita grande o suficiente para treinamento do sistema, recursos como bases de textos de tamanho elevado e um dicionário fonético de amplo vocabulário. Muitos sistemas de reconhecimento de voz para o Português Brasileiro (PB) já foram produzidos [3–10], porém os sistemas eram limitados (poucos locutores ou condições de testes limitadas), não possuem resultados de testes em tempo real e não foram disponibilizados.

Em trabalhos anteriores [11, 12], buscou-se criar um sistema de referência utilizando os *corpora* Spoltech e OGI-22, porém o sistema se limitava a um reconhecimento "controlado", ou seja, treino e teste com características semelhantes (locutores, ambiente de gravação, etc). Já o trabalho atual, possui o objetivo de criar um sistema apto a trabalhar em condições de descasamento acústico entre os *corpora* de treino e teste, mais precisamente busca-se construir um sistema para ser utilizado em aplicações reais, onde não se tem controle sobre o ambiente.

Neste contexto, este trabalho se dispõe, junto com o grupo de processamento de voz *FalaBrasil* [13] criado pelo laboratório de processamento de sinais (LaPS) da UFPA, a desenvolver e disponibilizar recursos para o PB, de forma a se criar um completo sistema de referência e permitir que outros grupos de pesquisa e desenvolvedores de *software* se utilizem dos recursos criados.

Dentre as contribuições deste trabalho tem-se a construção de modelos acústico, modelos de linguagem, dicionário fonético, um *corpus* de texto baseado em extração automática de textos de jornais na Internet e dois *corpora* de voz: um baseado em *audiobooks* e outro criado para avaliação de desempenho de sistemas LVCSR, todos disponibilizados em domínio público. Outra contribuição é o uso de técnicas de adaptação de locutor para diminuir o impacto da escassez de voz digitalizada e/ou uso de poucos locutores. Os resultados a serem apresentados mostram que técnicas de adaptação de locutor são eficazes para realizar a adaptação de ambiente e diminuir o impacto do descasamento acústico. Além disso, é apresentada uma interface de programação de aplicativos (API - *Application Programming Interface*) que facilita a implementação de aplicativos baseados em reconhecimento de voz. Por último é realizada uma

comparação entre três sistemas de reconhecimento de voz: HDecode, Julius e IBM ViaVoice.

Para o desenvolvimento dos recursos utilizou-se de vários *toolkits* de domínio público tais como o HTK (linguagem C) que consiste em uma ferramenta para manipulação de modelos ocultos de Markov, o SRILM utilizado para construção de modelos de linguagem e o decodificador Julius.

O presente trabalho está organizado da seguinte forma: o Capítulo 2 aborda os principais fundamentos teóricos sobre sistemas de reconhecimento de voz, dificuldades e problemas encontrados e os principais blocos que compõe tal sistema. No Capítulo 3 são listados os principais recursos utilizados na criação do sistema proposto. O Capítulo 4 descreve os resultados obtidos. No Capítulo 5 são expostas as conclusões do trabalho e os trabalhos futuros.

Capítulo 2

Reconhecimento Automático de Voz

O reconhecimento automático de voz consiste, de uma forma geral, no processo onde o sinal de voz (analógico) é convertido em sua representação textual. O avanço da tecnologia tem permitido utilização de várias técnicas antes não utilizadas devido à limitação computacional. A evolução de sistemas de reconhecimento de palavras isoladas para sistemas de reconhecimento de fala contínua é um grande progresso na área de voz. Novas aplicações vem sendo desenvolvidas a cada ano por vários grupos de pesquisa no mundo.

Neste capítulo é descrito um breve histórico da tecnologia de reconhecimento de voz, os principais elementos e características de um sistema RAV, problemas encontrados no desenvolvimento de tais sistemas e o que se espera no futuro da tecnologia RAV. Além disso, cada um dos blocos que compõe um sistema LVCSR é descrito procurando mostrar a influência dos mesmos na tarefa de reconhecimento de fala contínua.

2.1 Histórico da Tecnologia RAV

De acordo com [14], sistemas de reconhecimento automático de voz tem sido estudados desde os anos 50 nos Laboratórios Bell, onde foi desenvolvido o primeiro reconhecedor de dígitos isolados com suporte a apenas um locutor. Na mesma época, foi introduzido o conceito de redes neurais, mas devido a muitos problemas práticos a idéia não foi seguida. Nos anos 70, os russos iniciaram estudos sobre reconhecimento de padrões, no entanto muitas das idéias eram de difícil implementação devido a limitação tecnológica da época. A técnica predominante na época era o *Dynamic Time Warping* (DTW) [15], que consiste em um algoritmo que mede a similaridade entre duas seqüências que variam no tempo, como a voz. Com o passar dos anos a pesquisa foi evoluindo e muitos problemas tecnológicos foram sendo superados,

além da globalização e popularização dos computadores que levou a um aumento no número de pesquisas na área.

Inicialmente os sistemas de reconhecimento de voz tentavam aplicar um conjunto de regras gramaticais e sintáticas à fala [16]. Caso as palavras ditas caíssem dentro de um certo conjunto de regras, o programa poderia determinar quais eram aquelas palavras, para isso era preciso falar cada palavra separadamente (sistemas de voz discreta), com uma pequena pausa entre elas. Porém, devido as características como sotaques, dialetos e regionalismos e as inúmeras exceções da língua os sistemas baseados em regras não tiveram muito sucesso.

Nos anos 80, vários pesquisadores iniciaram pesquisas para reconhecimento de palavras conectadas utilizando métodos de modelos estatísticos, sendo o maior destaque para os modelos ocultos de Markov usados para modelar séries temporais, como por exemplo a voz. Além disso um estudo mais profundo mostrou a possibilidade de aplicação de redes neurais na classificação de sinais.

2.1.1 O Estado da Arte

Atualmente, o estado da arte em reconhecimento de voz é o uso de modelos ocultos de Markov. Para a língua inglesa, já se utiliza de modelos HMM trifones e quinfones treinados com mais de 180 horas de voz, modelos de linguagem baseados em classes, aprendizado discriminativo [17], seleção de misturas de gaussianas (GMS - *Gaussian Mixture Selection*) [18], entre outras. Uma outra metodologia utilizada é o uso de HMM e redes neurais artificiais em conjunto, obtendo assim, sistemas híbridos bastante robustos na tarefas de reconhecimento de fala contínua [19].

No entanto essa não é a realidade para o Português brasileiro. Uma das principais limitações para o PB é a disponibilidade de *corpora* de grande porte. O *corpus* mais utilizado em trabalhos publicados é o Spoltech [20], que conta com mais de 400 locutores que totalizam menos de 6 horas de voz. Outro *corpus* utilizado nos trabalhos é o West Point [21], porém semelhante ao Spoltech o mesmo não é livre. Muitas tentativas de se produzir grandes *corpora* já foram feitas [7, 22, 23], porém não obtiveram sucesso e/ou os *corpora* não foram disponibilizados.

Os trabalhos sobre LVCSR para PB produzidos, em geral, são limitados ao uso de palavras isoladas ou vocabulário reduzido [3, 5, 8, 24]. Em [25] um tentativa de construir um sistema LVCSR para o PB foi realizada, porém os resultados ainda não foram satisfatórios o suficiente para se produzir aplicações. Em [9], o autor produziu um *corpus* onde todas as frases (1000 frases) foram gravadas por um único locutor, modelos acústicos e de linguagem

foram produzidos e nos testes obteve-se taxas de erro de 19% quando se utilizava frases de 9 a 12 palavras, porém o *corpus* não foi disponibilizado e o sistema produzido se limitava a modelos dependentes de locutor.

Este trabalho se situa no estado da arte em reconhecimento de voz para o PB, levando em conta a limitações de recursos disponíveis, tentando superá-los e levar aos desenvolvedores de aplicativos e pesquisadores todos os recursos necessários para utilização de reconhecimento de voz em seus trabalhos.

2.2 Características de Sistemas RAV

Muitos são os fatores que influenciam e dificultam o desempenho de reconhecedores, desde ruídos causados pelo ambiente ao sotaque do locutor. No projeto de reconhecedores vários parâmetros devem ser analisados, tais como o tamanho do vocabulário e o estilo de fala (contínua ou com intervalos) de forma a se obter um melhor desempenho. Nesta Seção são descritas as principais características dos sistemas RAV.

2.2.1 Dependência de Locutor

Sistemas de reconhecimento podem ser classificados como dependentes ou independentes de locutor. No primeiro caso o sistema é treinado para um locutor específico, sendo assim, o mesmo é apto a reconhecer, com uma boa taxa de acerto, apenas o locutor para o qual foi treinado. Sistemas independentes de locutor são capazes de reconhecer a fala de qualquer locutor, mesmo aquele que não participou do treino do sistema. Nesse último caso deve haver uma grande variedade de locutores no *corpus* de treino, o que dificulta a construção de tais sistemas. No capítulo 3 tem-se um melhor detalhamento deste assunto, onde serão descritas técnicas para adaptação de locutor.

2.2.2 Tipo de Fala

Sistemas RAV podem ser construídos para reconhecer palavras isoladas ou palavras conectadas (fala contínua). Reconhecedores de palavras isoladas tem a tendência a apresentar um resultado muito superior aos de fala contínua. Isso acontece devido ao intervalo existente entre as palavras, que é utilizado pelo reconhecedor como refêrencia de início e fim de uma palavra. Um exemplo clássico de reconhecedores de palavras isoladas são os reconhecedores de dígitos, que alcançam taxa de menos de 2% de erro para dígitos de 0 à 10.

O reconhecimento de fala contínua é uma tarefa de difícil implementação, visto que ocorrem poucas pausas durante a fala espôntanea, não se tendo informação de onde começam e terminam determinadas palavras, muitas palavras são mascaradas, encurtadas e as vezes não pronunciadas. Os efeitos co-articulatórios estão fortemente presentes nesses casos, tornando ainda mais difícil a tarefa do reconhecedor em casos como "*ele vai morrer em dois dias*" que muitas vezes é dito como "*ele vai morrerem dois dias*".

2.2.3 Tamanho do Vocabulário

Um fator crucial na precisão de um sistema RAV é o número de palavras que o mesmo é apto a reconhecer. Quanto maior o vocabulário, maiores são as chances de equívocos por parte do decodificador que fará o reconhecimento. Em sistemas com grandes vocabulários existem muitas palavras ambíguas, ou seja, possuem realizações sonoras iguais ou semelhantes. Um maior vocabulário também gera um aumento no espaço de busca, que influencia no tempo de reconhecimento. Sistemas com vocabulários de tamanho reduzido são menos sucetíveis a erros e apresentam ótimos resultados.

2.2.4 Tamanho do Corpus para Treino

Neste trabalho, o termo *corpus* (plural *corpora*) será usado para definir um grande conjunto de dados. Mais especificamente, *corpus* de texto será um conjunto estruturado de sentenças e *corpus* de voz será um conjunto de arquivos de áudio com suas respectivas transcrições.

Como será mostrado adiante o sistema RAV criado é baseado em modelos estatísticos, com etapas de treino e teste. Sendo assim, o tamanho do *corpus* utilizado para treino e teste é um fator de extrema importância para se obter resultados confiáveis. Quanto mais dados houver para treino dos modelos acústicos e de línguagem, mais bem adaptados os mesmos estarão à língua em questão. Infelizmente os *corpora* existentes para o PB são considerados de pequeno porte (poucas horas). Sistemas de grande porte necessitam de centenas de horas para treinar seus modelos acústicos e textos de milhões de linhas para os modelos de linguagem. Apesar desse déficit de dados para o PB, o sistema desenvolvido neste trabalho apresenta bons resultados e boa confiabilidade.

2.2.5 Dificuldades na Construção de Sistemas RAV

Apesar do grande avanço da tecnologia de processamento de voz, o entendimento completo da fala humana por parte do computador ainda é uma tarefa difícil e complexa de se realizar. Existem muitos problemas que afetam na precisão do reconhecedor, alguns vem sendo reduzidos e eliminados com pesquisas na área e com o surgimento de novas tecnologias. Dentre as principais dificuldades destacam-se:

- A mesma palavra pronunciada várias vezes pode apresentar diferentes formas de onda devido à articulação dos órgãos do aparelho fonador;
- Uma mesma palavra pode ser pronunciada de diferentes maneiras dependendo do tipo de locutor e da região, por exemplo a palavra "tia" dependendo do sotaque pode ser pronunciada como "tia" ou "tchia".
- Alguns locutores tendem a falar muito rápido, sendo assim, muitas sílabas ou vogais podem ser "engolidas" ou mal pronunciadas durante a fala.
- Em línguas como o português, que possui um vocabulário extenso, existem muitas palavras ambíguas, ou seja, que possuem a mesma pronúncia, essas palavras são conhecidas como homófonas. Vários são os exemplos deste tipo de palavra: "sessão/cessão", "mais/mas", "consertar/concertar", etc. É praticamente impossível para o decodificador diferenciá-las apenas com informações acústicas, esse problema é resolvido com a ajuda do modelo de linguagem.
- Dificuldade na segmentação da fala: no reconhecimento da fala natural existe grande dificuldade por parte do decidificador em se determinar as fronteiras entre os fonemas (menor unidade da fala) ou palavras, principalmente durante conversas onde ocorrem poucas pausas.
- Variações nas características da fala como ritmo, timbre e intensidade causam grandes problemas durante o reconhecimento.
- Baixa relação sinal-ruído (SNR): Durante o reconhecimento, quanto mais "limpo" for o sinal de voz mais fácil é para o decodificador diferenciar as palavras. É difícil se conseguir arquivos de boa qualidade (com pouco ruído) para treino dos modelos acústicos. Características como o ruído ambiente (vozes de outros locutores, sons de equipamentos, etc) são difíceis de se contornar, principalmente em ambientes abertos onde o ruído se torna inevitável.

O avanço na área é notável, anualmente dezenas de papers são publicados e soluções encontradas. Futuramente, é possível que o termo "reconhecimento de voz" se torne "compreensão de voz". Os modelos estatísticos que permitem que computadores reconheçam o que é dito por um usuário também podem vir a permitir que eles entendam o significado das palavras. Embora isso seja um gigantesco passo em termos de potência de computação e sofisticação dos programas, alguns pesquisadores defendem que o desenvolvimento do reconhecimento de voz oferece o caminho mais direto entre os computadores atuais e a inteligência artificial [26].

2.3 Dicionário Fonético

A conversão de uma sequência de caracteres em sequência de fones é um importante pré-requisito para serviços que envolvem reconhecimento e/ou síntese de voz. Contudo, a tarefa não é trivial e diversas técnicas de conversão vêm sendo adotadas ao longo da última década. Existe um número bem menor de estudos na área dedicados ao PB quando comparado à língua inglesa, por exemplo.

Basicamente o dicionário consiste de um conjunto de palavras com suas respectivas transcrições fonéticas. Inicialmente uma primeira versão do dicionário (UFPAdic 1.0) com 11.827 palavras foi construída a partir de versões comerciais de dicionários eletrônicos. A segunda versão do dicionário foi desenvolvida através de técnicas de aprendizado de máquina utilizando o *software* Weka [27], porém os classificadores necessitavam de etapas de alinhamento e treinamento [28] que dificultavam a construção do mesmo.

Em [29], os autores disponibilizaram um algoritmo baseado em uma estrutura de regras descritas em [30] para conversão G2P (*Grapheme To Phoneme*) com determinação de vogal tônica para PB. Uma vantagem dos conversores baseados em regras é que o alinhamento lexical (dos grafemas) não se faz necessário, visto que o *software* não precisa ser treinado para gerar suas próprias regras. Ou seja, as propostas de conversão, baseadas em critérios fonológicos pré-estabelecidos, são fornecidas ao sistema de acordo com a língua a qual o aplicativo se destina. Sua arquitetura é *self-contained*, ou seja, não carece de estágios intermediários, nem depende de outros algoritmos, para realizar análises específicas, como divisão silábica ou identificação de pluralidade. Existe uma ordem obrigatória para aplicação das regras. Primeiro são analisadas as regras consideradas mais específicas e, por último, a regra, ou caso geral, que finaliza a análise. Nenhuma análise co-articulatória entre palavras foi realizada, já que este processo de conversão G2P lidou apenas com palavras isoladas. O conversor utiliza o alfabeto fonético

SAMPA[1], do inglês *Speech Assessment Methods Phonetic Alphabet*.

Como produto do trabalho obteve-se um *software* para conversão G2P. O dicionário fonético resultante, intitulado UFPAdic.3.0, possui mais de 65 mil palavras e se encontra disponível.

2.4 Blocos Básicos de um Sistema RAV

Os sistemas de reconhecimento automático de voz atuais são em sua maioria baseados em reconhecimento estatístico de padrões [31], onde após o treinamento dos modelos, o sistema realiza a busca pela sequência de palavras mais provável, ou seja, que melhor representa o sinal de entrada. Para a realização da busca é necessário um conhecimento a priori que é representado pelos modelo acústicos e modelo de linguagem. Basicamente, faz-se uso da regra de Bayes:

$$\hat{W} = \arg\max_{W} \frac{P(O|W)P(W)}{P(O)} = \arg\max_{W} P(O|W)P(W) \tag{2.1}$$

onde se busca pela sequência de palavras W que maximiza a probabilidade condicional $\hat{W}$. Sendo O a matriz de parâmetros que reprenta o sinal acústico, tem-se $P(O|W)$ como sendo a probalilidade de se observar a matriz O dado a sequência de palavras W, valor esse fornecido pelo modelo acústico, e $P(W)$ como sendo a probabilidade da sequência de palavras W obtido através do modelo de linguagem. Como $P(O)$ não depende de W o mesmo pode ser descartado. O sistema construido foi baseado em cadeias ocultas de Markov (HMMs, *Hidden Markov Models*) que serão detalhadas adiante.

Um sistema RAV é composto por vários blocos como mostrado na Figura. 2.1 , dentre os quais temos o *Front-End* que é o bloco inicial responsável pela extração de parâmetros (*features*) do sinal de voz, o modelo acústico (MA) que busca modelar, a partir das *features*, o sinal acústico, o modelo de linguagem (ML) que tenta, a partir de textos da língua, obter as possíveis sequências de palavras a serem reconhecidas e o decodificador que, junto com os blocos já citados, realiza o processo de transcrição do sinal de voz. A estrutura e o funcionamento de cada um destes blocos é descrita a seguir.

[1] "http://www.phon.ucl.ac.uk/home/sampa/index.html", visto em fevereiro de 2010

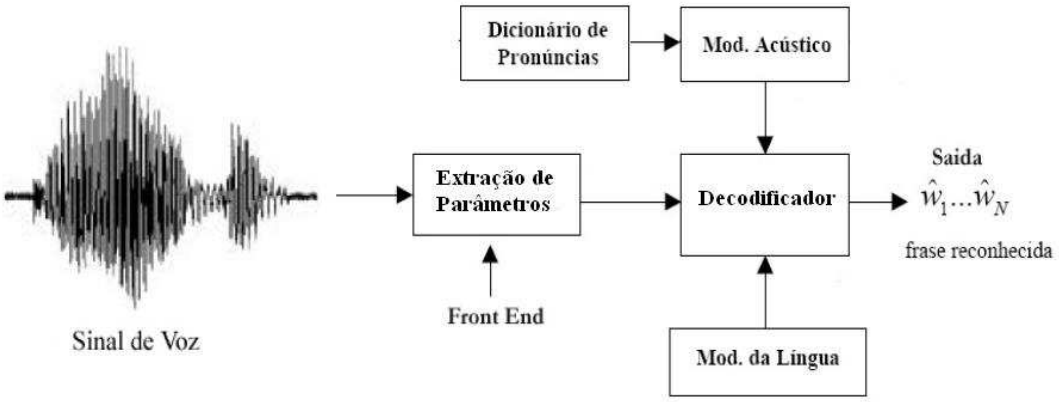

Figura 2.1: Principais blocos de um sistema de reconhecimento de voz.

2.4.1 Front-End

A primeira fase no processamento da voz é a conversão analógico-digital que traduz a onda analógica em dados digitais. Além disso, durante a conversão o sistema filtra o som digitalizado de forma a remover ruídos indesejados. Devido a natureza aleatória do sinal de voz, utiliza-lo na sua forma original não é uma prática recomendável. Sendo assim, é necessário encontrar uma outra forma de se representar o sinal acústico. Para isso faz-se uso da parametrização do sinal de voz, que nada mais é do que um processo de filtragem. Vários estudos foram realizados observando o aparelho auditivo e fonador do ser humano, como resultados tem-se diversos tipos paramétricos desenvolvidos: MFCC, PLP, FILTERBANK, LFCC. Neste trabalho foi utilizado o MFCC, do inglês *Mel-frequency Cepstrum Coefficients*. O MFCC foi proposto em 1980 por Davis e Merlmestein [32] e é bastante utilizado em tarefas de reconhecimento de voz [33] [34] [35].

A extração de parâmetros normalmente é uma transformação não-inversível, ou seja, não é possível recuperar o sinal original devido a perda de informação. Essa perda de informação torna-se necessária devido a enorme complexidade computacional que seria necessária por parte dos blocos seguintes para processar as informações por completo. Portanto, busca-se extrair as informações mais relevantes do sinal acústico, como informações que possam ser utilizadas para identificar diferenças fonéticas, além disso, a armazenagem compacta fornecida pelos parametros MFCC's torna-se um fator importante quando se trabalho com grande quantidade de dados.

Como processo inicial, tem-se a segmentação do sinal de voz em segmentos curtos (janelas) de 20 a 25 milisegundos, com o deslocamento da janela de análise de 10 milisegundos, buscando uma sobreposição de 50% entre os *frames*. Esse processo é conhecido na área de

processamento de sinais como janelamento [31]. O front-end opera em cada um desse *frames*, convertendo-os em vetores de parâmetros MFCC's. O processo de janelamento é descrito na Figura 2.2.

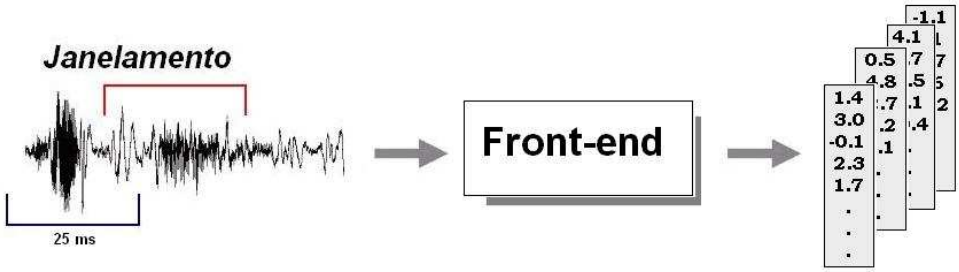

Figura 2.2: Processo de segmentação do sinal de voz e extração de parâmetros.

Após o janelamento, em cada *frame* é aplicada uma FFT (*Fast Fourier Transform*), obtendo assim seu espectro, que é passado por um conjunto de filtros triangulares na escala Mel, tal escala é a mesma do aparelho auditivo humando. Como passo seguinte tem-se uma compressão logarítmica, seguida de uma DCT (*Discrete Cosine Transform*) que diminui a correlação entre os elementos do vetor de parâmetros, tal processo é mostrado na Figura 2.3. Como saída tem-se os vários coeficientes cepstrais, porém no caso de modelamento de sistemas RAV é comum utilizar somentes os 12 primeiros coeficientes junto com a energia do sinal. A modelagem acústica assume que os vetores acústicos estão descorrelacionados dos seus vetores vizinhos, porém essa suposição não é garantida visto que os orgãos do aparato vocal humano garantem que há continuidade entre sucessivas estimativas espectrais. De acordo com Davis e Merlmestein, adicionando-se as derivadas de primeira e segunda ordem aos coeficientes e energia tem-se uma redução de tal problema. No final tem-se para cada frame uma saída com 39 parâmetros.

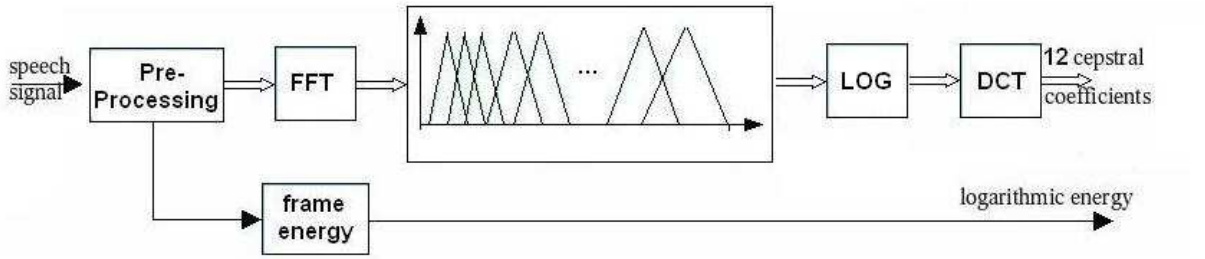

Figura 2.3: Processo de cálculo dos MFCCs.

2.4.2 Modelo Acústico

O modelo acústico busca através de *features* extraídas da voz criar um modelo matemático que represente o sinal original, de forma que dado segmentos desconhecidos os mesmos possam ser mapeados de forma correta para palavras no caso de reconhecimento de palavras isoladas, ou para fonemas no caso de reconhecimento de fala contínua. O trabalho proposto busca modelar sistemas contínuos, sendo assim são criados modelos para cada fonema[2] da língua. O conjunto de treino deve consistir de amostras contínuas, mas em geral, as fronteiras que dividem os fonemas não são conhecidas o que dificulta o processo de estimação dos modelos, porém a ferramenta HTK dispõe de ferramentas capazes de superar tal problema. No estado da arte modelos acústicos são representados por cadeias ocultas de Markov conectadas em sequência.

2.4.2.1 Cadeias Ocultas de Markov

Uma cadeia oculta de Markov [1] consiste em uma máquina de estados finita que modifica seu estado a cada unidade de tempo, mais especificamente, consiste de um modelo matemático formado por uma cadeia de estados conectados entre si, onde cada transição entre os estados possui uma probabilidade de ocorrência, além de cada estado está vinculado a um processo estocástico, que pode ser discreto ou contínuo, conhecido como processo de observação de saída. A observação de saída é a ocorrência do fenômeno sendo modelado, para o caso de sistemas LVCSR são os fonemas. A Figura 2.4 mostra o modelo básico de uma HMM com 3 estados emissores (estados 2, 3 e 4) e 2 estados que não emitem observações. Estados não emissores são utilizados para concatenação de modelos. A evolução da cadeia de estados, para o caso de reconhecimento de voz, é da esquerda para direita (*letf-right*). Temos que a_{ij} é a probabilidade de transição do estado i para o estado j que ocorre a cada tempo t, onde nos casos em que i=j um estado sofre transição para ele mesmo, e $b_i(x_t)$ é a probabilidade da observação x no tempo t dado que o estado é i. Durante o processo, não se tem infomação sobre a evolução da cadeia de estados (o que originou o termo *hidden*).

Na construção de sistemas LVCSR utiliza-se modelos de Markov de 1^a ordem [1], onde cada fonema possui seu próprio modelo HMM, representado por λ. Na fase de treino os M modelos são estimados a partir de um conjunto de observações O. Na fase de testes é apresentada uma sequência de observações O', onde o algoritmo de reconhecimento busca pelo modelo $\hat{\lambda}$ dentre os modelos $\lambda_i(i = 1, 2, ..., M)$ o que possui a maior probabilidade de ter

[2] O fonema é a menor unidade sonora que descreve um som.

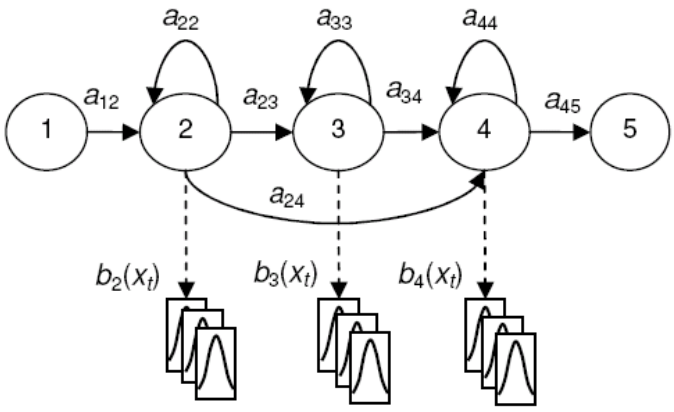

Figura 2.4: Modelo HMM.

gerado a sequência O', ou seja:

$$\hat{\lambda} = \arg\max_{\lambda} P(O'|\lambda) \tag{2.2}$$

Como mostrado na seção 2.4.1 o conjunto de observações consistem nos vetores MFCCs resultantes do bloco de extração de parâmetros. Abaixo são listados os principais parâmetros que constituem um modelo HMM.

- Número de estados N no modelo.
- Número de símbolos de observações distintas M por estado.
- Matriz de distribuição de probabilidades de transição entre os estados A, que armazena todos os possíveis valores de a_{ij}.
- Matriz de distribuição de probabilidades dos símbolos de observação no estado i, representada por B. A matriz B guarda todos os possíveis valores de $b_i(x_t)$.
- Matriz de distribuição do estado inicial π, que retorna a probabilidade do estado inicial ser o estado i.

A partir dos parâmetros mostrados um modelo completo de HMM pode ser representado por $\lambda = (A, B, \pi)$.

Na construção de modelos HMM encontramos basicamentes 3 problemas: a) calcular de maneira eficiente $P(O|\lambda)$, que seria a probabilidade da sequência de observações O ter sido gerada pelo modelo λ, este problema consiste basicamente no cálculo da equação 2.2; b) ajustar os parâmetros A, B e π de forma a maximizar $P(O|\lambda)$ para cada modelo λ_i, o que

equivale ao treinamento dos modelos HMM; c) Encontrar (durante o reconhecimento) dentre as possíveis seqüências de estados que o sistema pode seguir, aquela com maior probabilidade de ter gerado a sequência O'.

Para resolução do problema (a) utiliza-se dos algoritmos *forward/backward* [1], que consistem em algoritmos recursivos que facilitam o cálculo de $P(O|\lambda)$. No caso do problema (b) que não possui solução analíca, faz-se uso de técnicas interativas de otimização, tendo como foco o algoritmo de reestimação de *Baum-Welch* [36]. E para o problema (c) a solução é encontrada com o uso do algoritmo de *Viterbi* [37], o qual faz uso de programação dinâmica na busca da melhor seqüência de estados, além disso o algoritmo é bastante importante na decodificação onde a determinação da seqüência de estados mais provável é a chave para o reconhecimento de uma seqüência de palavras desconhecidas.

2.4.2.2 Modelagem Contínua de Trifones

Até o momento temos considerado que o modelo acústico contém um modelo HMM por fonema, com isso tem-se a idéia de que um fonema pode ser seguido por qualquer outro, o que não é verdade já que os articuladores do trato vocal não se movem de uma posição para outra imediatamente na maioria das transições de fonemas. Neste sentido, na criação de sistemas que modelam a fala fluente, busca-se um meio de modelar os efeitos contextuais causados pelas diferentes maneiras que alguns fonemas podem ser pronunciados. A solução encontrada é o uso de modelos HMM baseados em trifones. Modelos trifones consideram que um fonema em um contexto sofre influência dos fonemas vizinhos. Supondo a notação a-b+c, temos que b representa o fonema central ocorrendo após o fonema a e antes do fonema c.

Existem basicamente dois tipos de modelos trifones: os trifones intrapalavra (*internal-word triphones*) e os trifones entre-palavras (*cross-word triphones*). As diferenças entre os mesmos é que no caso do *internal-word* as fronteiras entre as palavras não são consideradas, sendo assim, menos modelos são necessários, já no caso do *cross-word*, que considera as fronteiras entre palavras, a modelagem é mais precisa, porém o número de modelos trifones cresce muito, o que dificulta o trabalho do decodificador e gera uma necessidade de mais dados para treino. A Tabela 2.1 mostra as exemplos de transcrições utilizando ambos os tipos de trifones.

2.4.2.3 Vínculo de Estados de Modelos Trifones

Como observado na Tabela 2.1, o número de modelos cresce bastante. Por exemplo se utilizarmos 39 fonemas como modelos iniciais, a migração de monofones para trifones do tipo *cross-word* será para aproximadamente 59.319 modelos. Diante disso, nos deparamos com um

Tabela 2.1: Exemplos de transcrições utilizando trifones.

	arroz com bife
Monofones	sil a R o s k o∼ b i f i sil
Internal-Word	sil a+R a-R+o R-o+s o-s k+o∼ k-o∼ b+i b-i+f i-f+i f-i sil
Cross-Word	sil sil-a+R a-R+o R-o+s o-s+k s-k+o∼ k-o∼+b o∼-b+i b-i+f i-f+i f-i+sil sil

grande problema na construção de sistemas LVCSR, insuficiência de dados para estimação dos modelos trifones, já que muitos desses trifones terão uma ou nenhuma ocorrência no *corpus*. Uma estratégia muito utilizada para redução desse problema é o compartilhamento (*tying*) de parâmetros entre os modelos. Muitos dentre os modelos trifones possuem características acústicas bastante semelhantes, sendo assim os mesmos podem compartilhar as distribuições de probabilidade em seus estados [38] [39] [40].

Existem basicamente duas técnicas para compartilhamento de estados, a primeira é conhecida como *data-driven*, na qual ocorre a clonagem dos monofones seguida pela conversão para trifones, por fim todos os estados centrais dos trifones devirados do mesmo monofone são vinculados conforme a Figura 2.5, isso é realizado levando em consideração que o contexto do trifone não afeta o estado central do modelo.

A segunda técnica consiste no uso de uma árvore de decisão fonética [41] [42], este método envolve a construção de uma árvore binária utilizando um procedimento de otimização seqüêncial *top-down* conforme [43], onde perguntas são anexadas a cada nó. As perguntas são relacionadas ao contexto fonético dos fonemas vizinhos do fonema examinado. As questões são do tipo "o fonema à esquerda é nasal ?", dependendo da resposta (sim/não) uma das possíveis direções é seguida. O objetivo da árvore e unir os estados que são acusticamente semelhantes, sendo assim, inicialmente todos os estados de um fonema são posicionados no nó raiz da árvore, de forma que ao percorrerem a árvore os estados vão sendo divididos, ao final do processo todos os estados no mesmo nó folha são agrupados. A Figura 2.6 mostra um exemplo desse processo.

As perguntas são escolhidas de forma a maximizar a verossimilhança entre os dados de treino e o conjunto resultante da união dos estados, de forma que existam dados de treino suficientes para estimar de forma robusta os parâmetros das distribuições de probabilidade gaussiana. Além disso, trifones não observados nos dados de treino podem ser sintetizados realizando uma busca na árvore pelos respectivos nós terminais de seus estados. Os resultados

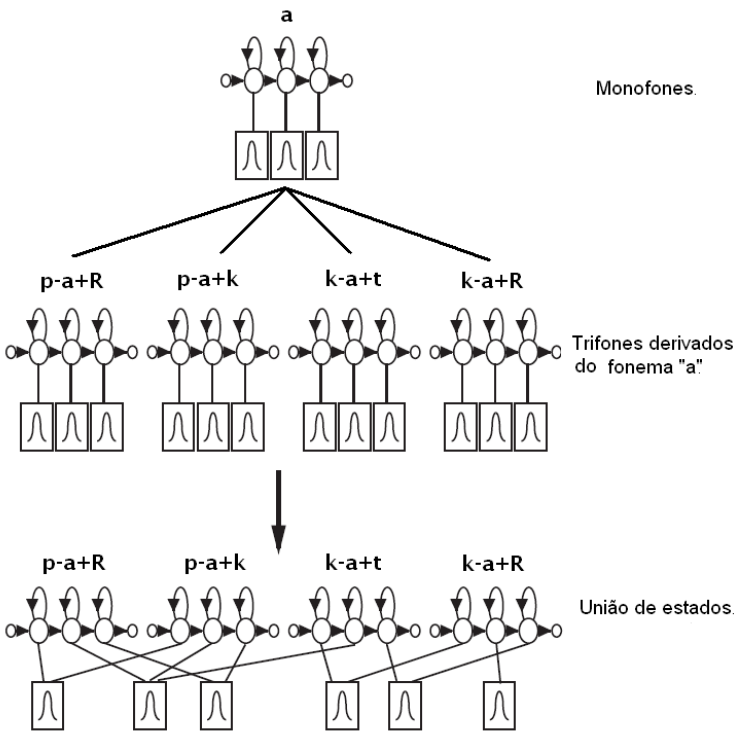

Figura 2.5: União de estados utilizando o método *data-driven*.

obtidos utilizando árvore são em geral melhores do que com o método *data-driven* como visto em [42] [34] [9].

Uma outra estratégia utilizada para se obter uma melhor estimação das HMMs é a adoção de misturas de gaussianas no modelamento das observações de saída. Dentre os métodos estatísticos conhecidos, as misturas tem-se mostrado mais robustas no modelamento de sinais variantes no tempo, como a voz. Vários trabalhos na área de voz obtiveram bom desempenho utilizando tal estratégia [44] [42] [9].

2.4.3 Modelagem da Língua

Utilizar somente informações acústicas não é suficiente para o bom desempenho de um sistema RAV, já que assim, uma palavra poderia ser seguida por qualquer outra, o que dificulta muito o trabalho do decodificador. Existem duas formas de se definir o que pode ser reconhecido: gramáticas livres de contexto ou através de modelos de linguagem.

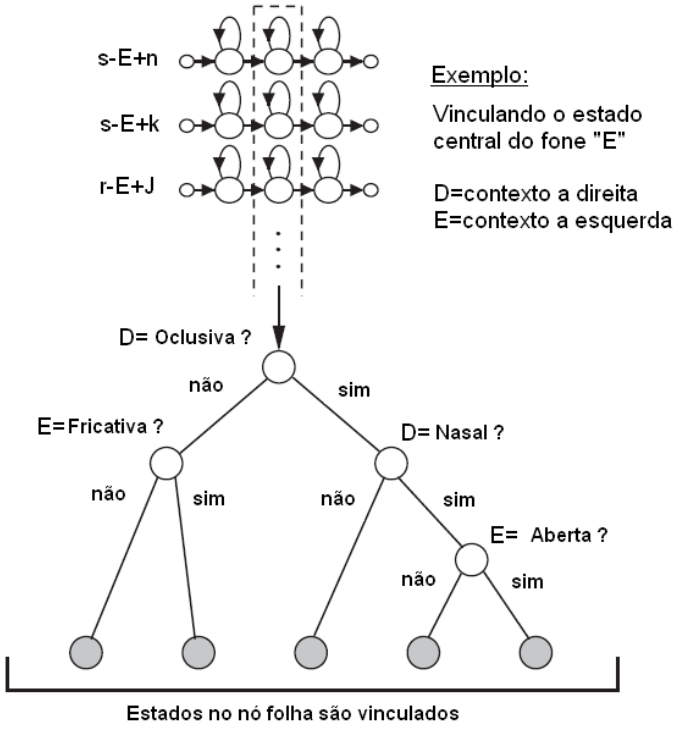

Figura 2.6: *Tied-States* utilizando árvore de decisão fonética.

2.4.3.1 Gramáticas

Uma gramática provê ao reconhecedor regras que definem o que pode ser reconhecido. Mais especificamente uma gramática consiste em um conjunto de variáveis seguidas de expressões regulares descrevendo quais palavras podem ser reconhecidas e em que ordem. Assim, o reconhecedor busca pela sequência de palavras mais provável dentro dos limites impostos pela gramática. Gramáticas são bem mais simples de se construir e apresentam melhores resultados, já que o espaço de busca durante o reconhecimento e bastante limitado.

2.4.3.2 Modelo de Linguagem

Quando se deseja construir sistemas para lidar com grandes vocabulários, se torna inviável a utilização de gramáticas, pois o trabalho necessário para se construir uma gramática com todas as possíveis frases de uma língua seria enorme. Supondo um vocabulário de tamanho V, no reconhecimento de uma seqüência de N paravras existem V^N possibilidades. Nesse sentido, faz-se uso de modelos de linguagem (ML). Esses modelos buscam caracterizar a língua, tentando capturar suas regularidades e assim condicionar as combinações de palavras durante o reconhecimento, descartando frases agramaticais. O uso de modelos de linguagem tem levado

a ganhos de desempenhos consideráveis [45], dentre as vantagens temos: a redução no espaço de busca, redução do tempo de reconhecimento e a resolução de ambigüidades acústicas [24].

O objetivo básico do modelo de linguagem é estimar, de uma forma confiável, a probabilidade de ocorrência de uma determinada seqüência de palavras $W = w_1, w_2, ...w_k$, onde k é o número de palavras na sentença. A probabilidade $P(W_1^k)$ é definida na equação 2.4

$$P(W_1^k) = P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)...P(w_k|w_1, ..., w_k - 1) \quad (2.3)$$

$$P(W_1^k) = P(w_1)\prod_{i=2}^{k} P(w_i|w_1, ..., w_i - 1) \quad (2.4)$$

2.4.3.3 Modelos de Linguagem N-Grama

É observado o alto grau de complexidade no cálculo de $P(W_1^k)$, principalmente para grandes valores de k, porém uma solução para tal dificuldade é a utilização de n-gramas, onde assume-se que a distribuição de probabilidade para a palavra w_k depende somente das $n-1$ palavras anteriores a mesma, de forma que a equação 2.4 é reescrita da forma:

$$P(W_1^k) \approx P(w_1)\prod_{i=2}^{k} P(w_i|w_{i-n+1}^{i-1}) \quad (2.5)$$

Em línguas como o inglês e o português, onde a ordem das palavras é importante, os n-gramas são bastante eficientes, uma vez que, durante o treino, os mesmos capturam simultaneamente sintaxe e semântica da língua, não havendo necessidade de criação de regras, e nem de uma gramática formal. As distribuições de probabilidade são computadas diretamente de frases prontas. A estimação pode ser executada simplesmente contando o número de ocorrências dos n-gramas. Nesse sentido, para se obter uma boa estimação é necessário um conjunto de milhares de frases.

Em reconhecimento de voz normalmente faz-se uso de bigramas ($n = 2$), porém o uso de trigramas ($n = 3$) tem apresentado melhor desempenho, pois a maioria das palavras possui uma forte dependência das duas palavras precedentes. Sendo $C(w_i)$ o número de ocorrências das palavra w_i em um texto, o treino de bigramas é feito através da contagem das ocorrências dos bigramas seguida de uma normalização por $w_i - 1$, como visto na equação 2.6. Para validar os casos em que i=1 utiliza-se marcadores de inicio e fim de sentenças, tais como $< s >$ e $< /s >$ respectivamente.

$$P(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})} \tag{2.6}$$

A mesma equação é facilmente estendida para trigramas com mostrado na equação 2.7

$$P(w_i|w_{i-2}, w_{i-1}) = \frac{C(w_{i-2}, w_{i-1}, w_i)}{C(w_{i-2}, w_{i-1})} \tag{2.7}$$

É observado que para um vocabulário de tamanho V, existem V^3 possíveis trigramas, onde muitos desses trigramas nunca ou raramente aparecerão nos dados de treino, esse problema é conhecido como problema da *frequência zero*. Na fase de reconhecimento palavras com probabilidade zero nunca serão reconhecidas pelo decodificador. Para resolução desse problema utiliza-se de técnicas de suavização (*smoothing*) [10] que buscam garantir que todas as palavras possuam probabilidades não nulas.

2.4.3.4 Perplexidade de um Modelo de Linguagem

Durante a criação de modelos de linguagem existem várias características que os diferenciam, dentre as quais destacam-se: o tamanho do vocabulário, ou seja o número de palavras presentes no modelo, e o tamanho e tipo de *corpus* utilizado para treino do mesmo. As frases utilizadas na estimação podem pertencer a domínios específicos, tais como: conversação, jornalismo, radiologia, etc. Como o modelo de linguagem busca facilitar o trabalho do decodificador, restringindo o número de possíveis palavras que podem seguir uma dada palavra melhora o seu desempenho, sendo assim, dois conceitos são introduzidos: *cross-entropy* e *perplexidade*.

Consideremos um vocabularios W e uma possível seqüência de palavras $w1, w2, ..., w_m$, a probabilidade de uma palavra w_i depende das palavras anteriores $w1, ..., w_{i-1}$. Para o cálculo da entropia cruzada (*cross-entropy*) tem-se a soma entre as probabilidades de todas as possíveis seqüências de palavras, porem considerando que tal soma é um processo ergódico, e adotando um alto valor para m, a entropia cruzada pode ser definida como:

$$H_P(T) = -\frac{1}{m} \log_2 P(w_1, w_2, ..., w_m) \tag{2.8}$$

A partir de 2.8 podemos definir a perplexidade (PP) de um modelo de linguagem como sendo:

$$PP = 2^{H_p(T)}. \tag{2.9}$$

Podemos imaginar o ML como sendo um grafo onde cada nó representa uma palavra, assim, a perplexidade seria o fator de ramificação médio em todos os pontos de decisão do grafo, ou seja, o número médio de palavras que podem seguir uma dada palavra. A perplexidade do conjunto de teste[3] avalia a capacidade de generalização do ML. É intuitivo perceber que quanto menor a perplexidade melhor tende a ser o desempenho do reconhecedor. Um exemplo de perplexidade para sistemas LVCSR é o da língua inglesa, valores encontrados variam de 50 a 250, dependendo do domínio e do tamanho do vocabulário.

Como será visto a seguir, uma medida muito utilizada para avaliação de sistemas RAV é a taxa de erro por palavra *WER* (*Word Error Rate*). Outra forma de avaliar modelos de linguagem é fixando o modelo acústico e observando como diferentes valores de PP afetam a *WER*. De acordo com [46] existe uma forte correlação entre a PP e a *WER*, como indicado na expressão 2.10

$$\text{WER} \approx -12.37 + 6.48 \log_2(PP) = -12.37 + 6.48 H_p(T) \tag{2.10}$$

A tabela 2.2 mostra resultados obtidos para vários sistemas RAV com diferentes configurações de modelos de linguagem.

Tabela 2.2: Relação entre tamanho do vocabulário, perplexidade e WER de acordo com [46].

Corpus	**Tamanho do Vocabulário**	**Perplexidade**	**WER**
TI Digits	11	11	$\sim 0.0\%$
OGI Alphadigits	36	36	8%
Resource Management (RM)	1.000	60	4%
Air Travel Information Service	1.800	12	4%
Wall Street Journal	20000	200-250	15%
Broadcast News	80.000	200-250	20%
Conversational Speech	50.000	100-150	30%

[3]O conjunto de teste consiste de frases não vistas pelo modelo na fase de treino

2.4.4 Decodificador

Os itens anteriores mostraram os modelos matemáticos para construção de modelos acústicos e de linguagem. Tem-se como etapa final a transcrição de amostras de voz desconhecidas para sua forma textual, tarefa desempenhada pelo decodificador. O processo de decodificação é controlado por uma rede de palavras construída a partir do modelo de linguagem. A rede consiste de um conjunto de nós (palavras) conectados por arcos, onde cada arco possui uma probabilidade de ocorrência (transição). Tendo em mãos tal rede, o dicionário fonético e um conjunto de modelos HMMs estimados, pode-se determinar, dada uma amostra de voz desconhecida, a probabilidade de qualquer um dos possíveis caminhos que a rede pode percorrer. A tarefa do decodificador é encontrar aqueles caminhos que são mais prováveis.

O processo de busca é descrito pela Equação 2.11, onde, tendo a sequência de observação $O_1^T = o_1, o_2 ... o_T$, busca-se pela sequência de palavras $W_1^N = w_1, w_2 ... w_N$ que maximize $\hat{W}_1^N$. O termo $P(O_1^T|W_1^N)$ pode ser expandido em função do modelo acústico, assim tem-se S_1^T como uma hipótese de seqüência de estados, onde busca-se pelo W_1^N que maximize $P(O_1^T, S_1^T|W_1^N)$ levando em conta todas as possíveis seqüências de estados. O somatório pode ser substituído por uma maximização, em um processo conhecido como aproximação de Viterbi [47], em que se utiliza apenas o caminho mais provável.

$$\begin{aligned} \hat{W}_1^N &= \arg\max_{W_1^N}\{P(W_1^N)P(O_1^T|W_1^N)\} \\ &= \arg\max_{W_1^N}\{P(W_1^N)\sum_{S_1^T} P(O_1^T, S_1^T|W_1^N)\} \\ &= \arg\max_{W_1^N}\{P(W_1^N)\max_{S_1^T}(P(O_1^T, S_1^T|W_1^N))\} \end{aligned} \tag{2.11}$$

Essa busca é realizada através de programação dinâmica (DP - *Dynamic Programming* [48]). A Figura 2.7 mostra um exemplo de reconhecimento de palavras isoladas, onde tem-se uma HMM com 3 estados para a palavra “camisa” e outra para palavra “azul”. No eixo X tem-se os *frames* vindos do bloco de extração de parâmetros. Cada um dos possíveis caminhos possui uma probabilidade de ocorrência, que é calculada pela soma do logaritmo das probabilidades de cada transição entre estados (a_{ij}) e das probabilidades dos estados emissores gerarem as respectivas observações ($b_i(x_t)$). A linha em vermelho na figura denota

um possível caminho a ser percorrido. Adotando S_1^T como sendo uma possível seqüência de estados $s_1, s2, ..., s_T$, temos que

$$P_{S_T} = \sum_{S_1^T} \log(a_{s_{T-1}s_T}) + \log(b_{s_T}(S_T)) \tag{2.12}$$

representa o cálculo da probabilidade de uma possível sequência de estados S_1^T. O algoritmo de *Viterbi* busca pela seqüência S_T com maior probabilidade P_{S_T}. Tendo em mãos as máximas probabilidades para cada HMM das palavras "azul" e "camisa", aquela que apresenta maior *score* (maior $P(O, S_T|\lambda)$) é escolhida pelo decodificador.

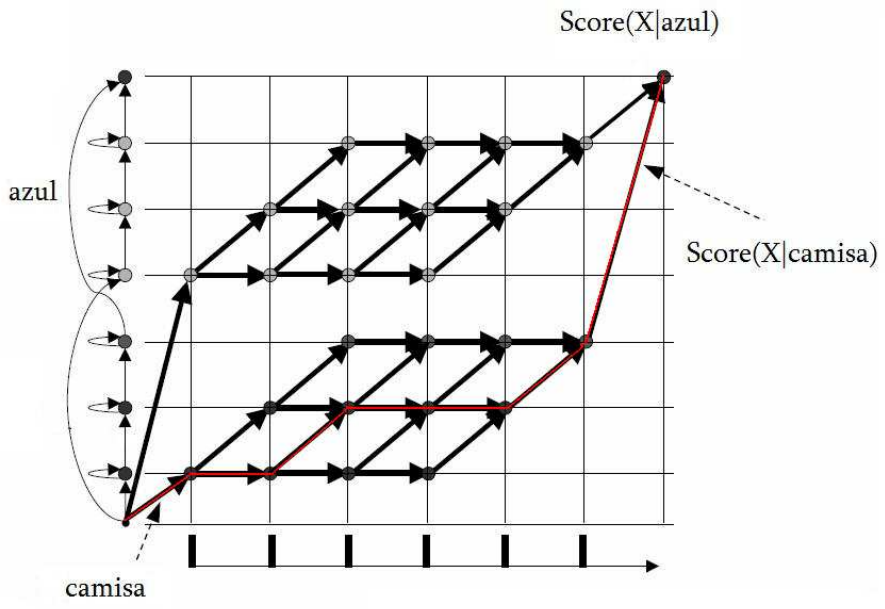

Figura 2.7: Exemplo de uma rede para reconhecimento de palavras isoladas.

Um raciocínio semelhante é feito para o reconhecimento de palavras conectadas. Inicialmente, as HMM's que representam fonemas são concatenadas de maneira a formarem palavras, e tais palavras são concatenadas criando-se frases, como mostrado na Figura 2.8. De modo geral, no reconhecimento de palavras conectadas, tem-se uma rede formada por um grande conjunto de HMMs concatenadas.

As probabilidades de transições entre os estados e transições entre palavras são determinadas pelo modelo acústico e de linguagem, respectivamente. Para encontrar o caminho com maior probabilidade o decodificador existem vários algoritmos na literatura, como exem-

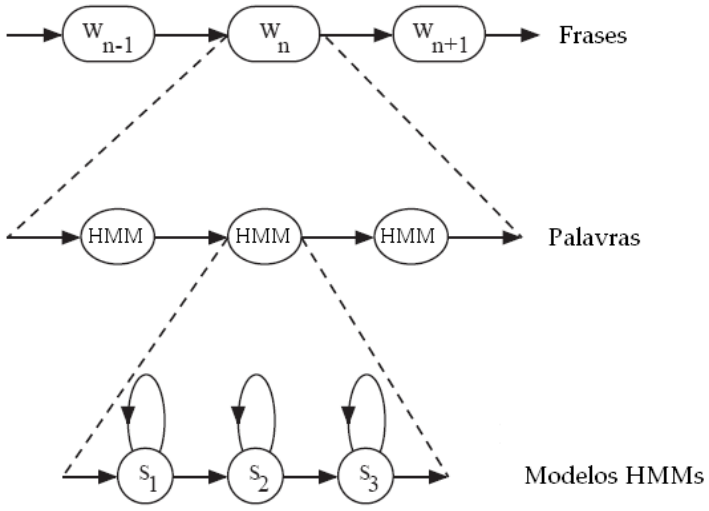

Figura 2.8: Modelos HMMs concatenados formando palavras e frases.

plo temos o algoritmo *Token Passing* [49] utilizado pela ferramenta HVite do HTK. Um *token* pode ser entendido como um trecho de um caminho dentro da rede de palavras que se extende do tempo 0 até o tempo t. No tempo 0 um *token* pode iniciar em qualquer um dos possíveis nós de início.

A cada passo, os *tokens* caminham dentro da rede através das transições parando sempre que encontram um estado emissor. Quando existem múltiplos caminhos a serem seguidos a partir de um nó, é realizada uma cópia do *token* e ambos os caminhos são explorados em paralelo. Conforme os *tokens* percorrem a rede, os *logs* das probabilidades de transição entre estados e de emissão dos estados são incrementados (*scores*). O número de *tokens* na rede Nt é limitado, sendo assim, no final de cada passo apenas os Nt melhores (com maiores probabilidades) são mantidos e os outros são descartados, tal processo é conhecido como *pruning*. O *pruning* é realizado a cada passo, descartando os *tokens* que possuem *scores* abaixo do limiar definido pelo *beam-width*. A equação 2.13 descreve como o limiar é calculado.

$$Limiar_t = BestScore_t - beam \tag{2.13}$$

Onde $Limiar_t$ e $BestScore_t$ são o limiar e *score* do melhor candidato no tempo t, respectivamente.

Por exemplo, se o melhor candidato em um tempo t possui um *score* de 1200 e utilizar-

mos um *beam* de 300, todos os candidatos com *scores* abaixo de 900 serão descartados. Como veremos na Seção 4.3, o tamanho do *beam* influência diretamente na precisão e velocidade do decodificador.

Cada *token* que passa pela rede mantém um histórico da sua rota. O nível de detalhe das informações guardadas depende do tipo de saída desejada. Normalmente apenas as seqüências de palavras são armazenadas, entretanto, é possível armazenar informações como a seqüência de estados percoridas em uma HMM e o intervalo de tempo em que a mesma ocorreu. O conjunto de informações armazenadas no histórico e o valor de Nt influenciam de maneira significativa nos recursos computacionais utilizados e no tempo de reconhecimento.

Uma vez que os dados de teste foram processados pelo decodificador, o próximo passo é a análise dos resultados. A medida de desempenho utilizada é a taxa de erro por palavra (WER) que pode ser definida como:

$$WER = \frac{S + D + I}{N} \tag{2.14}$$

onde N é o número total de palavras do conjunto de teste, S é o número de erros por substituição, D número de erros por deleção e I número de erros de inserção. O cálculo da WER é realizado facilmente através de ferramentas do HTK.

2.5 Ferramentas

Na construção do sistema RAV foi indispensável a utilização de vários *toolkits* disponíveis livremente na Internet, tais como: HTK, Julius e SRLIM.

2.5.1 HTK - *The Hidden Markov Model Toolkit*

O HTK[4] é um toolkit criado para construção e manipulação de cadeias ocultas de Markov. O HTK foi desenvolvido para pesquisa em reconhecimento de voz, porém como pode modelar qualquer série temporal, o mesmo tem sido usado em inúmeras aplicações, como por exemplo na pesquisa de síntese de voz, reconhecimento de caracteres e sequenciamento de

[4]http://htk.eng.cam.ac.uk/

DNA. O *toolkit* foi desenvolvido no laboratório de inteligência de máquina do departamento de engenharia da Universidade de Cambridge (CUED).

O HTK consiste em uma biblioteca de módulos e ferramentas disponível na linguagem C. As ferramentas proporcionam facilidades para análise da fala, treino de modelos ocultos de Markov, teste e análise de resultados. O *software* suporta implementação de HMMs contínuas e discretas, ambas com múltiplas distribuições gaussianas, o que permite a construção de sistemas com alto grau de complexidade. Para o caso de sistemas RAV, a Figura 2.9 mostra os principais estágios. No primeiro estágio as amostras de voz com suas respectivas transcrições são utilizadas pelas ferramentas do HTK para estimar os parâmetros das HMMs. No estágio seguinte, declarações desconhecidas são transcritas através das ferramentas de reconhecimento (decodificação) do HTK, sendo o HDecode a mais robusta e à utilizada nos testes deste trabalho. A versão do HTK utilizada neste trabalho foi a 3.4.1.

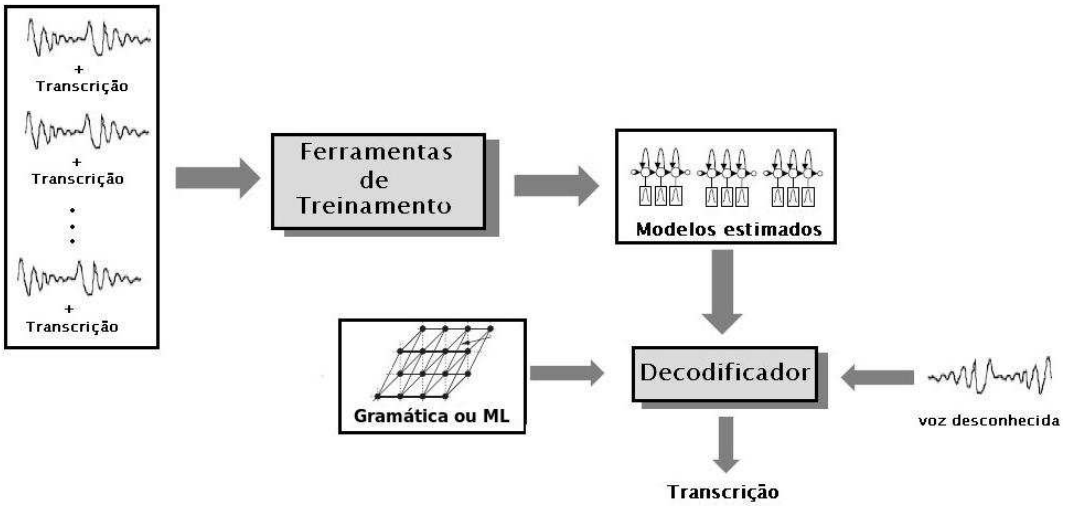

Figura 2.9: Principais estágios na criação de sistemas RAV com o HTK.

2.5.2 SRILM - *SRI Language Modeling Toolkit*

O SRILM[5] é um *toolkit* para construção e manipulação de modelos estatísticos de linguagens, criado para aplicações em reconhecimento de voz e processamento de linguagem natural. Criado pelo *SRI Speech Technology and Research Laboratory* o SRILM ainda está em processo de desenvolvimento, mas já conta com um grande número de ferramentas. O

[5]http://www.speech.sri.com/projects/srilm/, visto em Maio, 2008

seu desenvolvimento foi baseado em algumas ferramentas de construção de MLs do HTK (*CMU-Cambridge toolkit*), porém, diferente do HTK, o mesmo implementa mais opções para construção de MLs, além de várias técnicas de suavização, o que o torna muito útil na criação do sistema prosposto neste trabalho. O *software* possui código aberto, além de ser multiplataforma e já possuir uma boa documentação. Uma maior descrição do *software* pode ser vista em [50]. A versão utilizada neste trabalho foi a 1.5.10.

O SRILM consiste basicamente dos seguintes componentes:

- Um conjunto de bibliotecas em C++ que implementam de maneira eficiente algoritmos para modelos de linguagem, suporte a estruturas de dados e a uma variada gama de funções.
- Uma variedade de programas executáveis que utilizam as bibliotecas para executar tarefas padrões como treino e teste de MLs.
- Uma coleção de *scripts* que facilitam a maioria das tarefas citadas, além de uma API (*Application Programming Interface*) composta por vários *toolbox* de comandos pra criação e testes de MLs.
- Além de implementar MLs baseados em palavras (*word-based models*), o SRILM também pode implementar vários outros tipos de MLs, tais como: modelos baseados em classes (*Class-based models*), modelos baseados em eventos ocultos (*Disfluency and hidden event language models*), modelos com interpolamento dinâmico (*Dynamically interpolated LMs*), dentre outros.

2.5.3 Julius

O Julius [51] é um decodificador de alto desempenho para reconhecimento de voz. Inicialmente projetado para reconhecimento de voz em Japonês, hoje é bastante utilizado por pesquisadores e desenvolvedores de *software* em todos os idiomas.

Com suporte a modelos N-grama e HMM's dependentes de contexto, o Julius consegue realizar decodificação em tempo-real, tornado-o muito útil em aplicações práticas. Os formatos adotados foram escolhidos de tal forma a serem compatíveis com várias outras ferramentas de

modelagem, tais como HTK, CMU-Cam toolkit SLM, SRILM, etc. Sendo assim, mesmo tem suporte aos modelos acústicos e de linguagem do HTK.

O Julius é um decodificador de "duas passadas", ou seja, a decodificação é feita em dois estágios conhecidas como avanço (*forward*) e retorno (*backward*). Nesta estrátégia realiza-se primeiramente uma busca síncrona, no sentido do tempo, que visa facilitar uma segunda busca em sentido contrário, mais complexa e que requer maior esforço computacional. Esse método tem geralmente o efeito de acelerar a busca da passada de retorno, dado que o número de hipóteses a serem exploradas é bastante reduzido pela busca no sentido de avanço [9]. No primeiro passo (*forward*), é utilizado o algoritmo *frame synchronous beam search* [52] com um modelos de linguagem bigrama. No segundo passo, é utilizado um modelo de linguagem N-grama (onde N > 2), em geral utiliza-se um trigrama, onde a busca é realizada no sentido inverso (*backward*) através do algoritmo *stack decoding search* [53]. Mais detalhes sobre o processo de decoficação do Julius podem ser vistos em [54].

Dentre as principais características do Julius podemos destacar:

- Possui código aberto e bem documentado.
- Excelente desempenho, precisão e suporte a tempo-real.
- Utiliza pouca memória[6] no processo de decodificação.
- Suporta reconhecimento a partir de modelos N-grama (para qualquer valor de N), gramáticas e palavras isoladas.
- Suporta modelos de Linguagem do HTK e modelos de linguagem no formato padrão ARPA.
- Realiza alinhamento forçado a nível de palavra, fone ou estado.

A principal plataforma é o Linux, porém também funciona no Windows. Julius é distribuído com licença aberta juntamente com códigos-fonte. A versão utilizada neste trabalho foi a 4.1.5.

[6]Utiliza menos de 64MBytes de memória para modelos trigramas com 20 mil palavras, por exemplo.

Capítulo 3

Principais Contribuições

Neste capítulo serão descritos os principais recursos desenvolvido neste trabalho. Serão descritos os corpora produzidos, o processo de construção dos modelos acústico e de linguagem, as técnicas de adaptação de locutor utilizadas e a construção de uma API para desenvolvimento de aplicativos baseados em reconhecimento de voz.

3.1 Corpora

Nesta seção serão descritos os principais *corpora* de voz e texto utilizados na construção do sistema de reconhecimento de voz para o PB.

3.1.1 LapsStory

É conhecido que um dois maiores problemas na construção de sistemas LVCSR é a carência de dados para treino. Para o Inglês existem *corpora* com mais de 240 horas de voz transcrita, como o *Switchboard* [55], por exemplo.

Dentre as dificuldades encontradas em se produzir grandes *corpora*, tem-se a coleta de dados (voz) e transcrição ortográfica. Visando contornar tais problemas, foi construído um novo *corpus* de voz baseado em *audiobooks*. *Audiobooks* são livros falados disponíveis na Internet. Com os arquivos de áudio e suas respectivas transcrições (livros) tem-se uma redução considerável na tarefa de produção de um *corpus*.

Inicialmente, foram obtidos 5 livros com aproximadamente 1 hora de duração cada. Os arquivos de áudio foram reamostrados para 22.050 Hz com 16 bits. Em seguida os mesmos foram segmentados em arquivos menores, com aproximadamente 30 segundos de duração cada, e por fim transcritos. Na segunda fase do desenvolvimento foram obtidos os áudio da Constituição Brasileira e Código de Defesa do Consumidor[1], seguindo pelas mesmas etapas de segmentação e transcrição. Atualmente, o *corpus* é composto por 4 locutores do sexo masculino e 3 do sexo feminino. Os arquivos totalizam 15 horas e 42 minutos.

Uma característica da utilização de *audiobooks* é que o ambiente de gravação utilizado é bastante controlado, sendo assim, os arquivos não possuem ruído audível, têm alta razão sinal/ruído, etc. Quando tais arquivos são usados para treinar um sistema que irá operar em ambiente ruidoso, tem-se um problema com o descasamento acústico. Esse problema pode ser aliviado com técnicas de adaptação de locutor, como será descrito no Capítulo 4.

3.1.2 LapsBenchmark

Com o intuito de obter uma boa avaliação de desempenho e possibilitar a comparação de resultados com outros grupos de pesquisas, vem sendo construído um *corpus*, chamado LapsBenchmark, para testes de sistemas de LVCSR em PB.

Busca-se aqui criar um *corpus* de referência com características mais próximas da operação de um sistema de reconhecimento de voz em ambiente de escritório, ou seja, com ruído do ambiente, condicionador de ar, etc. Isso distingue o LapsBenchmark do *corpus* Laps-Story, previamente apresentado.

Para construção do *corpus* LapsBenchmark utilizou-se de frases retiradas do *corpus* CETENFolha. Mais especificamente foram utilizadas as frases descritas em [56]. Atualmente, o *corpus* possui 35 locutores com 20 frases cada, sendo 25 homens e 10 mulheres, o que corresponde a aproximadamente 54 minutos de áudio. Todas as gravações foram realizadas em computadores utilizando microfones comuns. A taxa de amostragem utilizada foi de 22.050 Hz e cada amostra foi representada com 16 bits. Como mencionado, o ambiente não foi controlado, existindo a presença de ruído nas gravações.

[1] http://www2.camara.gov.br/internet/acessibilidade/constituicaoaudio.html, visto em março de 2010

O LapsBenchhmark precisa ter seu tamanho consideravelmente aumentado para ser utilizado plenamente na realização de experimentos considerados como LVCSR. Nesse trabalho, usa-se uma estratégia que busca imitar a operação de um sistema LVCSR: o modelo de linguagem possui mais de 60 mil palavras, e o decodificador precisa lidar com alta perplexidade e descasamento acústico. Obviamente, tal estratégia permite avaliar aspectos importantes mas possui limitações. Uma dessas limitações, inerente à pouca quantidade de dados para teste, é a robustez das estimativas de taxa de erro.

3.1.3 Spoltech

O Spoltech [20] é um *corpus* para o PB desenvolvido pela Universidade do Rio Grande do Sul, Universidade de Caxias do Sul e OGI (*Oregon Graduate Institute of Science and Technology*)[2] e subsidiado pelo CNPq, no Brasil e pelo NSF nos Estados Unidos. O *corpus* é distribuido pelo LDC *Linguistic Data Consortium*[3] e OGI. O mesmo consiste de gravações (através de microfones) de 477 locutores de várias regiões do Brasil com suas respectivas transcrições fonéticas e ortográficas. As gravações consistem tanto de leituras de frases curtas quanto de respostas a perguntas (fala espontânea). No total, o *corpus* é composto de 8.080 arquivos WAV, 2.540 arquivos com transcrições em nível de palavra (TXTs sem alinhamento temporal) e 5.479 arquivos com transcrições em nível de fone (PHNs com alinhamento temporal).

O ambiente de gravação não foi controlado, sendo assim, algumas gravações foram feitas em estúdios e outras em ambientes ruidosos (feiras, escolas, etc). Os dados foram gravados a uma taxa de 44.1 KHz (mono, 16-bit). Para utilização do *corpus* foi necessário uma grande revisão e correção de vários arquivos (wavs e txts) como descrito em [11]. Além disso, para utilização do mesmo de forma compatível aos demais *corpora* citados, foi realizada uma reamostragem para 22.050 Hz. Neste trabalho foram utilizados 7.190 arquivos WAV, os quais correspondem a aproximadamente 4 horas de voz.

[2]http://cslu.cse.ogi.edu/corpora Visto em Março, 2008.

[3]http://www.ldc.upenn.edu , Visited in March, 2008.

3.1.4 O CETENFolha

O CETENFolha [57] (Corpus de Extractos de Textos Electrônicos NILC/Folha de S. Paulo) é um *corpus* de cerca de 24 milhões de palavras em português brasileiro, criado pelo projeto de processamento computacional do português, com base nos textos do jornal Folha de S. Paulo, tais textos fazem parte do *corpus* NILC/São Carlos, compilado pelo Núcleo Interinstitucional de Lingüística Computacional (NILC). Para utilização do *corpus* foi necessário uma grande formatação, de forma que pudesse ser usada pela ferramenta HTK e SRILM. A formatação consistiu de:

- Remoção de pontuação e *tags* presentes nos textos ([ext], [t], [a], entre outras).
- Conversão de letras maiúsculas para minúsculas.
- Conversão de números para forma extensa.
- Expansão de siglas (PM, MEC, PT , etc).
- Correção de palavras.

Um trecho do *corpus* antes da formatação é mostrado abaixo:

```
<ext id=165691 cad="Mundo" sec="pol" sem="94a">
<s> <t> Recuo dilui dividendo político do presidente </t> </s>
<s> <situacao> De Washington </situacao> </s>
<p>
<s> No primeiro momento, a «Operação Sustentar a Democracia»
traz dividendos políticos para Bill Clinton . </s>
<s> Mas o futuro é incerto . </s>
</p>
<p>
<s> O Senado tem uma <<caixa preta>> de R$ 2 milhoes </s>
</p>
</ext>
```

Após a formatação o mesmo trecho:

```
<s> Recuo dilui dividendo político do presidente </s>
<s> situacao De Washington </s>
<s> No primeiro momento a operação sustentar a democracia
traz dividendos políticos para bill clinton </s>
<s> Mas o futuro é incerto </s>
<s> o senado tem uma caixa preta de dois milhoes de reais </s>
```

Como resultado o *corpus* formatado possui 1,5 milhões de frases e aproximadamente 23 milhões de palavras.

3.1.5 LapsNews

De forma a expandir a quantidade de dados utilizado no treino dos modelos de linguagem, foi construído o *corpus* LapsNews. A idéia é complementar os textos do CETENFolha.

A construção do LapsNews vem sendo feito através de um processo totalmente automatizado de coleta e formatação diária de jornais disponíveis na Internet. Inicialmente o *software* coletas as páginas HTML dos jornais, obtem-se os textos dos jornais através de tags específicas, e por último formata-se o texto de forma semelhante a realizada no *corpus* CETENFolha.

Aproximadamente 9 meses de textos coletados e processados automaticamente encontram-se disponíveis em [13], os quais correspondem a aproximadamente 820 mil frases.

3.2 Treinamento do Sistema RAV

Nesta seção são descritos os processos de treino dos modelos acústico e de linguagem. Todos os modelos e *scripts* utilizados encontram-se disponíveis.

3.2.1 Treino do Modelo de Linguagem

O primeiro bloco construído foi o modelo de Linguagem. Para treino dos modelos de linguagem, foram utilizadas ferramentas do *toolkit* SRILM. Apesar de o HTK possuir um conjunto de ferramentas destinadas a essa tarefa (*HLMToolkit*), o SRILM possui uma

variedade muito maior de configurações de treino que a torna mais eficiente para essa tarefa, principalmente na construção de modelos de linguagem do tipo n-grama.

Para construção do modelo de linguagem foi realizada a junção de todas as frases dos *corpora*: CETENFolha, Spoltech, OGI-22, WestPoint, LapsStory e LapsNews de forma a se obter o maior número de frases possível. Após a junção e formatação (remoção de pontuações, expansão de siglas, etc), o arquivo total de treino do ML contabilizou mais de 2,3 milhões de frases. Para o cálculo da perplexidade, utilizou-se de 10 mil frases não vistas na fase de treino. O formato utilizado para treino e teste foi o XML, que possui marcadores de inicio e final de sentença ($< s >$,$< /s >$), tais marcadores são contabilizados no treinamento do modelo tal como palavras, um exemplo foi descrito na Seção 3.1.4.

Para treino utilizou-se a ferramenta *ngram-count* do SRILM, a qual pode ser usada para gerar ou manipular n-gramas. O *software* primeiramente conta o número de ocorrência dos bigramas e/ou trigramas a partir da leitura de um arquivo de texto, um dos parâmetros repassados à ferramenta é o vocabulário, visto que palavras encontradas nas frases de treino que não estejam no vocabulário (*OOV - Out of Vocabulary*) não terão suas freqüências contabilizadas, por seguinte tem-se na saída um modelo de linguagem n-grama no formato ARPA contendo o resultado das contagens. Dentre as várias opções para estimação de MLs, temos como as principais: a escolha do vocabulário e o método de suavização que será utilizado. Vários métodos de suavização foram testados, e o que apresentou melhor resultado (menor perplexidade) foi o modelo de desconto absoluto de Kneser-Ney [58]. Para todos os modelos construídos foram contabilizadas apenas as palavras contidas no dicionário (65.531 palavras) e para cálculo da perplexidade fez-se uso da ferramenta *ngram* (SRILM). No Capítulo 4, serão mostrados resultados com o HDecode e Julius. Para os testes com o HDecode foram utilizados apenas modelos trigrama. Como descrito na seção 2.5.3, o decodificador *Julius* necessita de dois modelos de linguagem, um modelo bigrama e outro modelo trigrama. Uma particularidade é que, como a segunda passada do Julius é feita de trás para frente (*backward*), o modelo trigrama deve ser treinado com as frases invertidas como mostrado abaixo.

```
</s> inaceitável é políticos alguns de antiética postura a <s>
</s> ar no arriscadas piruetas dão trapezistas os <s>
</s> chuva a com confundido é vezes as manhã da orvalho o <s>
```

A tabela3.1 mostra as características dos modelos de linguagem obtidos.

Tabela 3.1: Características dos modelos de linguagem utilizados.

Número de frases usadas no treino	2.345.696
Número de palavras presentes no *corpus*	41.347.694
Perplexidade - Trigrama	166.0
Perplexidade - bigrama	253.9

3.2.2 Treino do Modelo Acústico

O processo de treino foi realizado com ferramentas do *software* HTK, junto com vários *scripts* Java, necessários para criação de arquivos específicos, como será mostrado adiante.

3.2.2.1 Preparação dos Dados

Como descrito no Capítulo 2, o primeiro bloco na construção de um sistema LVCSR consiste na extração de parâmetros do sinal de voz. Para realizar tal tarefa, fez-se uso da ferramenta *HCopy* do HTK. O *HCopy* aceita como entrada vários tipos de arquivos, além do formato WAV, e os converte para o tipo paramétrico desejado. A Figura 3.1 mostra todos os possíveis formatos de entrada e saída suportados.

Como já citado, utilizou-se dos paramêtros MFCCs como representação acústica. No processo de extração, para cada arquivo de entrada um outro correspondente contendo os vetores de parâmetros é criado (vide Figura 3.2). Para utilização da ferramenta é necessário um arquivo de configuração que descreve todas as características desejadas na conversão do sinal. O arquivo de configuração usado na criação do modelo acústico é mostrado abaixo:

```
Arquivo de configuração: hcopy-wav.conf

TARGETKIND = MFCC_E_D_A_Z   # tipo paramétrico
TARGETRATE = 100000.0     # taxa de amostragem em centésimos de nano segundos
```

Formatos de entrada	*Formatos de saída*										
	WAVEFORM	LPC	LPREFC	LPCEPSTRA	IREFC	MFCC	FBANK	MELSPEC	USER	DISCRETE	PLP
WAVEFORM	√	√	√	√	√	√	√	√		√	
LPC		√	√	√	√					√	
LPREFC		√	√	√	√					√	
LPCEPSTRA		√	√	√	√					√	
IREFC		√	√	√	√					√	
MFCC						√				√	
FBANK						√	√			√	
MELSPEC						√	√	√		√	
USER									√	√	
DISCRETE										√	
PLP										√	√

Figura 3.1: Tipos de codificação suportados pela ferramenta *HCopy*.

```
SAVECOMPRESSED = F        # Salvar o arquivo de saída com compressão
SAVEWITHCRC = F           # Adiciona um "checksum" a saída do arquivo
WINDOWSIZE = 250000.0     # Tamanha da janela de análise
USEHAMMING = T            # Utilizar o janelamento de Hamming
PREEMCOEF = 0.97          # Indica o coeficiente de pré-ênfase
NUMCHANS = 26             # Número de canais no banco de filtros
CEPLIFTER = 22            # Coeficiente de liftering cepstral
NUMCEPS = 12              # Número de parâmetros ceptrais
ENORMALISE = T            # Normalização do log da energia do sinal
SOURCEFORMAT = WAV        # Formato do arquivo de entrada
SOURCEKIND  = WAVEFORM    # Tipo de codificação de entrada
ZMEANSOURCE = T           # Fonte com formato de onda com média zero
SOURCERATE  = 453         # valor definido como: 10000/fs
```

Como saída temos 13 parâmetros (12 coeficientes cepstrais mais a energia), dos quais são extraídos as primeira e segunda derivada resultando em 39 parâmetros representando cada *frame* do sinal.

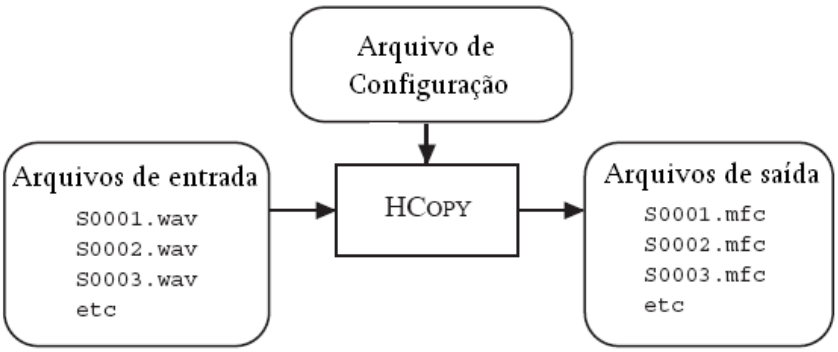

Figura 3.2: Funcionamento da ferramenta *HCopy*.

3.2.2.2 Criação de Modelos Monofones

A estratégia utilizada na construção de modelos acústicos com a ferramenta HTK é que as HMMs devem ser refinadas gradualmente. Nesse sentido, iniciamos com um simples conjunto de modelos, baseados em fones, com uma única gaussiana por mistura e independente do contexto, então gradualmente refina-se tais modelos, os expandindo para modelos dependentes do contexto (trifones) e usando distribuições com múltiplas misturas de componentes gaussianas.

Primeiro, um conjunto inicial de modelos (protótipos) é criado, cada modelo HMM contem 3 estados emissores e 2 não emissores utilizando a estrututa *left-right*. O sistema construído utiliza 38 fones mais um fone que representa o silêncio. Após a criação dos protótipos, a ferramenta *HCompV* é utilizada para calcular médias e variâncias globais, que são replicadas para todos os modelos HMMs criados, sendo assim, todos os modelos iniciais possuem estados (médias e variâncias) e matrizes de transição iguais.

Os modelos HMM são salvos no formato MMF (*Master Macro File*), que consiste de 2 arquivos: o arquivo *hmmdefs* que armazena todos os fones com seus respectivos estados e matrizes de variância, e o arquivo *macros* que guarda o tipo paramétrico utilizado e a variável *vFloors* gerada pelo *HCompV*, essa variável armazena o valor base de variância, que é igual a 0.01 vezes o valor da variância global.

Para treino dos modelos faz-se uso da ferramenta *HERest*, a mesma realiza o treinamento *embedded* do sistema através do algoritmo de *Baum-Welch*. Para utilização dessa ferramenta é necessário a criação de um arquivo MLF (*Master Label File*) que contém as

transcrições fonéticas dos arquivos de treino. Desse modo, inicialmente cria-se o arquivo MLF com transcrição a nível de palavra (*words.mlf*), tal arquivo é criado através de um *script* Java que tem como entrada a lista de todos os arquivos com transcrições ortográficas. Então a ferramenta *HLEd*, junto com o dicionário fonético, realiza a conversão das palavras para sua respectiva representação fonética (*phones.mlf*), como mostrado na tabela 3.2.

Tabela 3.2: Exemplos dos arquivos MLF com transcrições a nível de palavra e fone.

words.mlf	**phones.mlf**
"*/BR-00310.work.lab"	"*/BR-00310.work.lab"
estudante	sil
.	e
	s
	t
	u
	d
	a∼
	t
	i
	sil
	.

O *HERest* segmenta os dados (MFCC's) uniformemente de acordo com o MLF (*phones.mlf*), então realiza a união das HMMs como mostrado na Figura 3.3 e executa simultâneamente uma única re-estimação de Baum-Welch sobre todo o conjunto de modelos HMMs.

Para as amostras de treino os correspondentes modelos HMMs são concatenados e então repassados ao algoritmo *forward/backward* que acumula as estatísticas de ocupação de estados, médias, variâncias, entre outros, para cada HMM. Quando todos os dados de treino são processados, as estatísticas acumuladas são usadas para computar os parâmetros das HMMs re-estimadas.

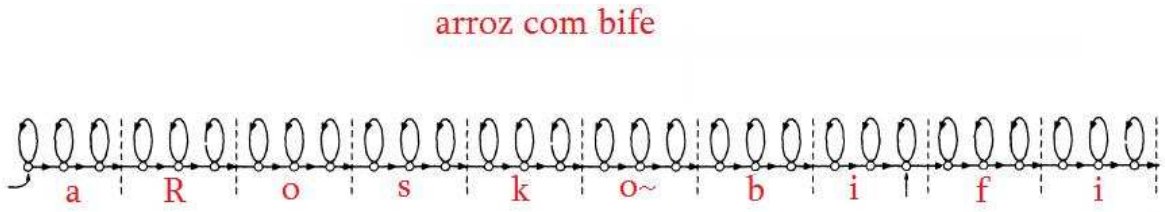

Figura 3.3: Alinhamento das HMMs com os fones realizados pela ferramenta *HERest*.

A probabilidade de transição entre dois estados não emissores é zero, de forma que é obrigatório a ocorrência de pelo menos uma observação (estado emissor), caso ocorra algum tipo de ruído durante o reconhecimento o mesmo é contabilizado, diminuindo o desempenho do decodificador. Um outro problema é que, como o sistema se propõe a modelar a fala continua, dificilmente ocorrerá silêncio entre as palavras, o que dificulta a identificação de onde começa e termina uma palavra, para redução desses problemas faz-se uso do modelo *short-pause* (também conhecido como *tee-model*) [42], que consiste em um modelo HMM composto somente de 1 estado emissor e 2 não emissores, o modelo possibilita a não emissão de nenhuma observação, representado pela transição entre os estados não emissores, isso torna o modelo mais robusto, permitindo que o mesmo absorva ruídos impulsivos. Para criação do modelo *short-pause*, representado pela sigla *sp*, faz-se uma cópia do estado central do modelo do silêncio (*sil*), seguida por um vínculo do estado central do modelo do silêncio com o estado emissor do modelo *short-pause*, de forma que, durante o treino, os mesmos compartilhem os mesmos parâmetros, como mostrado na Figura 3.4. O Vínculo de estados é realizado através da ferramenta *HHEd*. Após o vínculo de estados, faz-se uma modificação no arquivo de treino *phones.mlf*, onde se insere o fone *sp* na fronteira entres as palavra.

Além disso, é necessária a inserção da HMM *sp* no final de cada palavra do dicionário fonético, como mostrado abaixo. Assim, rápidas pausas podem ser identificadas, facilitando a identificação do fim de uma palavra durante o reconhecimento.

```
...
absolutamente    a b s o l u t a m e~ t i sp
absolutas        a b s o l u t a s sp
absolutismo      a b s o l u t i z m u sp
absoluto         a b s o l u t u sp
absolutos        a b s o l u t u s sp
...
```

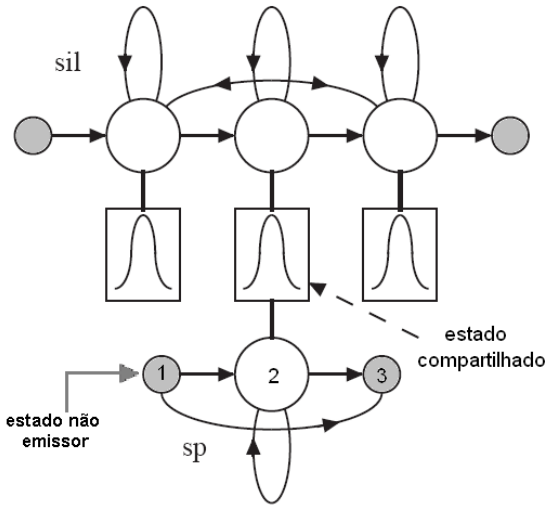

Figura 3.4: Modelo HMM do *short-pause* que compartilha parâmetros com o modelo do silêncio.

3.2.2.3 Criação de Modelos Trifones

Como etapa seguinte, tem-se a adaptação das HMMs para modelos dependentes de contexto, para isso é utilizada a ferramenta *HLEd* do HTK. Para utilização do *HLEd* é necessário um *script* (*mktri.led*) contendo as modificações a serem realizadas para a expansão de modelos monofones para modelos trifones, além disso, a ferramenta modifica o arquivo MLF utilizado para treino, de forma a representar modelos dependentes de contexto. Como mostrado na subseção 2.4.2.2, modelos trifones podem ser do tipo *cross-words* ou *word-internal*. Nos testes realizados foram utilizados trifones do tipo *cross-word*, já que modelam melhor as transições entre as palavras.

Após a criação do MLF trifone e da lista de trifones, faz-se uso da ferramenta *HHEd*, a qual clona todos os trifones e realiza o vínculo das matrizes de transição. Parte do *script* (*mktri.hed*) é mostrado abaixo.

```
CL trifone
TI T_v {(*-v+*,v+*,*-v).transP}
TI T_a {(*-a+*,a+*,*-a).transP}
TI T_i {(*-i+*,i+*,*-i).transP}
TI T_s {(*-s+*,s+*,*-s).transP}
```

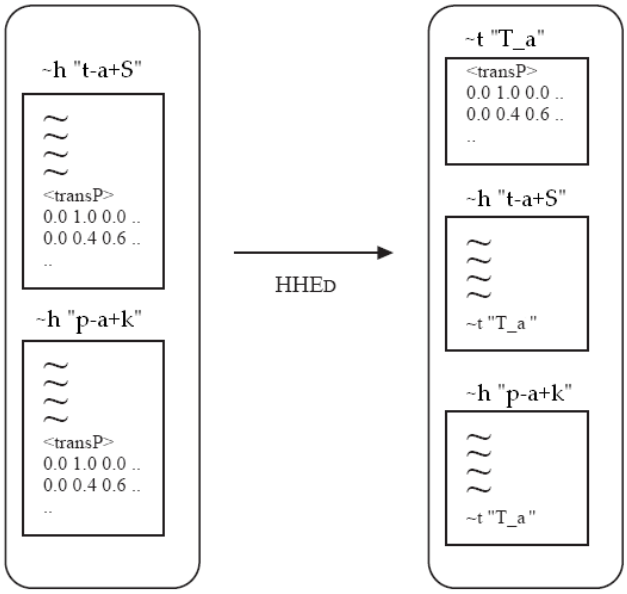

Figura 3.5: Vínculo das matrizes de transição para trifones com o mesmo fone central

O comando CL tem como argumento o arquivo com a lista de todos os trifones, o comando realiza a clonagem de cada trifone, onde todos os trifones herdam os 3 estados do seu monofone central, seguido pelo comando TI que vincula todas as matrizes de transição para cada conjunto de trifones que possuem o mesmo estado central, como mostrado na Figura 3.5. Após essas modificações, novamente os modelos passam pela re-estimação de Baum-Welch com a ferramenta *HERest*.

3.2.2.4 Vínculo de Estados

Como próximo passo na construção do modelo acústico tem-se o vínculo de estados dos trifones, esse vínculo permite o compartilhamento de dados, de forma a se obter uma estimação mais robusta dos parâmetros das HMMs. Com o intuito de otimizar os modelos trifones, o método escolhido foi o vínculo através de árvore de decisão fonética, como descrito na subseção 2.4.2.3. A árvore de decisão fonética usada neste trabalho foi construída através de uma vasta pesquisa sobre classificação de fonemas, procurando agrupar fones com características semelhantes, um trecho da árvore é visto abaixo.

```
QS  "R_V-Oral"      { *+a,*+e,*+i,*+o,*+u,*+O,*+E }
QS  "R_V-Front"     { *+i,*+E,*+e }
QS  "R_V-Central"   { *+a }
```

```
QS  "R_V-Back"      { *+u,*+o,*+O }
QS  "R_V-Aberta"    { *+a,*+E,*+O }
```

Cada comando QS define uma questão, e cada questão é definida para os contextos a esquerda e a direita do trifone. Por exemplo, a primeira questão da árvore *R_V-Oral* é verdadeira se o contexto a direita corresponde á alguma dentre as vogais orais: *a, e, i, o, u, O* ou *E*. Uma descrição completa dos comandos do *script* pode ser vista em [42].

Para aplicação da árvore é utilizada a ferramenta *HHEd*. Tendo como entrada os arquivos de definição de HMMs, (*macros* e *hmmdefs*), e o *script* contendo a árvore (*tree.hed*), a ferramenta tem como saída o arquivo *tiedlist* contendo todos os trifones, junto com os respectivos rótulos mostrando com quais outros trifones os mesmos compartilham estados. A redução no número de trifones é notável, por exemplo, para o modelo acústico construído, tem-se aproximadamente 59.321 trifones lógicos, que passam a ser 8.324 físicos após a aplicação da árvore, onde apenas as HMMs físicas é que são modificadas durante o treino.

3.2.2.5 Mistura de Gaussianas

Como etapa final tem-se a implementação de múltiplas componentes gaussianas nas HMMs. O mecanismo utilizado para realizar tal tarefa é o comando *MU* da ferramenta *HHEd*, o qual incrementa o número de componentes gaussianas, tal processo é conhecido como *mixture splitting*. Normalmente o número de gaussianas é incrementado gradualmente, onde cada incremento é seguido por várias re-estimações através do *HERest*, esse processo se repete até atingir o número desejado. A ferramenta *HHEd* tem como argumento o arquivo de definições de HMM, seguido pelo *script* que contém o comando MU, como exemplificado abaixo:

```
MU 2 {*.state[2-4].mix}
```

O comando incrementa, para todas as HMMs, o número de componentes gaussianas dos estados 2, 3 e 4, que são os estados que emitem observação. Depois de vários testes e pesquisas, foi observado que a melhor seqüência de incremento foi de 1 gaussianas para 2, seguido de incrementos duplos (1 gauss/mix, 2 gauss/mix, 4, 6, 8, etc). Neste trabalho, utilizou-se 14

componentes gaussianas, esse número foi obtido após vários testes, onde com valores superiores a este o sistema sofre *overfitting*, onde os resultados estabilizam. A Figura 3.6 mostra o fluxograma do processo de criação do modelo acústico. O *script* completo utilizado para criação do modelo acústico se encontra disponível em [13].

3.3 Adaptação de Locutor

As seções anteriores descreveram as etapas necessárias para construir um sistema LVCSR de voz independente de locutor. Como descrito na Seção 2.2.1 sistemas independente de locutor são produzidos utilizando dados de vários locutores, tais sistemas apresentam ótimo desempenho em tarefas de comando e controle para qualquer locutor, mesmo para aqueles não vistos na etapa de treino. Porém em tarefas de ditado sistemas independentes de locutor apresentam desempenho razoável (taxas de erro acima de 20%), isso ocorre devido a maior complexidade e aos diferentes biotipos vocálicos humanos, sendo necessária, na maioria das vezes, uma etapa de adaptação de locutor.

Um sistema adaptado não é um sistema dependente de locutor por completo, mas sim um sistema que combina o conhecimento geral dado pelo sistema independente com informações mais específicas do novo locutor, obtidas através dos seus dados de adaptação [59]. Em geral, técnicas de adaptação são utilizadas em modelos independentes de locutor bem treinados, ou seja, modelos treinados com dezenas ou centenas de horas de voz, porém para a fase de adaptação apenas uma pequena quantidade de dados é necessária, em geral, alguns minutos de voz.

Uma vantagem da adaptação de locutor é que nem todos os parâmetros do modelo independente necessitam ser reestimados. De acordo com [60] a adaptação das probabilidades de transição e dos pesos das gaussianas (quando se utiliza misturas) não proporcionam ganhos relevantes nos resultados. Também, segundo [15] a adaptação das matrizes de covariância quando se possui uma quantidade limitada de dados, não apresenta ganho significativo nos resultados. Sendo assim, as técnicas utilizadas neste trabalho realizam adaptação somente nas médias das gaussianas dos modelos HMM. Os parâmetros que não são adaptados são herdados do modelo independente, isso diminuiu consideravelmente a carga computacional requerida no

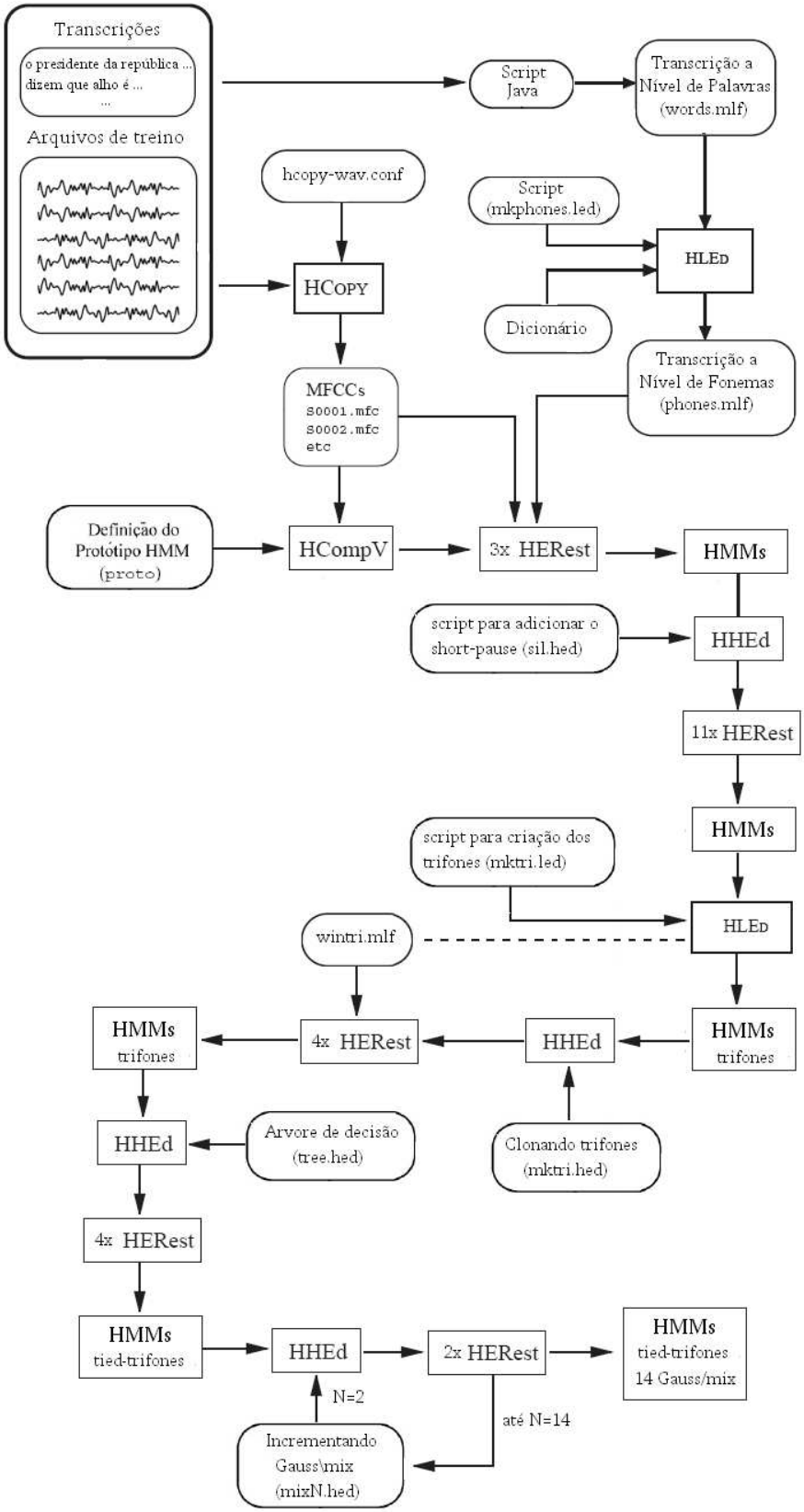

Figura 3.6: Fluxograma do processo de criação do Modelo Acústico.

processo de adaptação, o que torna esse método mais aplicável em sistemas reais [61].

Neste trabalho são apresentadas duas técnicas de adaptação, a técnica de regressão linear de máxima verossimilhança (de *Maximum Likelihood Linear Regression*, MLLR) e a técnica conhecida como MAP (*Maximum a Posteriori*).

Técnicas de adaptação de locutor podem ser utilizadas basicamente de dois modos:

- Adaptação supervisionada: a transcrição da voz é conhecida, ou seja, tem-se conhecimento do que está sendo falado.
- Adaptação não supervisionada: não se tem a transcrição do que está sendo falado, o sistema reestima os modelos baseado no que foi reconhecido, mesmo quando ocorre erros no reconhecimento.

Para adaptação foi utilizado o *toolkit* HTK. O HTK suporta os dois modos de adaptação citados. A ferramenta **HERest** realiza adaptação supervisionada *offline* utilizando MLLR e/ou *maximum a-posteriori* (MAP), enquanto que a adaptação não supervisionada é suportada apenas pelo **HVite**[4], onde simultâneamente (*online*) é realizado o reconhecimento e adaptação do modelo acústico.

3.3.1 MLLR

A técnica MLLR, desenvolvida por [32], estima um grupo de transformações lineares para os parâmetros de média e de convariâncias das gaussianas. O efeito dessas transformações é de deslocar as componentes de média e de variância tornando-os mais semelhantes aos dados do novo locutor.

De acordo com [61, 62] uma das vantagem de se utilizar transformações lineares é que uma mesma transformação pode ser utilizada por várias gaussianas do modelo HMM, o que permite uma convergência mais rápida. Um modo de classificar as matrizes de transformação é a utilização do processo conhecido como *Regression Class Tree* [63]. Utilizando essa técnica cada matriz de transformação (classe) é aplicada a um certo grupo de componentes gaussianas. As gaussianas são agrupadas de acordo com semelhanças acústicas, assim componentes

[4]O HVite suporta apenas a técnica MLLR para adaptação.

semelhantes são transformadas de modo semelhantes. O número de classes utilizadas depende da quantidade de dados disponível, quanto mais dados maior será o número de classes. Mais detalhes pode ser vistos em [42].

Dada uma gaussiana do modelo HMM, sua média é adaptada da seguinte forma:

$$\hat{\mu} = W\xi \tag{3.1}$$

onde W é a matrix de transformação $n \times (n + 1)$, sendo n a dimensão do vetor de observação, e ξ é o vetor de médias,

$$\xi = [w\ \mu_1\ \mu_2\ ...\ \mu_n]^T \tag{3.2}$$

onde w é conhecido com *bias offset*[5] e tem o valor fixado em 1 no HTK.

A matriz W é estimada de forma a maximizar a semelhança entre o novo modelo e os dados de adaptação através do algoritmo *Expectation-Maximization*(EM).

Por *default* o HTK armazenada essa matriz de transformação em um arquivo separado do modelo acústico original, sendo assim, no momento da decodificação (via **HDecode** ou **HVite**) o HTK carrega o modelo acústico independente de locutor e a matriz de transformação, ou seja, a transformação linear só ocorre no momento da decodificação. Porém, alterando as váriáveis *HADAPT:KEEPXFORMDISTINCT* e *HADAPT:SAVESPKRMODELS* para *TRUE*, via arquivo de configuração do **HERest**, é possível salvar o modelo já transformado. Esse processo foi necessário para se utilizar os modelos adaptados no decodificador Julius.

3.3.2 MAP

Uma outra maneira de ser realizar a adaptação de locutor é com o uso da MAP [64,65]. Essa técnica, também é conhecida como *Bayesian adaptation*, utiliza o arcabouço do aprendizado Bayesiano, onde as distribuições iniciais do modelo independente de locutor representam

[5] O *offset* é uma constante utilizada para efetuar um deslocamento fixo nos parâmetros a serem adaptados.

o conhecimento a priori e, são atualizadas através da regra de Bayes para se alcançar um novo modelo, dependente do locutor.

$$P(\lambda|O) = \frac{P(O|\lambda)P_0(\lambda)}{P(O)} \tag{3.3}$$

onde $P_0(\lambda)$ é a probabilidade a priori do modelo λ.

Para uma dada gaussiana, a sua média pode ser reestimada como sendo

$$\vec{\hat{\mu}} = \frac{\tau\vec{\mu_0} + \sum_{t=1}^{T}\gamma(t)\vec{o_t}}{\tau + \sum_{t=1}^{T}\gamma(t)} \tag{3.4}$$

em que τ é chamado *meta-parâmetro* que indica a "inclinação" entre a estimação de máxima verossimilhança da média e a média a priori μ_0, e $\gamma(t)$ é a probabilidade da gaussiana gerar o vetor de observações o_t. Na MAP, cada média de uma componente gaussiana é adaptada por individualmente, sendo assim, a técnica MAP necessita de mais dados para se obter uma boa adaptação quando comparada a técnica MLLR. Porém quando se tem uma grande quantidade de dados disponível, a técnica MAP, em geral, possui desempenho melhor que a MLLR.

Ambas as técnicas possuem suas vantagens e desvantagens dependendo da quantidade de dados disponível. A MLLR é recomendada quando se possui poucos dados para a adaptação, pois sua convergência é mais rápida, porém também "satura" mais rapidamente. A MAP deve ser utilizada quando se possui uma quantidade de dados maior, pois sua convergência é mais lenta, porém consegue uma melhor adaptação já que reestima as componentes individualmente. Trabalhos relacionados apontam vantagens do uso combinados das duas técnicas [66]. Na seção4.1 é apresentada uma comparação das técnicas com resultados individuais e a combinação de ambas.

3.4 Interface de Programação de Aplicativos

Ao promover o amplo desenvolvimento de aplicações baseadas em reconhecimento de voz, foi observado que não era suficiente apenas tornar os recursos disponíveis, tais como

modelos acústicos e de linguagem. Esses recursos são úteis para pesquisadores, mas o que a maioria dos programadores desejam é a praticidade oriunda de uma interface de programação de aplicativos (API). Por isso, foi necessário complementar o código que faz parte do pacote de distribuição do Julius.

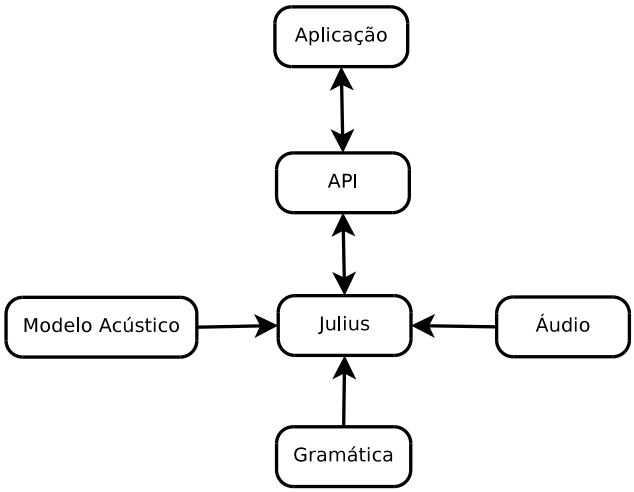

Figura 3.7: Modelo de interação com a API.

Assim, uma API foi desenvolvida na linguagem de programação C++ com a especificação *Common Language Runtime*[6], que permite comunicação entre as linguagens suportadas pela plataforma .NET. A API proposta permite o controle em tempo-real do *engine* para reconhecimento de voz, Julius, e da interface de áudio do sistema. Como pode ser visto na Figura 3.7, as aplicações interagem com o reconhecedor Julius através da API. Basicamente, a API "esconde" do programador detalhes de baixo nível relacionados à operação do *engine*.

A API foi originalmente projetada para o sistema operacional Windows. Visto que a API suporta objetos compatíveis com o modelo de automação COM[7] (*Component Object Model*) , é possível acessar e manipular (ex: ajustar propriedades, invocar métodos) objetos de automação compartilhados que são exportados por outras aplicações. Do ponto de vista da programação, a API consiste de uma classe principal denominada *SREngine*. Essa classe expõe à aplicação um conjunto de métodos e eventos descritos na Tabela 3.3.

O método construtor *SREngine* permite que a aplicação controle aspectos do reconhecedor Julius. Esse método possibilita que a aplicação carregue os modelos acústico e de

[6]http://msdn.microsoft.com/pt-br/library/ddk909ch.aspx, visto em março 2010

[7]http://www.microsoft.com/com/, visto em março 2010

Metodo/Evento	Descrição básica
SREngine	Método para carregar e inicializar o reconhecedor
startRecognition	Método para iniciar o reconhecimento
stopRecognition	Método para pausar/parar o reconhecimento
OnRecognition	Evento chamado quando alguma sentença é reconhecida
OnSpeechReady	Evento chamado quando o reconhecimento é ativado

Tabela 3.3: Principais métodos e eventos da API.

linguagem a serem utilizados, inicie e pare o reconhecimento e receba eventos e resultados provenientes do *engine* de reconhecimento.

Uma aplicação baseada em voz precisa criar, carregar e ativar uma gramática, que essencialmente indica o método de reconhecimento empregado, ou seja, ditado ou *context-free grammar*. A gramática para ditado é implementada via modelo de linguagem, que define um extenso conjunto de palavras. Por sua vez, essas palavras podem ser pronunciadas de uma forma relativamente irrestrita. Já a gramática livre de contexto age como um modelo de linguagem. Ela provê ao reconhecedor regras que definem o que pode ser dito. O formato de texto da gramática utilizada pelo Julius é especificado em [67].

O método *startRecognition*, responsável por iniciar o processo de reconhecimento, basicamente ativa as regras gramaticais e abre o *stream* de áudio. Similarmente, o método *stopRecognition* desativa uma determinada regra e fecha o *stream* de áudio.

Adicionalmente aos métodos, alguns eventos também foram implementados. O evento *OnSpeechReady* sinaliza que o *engine* está ativado para reconhecimento. Em outras palavras, ele surge todo vez que o método *startRecognition* é invocado. Já o evento *OnSRecognition* acontece sempre que o resultado do reconhecimento encontra-se disponível, juntamente com o seu nível de confiança.

A confiabilidade do que foi reconhecido pelo *engine* é essencial para aplicações reais, dado que sempre ocorrerá erros de reconhecimento e, portanto, os resultados podem ser aceitos ou rejeitados. A sequência de palavras reconhecidas e o seu nível de confiança[8] são passados da API para a aplicação através da classe *RecoResult*. Na seção seguinte é descrita uma simples

[8] A confiança assume valores entre 0 e 1, onde 1 representa 100% de confiança.

aplicação utilizando a API.

3.4.1 PPTController

A aplicação que será mostrada a seguir faz uso de recursos próprios construídos no Laboratório de Processamento de Sinais da Universidade Federal do Pará (LaPS). Com esse aplicativo é possível controlar uma apresentação de *slides* via comandos de voz, como por exemplo abrir uma apresentação, avançar para o último *slide*, entre outras funcionalidades.

Para ilustrar a sua funcionalidade e mostrar a possibilidade de comunicação com outros programas, será apresentado o PPTController que, escrito na linguagem de programação C#, faz uso da API e do *Microsoft Office PowerPoint 2007*. O PPTController permite ao usuário controlar o documento através de comandos de voz específicos. São eles:

- **Mostrar:** primeiro comando que deve ser enviado, pois abre o *slide show*.
- **Próximo ou Avançar:** ir para o próximo *slide* da apresentação.
- **Anterior ou Voltar:** voltar para o *slide* anterior na apresentação.
- **Primeiro:** ir imediatamente para o primeiro *slide* da apresentação.
- **Último:** ir imediatamente para o último *slide* da apresentação.
- **Fechar:** fechar a apresentação e voltar para a tela principal do aplicativo.

O pacote de distribuição da API possui três DLL's: *julius.dll*, *sent.dll* e *LapsAPI.dll*. As duas primeiras possuem códigos do reconhecedor Julius e devem ser copiadas para a pasta do sistema operacional: $\backslash WINDOWS\backslash system$. Já a última biblioteca é a API, propriamente dita, que deve ser incluída como referência no código fonte.

O reconhecedor é inicializado através do construtor *SREngine* que recebe como argumento o arquivo de configuração do Julius ("jConf"). Esse arquivo contém as especificações de todos os recursos utilizados pelo Julius durante o processo de reconhecimento. Dentro os principais, encontra-se o modelo acústico, que também foi construído com recursos próprios [68], e a gramática livre de contexto, que foi elaborada seguindo a documentação do Julius [67].

```
SREngine re;
re = new SREngine("C:/User/Pedro/ppt.jconf");
```

Com o reconhecedor instanciado deve-se delegar para o *SREngine* quais funções vão representar os eventos, ou seja, o que acontecerá quando o reconhecimento for ativado ou quando alguma sentença for reconhecida, eventos *OnSpeechReady* e *OnRecognition*, respectivamente.

```
SREngine.OnRecognition += sr_onRecognition;
re.OnSpeechReady += sr_speechReady;
```

Ao evento *OnSpeechReady* deve ser passada uma função sem argumentos, informando que o Julius está ativado para reconhecimento. Já o evento *OnRecognition* recebe uma função com o argumento *RecoResult*. A *RecoResult* é uma classe da API produzida sempre que uma sentença é reconhecida. Ela possui informações sobre o reconhecimento, como a própria sentença e o seu nível de confiança informado pelo reconhecedor. O uso das funções pode ser conferido no código abaixo.

```
void sr_onRecognition(RecoResult result){
  if (result.getConfidence() > 0.7)
    Actions(result.getUterrance());
}
void sr_speechReady(){
  recogetionRichTextBox1.AppendText("Reconhecendo\n");
}
```

A função *Actions*, presente no código acima, é responsável por controlar as funcionalidades do programa Microsoft PowerPoint. Para isso, ela faz uso de componentes integrados dentro de uma arquitetura OLE[9] (*Object Linking and Embedding*). Essa comunicação não será aqui descrita, porém seu código é de fácil compreensão e encontra-se disponível no pacote do PPTController.

Assumindo que o reconhecedor está instanciado, o método *startRecognition* pode ser invocado para iniciar o processo de reconhecimento, assim como o método *stopRecognition* para

[9]http://support.microsoft.com/kb/122263, visto em março de 2010

parar o reconhecimento. No PPTController, especificamente, esses métodos são chamados a partir de ações de dois botões mostrados abaixo.

```
void button1_Click(object sender, EventArgs e){
  re.startRecognition();
}
void button2_Click(object sender, EventArgs e){
  re.stopRecognition();
}
```

Fazendo uso do conjunto limitado de métodos e eventos apresentados acima, mostrou-se que é viável construir aplicações baseadas em voz com a API. O PPTContoller é um *software* livre e encontra-se disponível na página do grupo FalaBrasil [13], juntamente com as versões mais atualizadas da API e dos recursos construídos.

3.4.2 Versão para Linux

Foi desenvolvida também uma versão da API para ambiente Linux, o princípio de funcionamento é o mesmo da versão para Windows, a diferença básica é que as ações são passadas para a API através de ponteiro de função, usando o método *setOnRecognizeAction*.

```
void reconheceu(RecoResult *result){
   cout << result->getUterrance() << " Confiança "
      << result->getConfidence() << endl;
}
re->setOnRecognizeAction(&reconheceu);
```

Para usar a biblioteca compartilhada (*libLapsAPI.so*) deve-se copiá-la para */usr/lib* e passá-la como parâmetro para o compilador. O código abaixo mostra como compilar a função *main*.

```
gcc -ILapsAPI/include/ -Ijulius-4.1.3/libjulius/include/
    -Ijulius-4.1.3/libsent/include/ -o main main.cpp
    -LLapsAPI/Release/ -lLapsAPIJulius4.1.3
```

Caso não se tenha acesso a pasta de bibliotecas, pode-se sobrescrever a variável *LD_LIBRARY_PATH*.

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:"caminho/para/libLapsAPI.so"
```

3.4.3 Coruja

Ao conjunto *LapsAPI + Julius + Modelo Acústico + dicionário e outros recursos* deu-se o nome de **Coruja**. Então tem-se o Coruja como sendo um completo sistema de reconhecimento de voz. No site do grupo FalaBrasil é possível obter o *software*, sua documentação e acesso a lista de discussão.

O Coruja está sendo utilizado em vários projetos, dentro os quais podemos citar:

- FFTranscriber[10]: O projeto se propõe a pesquisar novas técnicas de reconhecimento e realce de voz, focalizando no desenvolvimento de uma ferramenta para uma transcrição forense eficaz baseada nessas tecnologias. Essa nova ferramenta e metodologia têm o propósito de agilizar principalmente os casos de transcrição de registros de áudio sigilosos realizadas pelo Departamento de Fonética Forense do Centro de Perícias Científicas (CPC) Renato Chaves, podendo posteriormente ser disseminada a departamentos similares.

- SimonBR: o projeto visa a construção da versão para PB do *software* Simon[11]. O Simon é um *software* para comando e controle de um computador sem utilizar mouse ou teclado. O mesmo possui código livre e foi desenvolvido para ser independente de língua.

- Interfaces Naturais de Interação com Robôs Autônomos[12] - Alunos de iniciação científica do curso de Engenharia Mecatrônica da Universidade de Brasília utilizam o Coruja em um projeto para comandar um robô Pioneer via comandos gestuais e comandos de voz.

- Coruja Navigator[13]: A meta do projeto é tornar a Web mais acessível para pessoas com necessidades especiais, mais especificamente para deficientes visuais. Com este

[10] www.laps.ufpa.br/joomla/images/publicacao/2008/FFTranscriber.pdf, visto em março de 2010
[11] www.cyber-byte.at/wiki , visto em abril de 2010
[12] http://groups.google.com.br/group/human-robot-interaction , visto em março 2010.
[13] http://code.google.com/p/coruja-navigator/ , visto em maio 2010

aplicativo é possível navegar em sites de notícias usando reconhecimento e síntese de voz. Um template deve ser desenvolvido e associado a cada site, possibilitando a análise do conteúdo da página. Nesta versão estamos disponibilizando o aplicativo com o template para o site do jornal Folha de São Paulo.

- SpeechOO[14]: consiste em uma extensão do BrOffice.org que oferece reconhecimento de voz (Coruja) na suíte de escritório livre. A versão atual do SpeechOO é capaz de transcrever um ditado proferido pelo usuário em um documento do *Writer*. Entre as funcionalidades previstas estão ainda comandos de voz para navegação pela interface do usuário. Ele é um software livre desenvolvido por brasileiros, pronto para funcionar nas línguas portuguesa e inglesa.

[14]http://code.google.com/p/speechoo/ , visto em maio de 2010

Capítulo 4

Resultados

Neste capítulo são descritos os principais resultados obtidos, onde diferentes configurações de modelo acústico, gramáticas e modelos de linguagem foram testadas, buscando assim entender como os mesmos se relacionam e o modo como influenciam na tarefa de decodificação. Será mostrado o impacto das técnicas de adaptação descritas nos capítulos anteriores. Na última seção temos os resultados obtidos com as várias simulações, onde realizou-se uma comparação entre três sistemas de reconhecimento de voz.

Os modelos de linguagem trigrama descritos na Seção 3.2.1 foram utilizados nos testes dos sistemas. As medidas de desempenho utilizadas foram a WER (*word error rate*) e o fator de tempo real xRT (*real-time factor*) médio. O xRT é obtido dividindo-se o tempo que o sistema gasta para reconhecer uma frase, pela duração da mesma, e serve como medida de velocidade do sistema. Para realização da tarefa de decodificação foram utilizados o HDecode e Julius.

4.1 Adaptação de Locutor

Os resultados apresentados nessa seção, buscam mostrar como técnicas de adaptação de locutor podem ser utilizadas para tratar o problema de descasamento acústicos entre *corpora* diferentes.

Inicialmente, simulações foram realizadas utilizando somente o decodificador *HDecode*. Três modelos acústicos utilizando 14 Gaussianas por mistura com trifones dependentes de

contexto foram criados a partir dos *corpora* Spoltech e LapsStory, dois utilizando os *corpora* individualmente e um combinando as bases. Os parâmetros utilizados na decodificação são mostrados na Tabela 4.1. Para as simulações foi utilizado um computador Intel(R) Pentium Dual Core 1,8 GHz com 2 GB de memória RAM.

Tabela 4.1: Parâmetros utilizados para decodificação.

word insertion penalty	22
LM scale factor	20
pruning beam width	230
acoustic scale factor	1.0 e 2.0

Todos os testes foram realizados com o LapsBenchmark. Com isso, busca-se simular e avaliar como o sistema se comportaria em aplicações com descasamento acústico. A Tabela 4.2 mostra os resultados obtidos com os modelos acústicos criados a partir do LapsStory, Spoltech e com a combinação de ambos.

Tabela 4.2: Resultados obtidos com os modelos acústicos do LapsStory e Spoltech.

Modelos Acústicos	**WER (%)**	**xRT**
Spoltech	34.8	6
LapsStory	52.5	10
LapsStory+Spoltech	40	9
MA's com peso 2.0	**WER (%)**	**xRT**
Spoltech	44.3	1.5
LapsStory	55.9	3
LapsStory+Spoltech	48.5	1.6

Nos modelos treinados individualmente com o Spoltech e LapsStory, é visto que ambos obtiveram taxas de erro altas, principalmente no caso do LapsStory, já que o mesmo foi produzido em um ambiente acústico totalmente diferente do LapsBenchmark. Já a combinação dos *corpora*, mostrou-se ineficaz mesmo quando se realizou a normalização (normalização da média e variância dos parâmetros MFCC's), fato justificado pela grande diferença entre os mesmos. Foi observado que para todos os modelos acústicos utilizados, o sistema apresentou

um alto valor de xRT, o que impossibilita a utilização dos mesmos em aplicações de tempo-real. De forma a aumentar a velocidade do sistema, modificou-se o peso do modelo acústico para 2.0 no processo de decodificação (parâmetro *-a* no HDecode), o qual acelera o processo de busca aumentando os *scores* observados nas HMM's. Essa alteração torna o sistema até 4 vezes mais rápido, porém acarreta em perda de precisão, como mostrado na segunda parte da Tabela 4.2.

Na segunda etapa de testes, foram adotadas as técnicas de adaptação de locutor MLLR e MAP, descritas na Seção 3.3, para melhorar o desempenho do sistema. Porém, diferente de uma adaptação de locutor convencional, realizou-se na verdade uma adaptação de ambiente (*environment adaptation*). Partindo do modelo acústico produzido com o LapsStory, fez-se uma adaptação utilizando todo o *corpus* Spoltech. Assim, o modelo acústico do LapsStory, treinado com uma maior quantidade de dados, com poucos locutores e alta razão sinal/ruído, foi modificado de maneira a melhor modelar ambientes ruidosos e a fala de diversos locutores. Os resultados da adaptação são exibidos na Tabela 4.3.

Tabela 4.3: Resultados obtidos com as técnicas de adaptação de locutor.

Técnica de adaptação	**WER (%)**	**RT**	Redução na WER em relação ao modelo acústico Spoltech (34.8)
MLLR	30.2	9.5	-4.6
MAP	29.3	7.0	-5.5
MLLR+MAP	25.5	8.3	-9.3
MA's com peso 2.0	**WER (%)**	**RT**	Spoltech (44.3)
MLLR	34.2	2.2	-10.3
MAP	35.6	2.0	-8.7
MLLR+MAP	29.6	2.4	-14.7

A combinação MLLR+MAP apresentou melhores resultados. Trabalhos relacionados apontam vantagens do uso combinados das duas técnicas [66]. De forma geral, a técnica MLLR trabalha com agrupamento de Gaussianas com características semelhantes via matrizes de transformação. Isso permite que a mesma seja efetiva mesmo para pequenos conjuntos

de dados, mas não obtém desempenho ótimo quando o conjunto de dados é grande. Em contrate, a MAP utiliza as Gaussianas dos trifones individualmente. Isso permite aumento do desempenho ao custo da necessidade de um maior conjunto de dados para treino. Como visto na segunda parte da Tabela 4.3 a queda na WER com o uso das 2 técnicas de adaptação em conjunto é de mais de 14% quando comparado com os resultados obtidos com o modelo acústico do Spoltech.

Todos os resultados apresentados nas próximas Seções utilizaram o modelo acústico independente de locutor treinado com o LapsStory e adaptado com as técnicas MLLR e MAP. Atualmente, o modelo está na sua versão 1.5 e será referenciado, a partir de agora, como LapsAM-1.5.

4.2 Testes utilizando Gramáticas

Como descrito na Seção 2.4.3.1, sistemas de reconhecimento de voz também podem utilizar gramáticas para definir exatamente o que pode ser reconhecido, reduzindo o espaço de busca e aumentando a velocidade do decodificador.

Nesse contexto, realizou-se testes de desempenho utilizando gramáticas. Para se realização dos testes utilizou-se do decodificador Julius, pois o mesmo é o decodificador utilizado no *software* Coruja e sua gramática é bastante simples de implementar. Inicialmente, foi construída uma gramática composta por 31 frases que correspondem a possíveis comandos dados a um computador. Abaixo segue a lista dos comandos:

```
abrir/fechar navegador da internet
abrir/fechar microsoft word
abrir/fechar powerpoint
abrir/fechar windows média player
abrir/fechar calendário
abrir/fechar skype
abrir pesquisa no google
abrir/fechar excel
abrir/fechar meus documentos
abrir/fechar calculadora
abrir/fechar bloco de notas
```

```
abrir/fechar terminal
reiniciar/delisgar computador
escrever email
exibir hora
exibir serviços de impressão
executar comando
acessar área de trabalho
discar números NÚMEROS
```

É válido observar que dentre os comandos listados, o comando “discar números” pode ser seguido por qualquer quantidade de números entre 0 e 9. Após a construção da gramática, foram gravadas 99 sentenças utilizando 3 locutores, onde cada um gravou 33 sentenças baseados na gramática, onde a sentença ”discar números“ é sempre dita duas vezes por cada locutor, utilizando diferentes sequências númericas.

Nos testes utilizando o LapsAM-1.5, obteve-se uma WER de 4.1% com um xRT de 0.2. Realizando-se uma nova adaptação do LapsAM-1.5 com 10 minutos de voz de cada locutor a WER caiu para 1.5% com um xRT de 0.13. Pelos resultados, observamos que a construção de aplicações utilizando gramáticas, tendem a apresentar excelentes resultados.

4.3 Ajuste do *Beam* nos Decodificadores

Como descrito na Seção 2.4.4, o *beam* é um parâmetro de grande importância no processo de decodificação, sendo assim, vários testes foram realizados com diferentes valores de *beam* para os decodificadores HDecode e Julius. Em todos os testes foram utilizado o LapsAM-1.5, os modelos de linguagem descrito na Seção 3.2.1 e para testes o LapsBenchmark.

Na Figura 4.1, são observados os resultados dos testes com o HDecode e Julius. É notável a influência do *beam* na WER. Observa-se que com o aumento do *beam* tem-se uma considerável redução na WER, isso já era era esperado, pois com um maior *beam*, menos hipóteses são descartados durante a busca, aumentando as chances de acerto por parte do decodificador. Porém, como visto na Figura 4.2, o aumento do espaço de busca através do aumento do *beam* torna os decodificadores mais lentos, principalmente no caso do HDecode que chega a um xRT de 98.1 quando utiliza-se um beam de 2000, não foram realizados testes

para valores maiores que este, pois o tempo para realização seria inviável. Pela Figura 4.1, observa-se que o HDecode tem sua WER estabilizada em 26,19% quando o *beam* chega a 500, porém o Julius consegue utilizar valores bem superiores, sem que isso proporcione um aumento exagerado do xRT, para um *beam* de 40 mil o Julius obteve uma WER de 30.35% com um xRT de 13.4. Para quase todos os *beams* testados o Julius sempre apresentou um baixo xRT, isso ocorre, pelo fato do Julius, diferente do HDecode, utilizar várias aproximações e diversos métodos de decodificação que o tornam bastante veloz como descrito em [54].

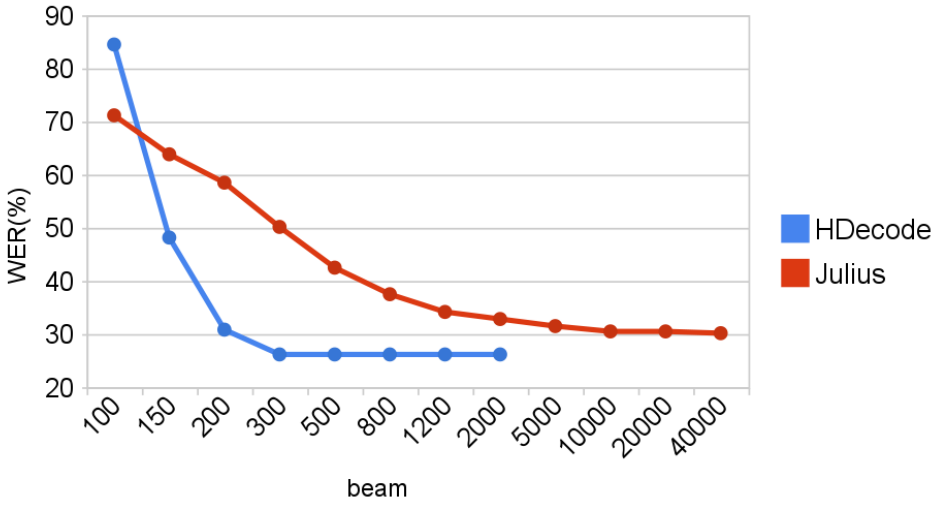

Figura 4.1: Variação da WER com o aumento do *beam*.

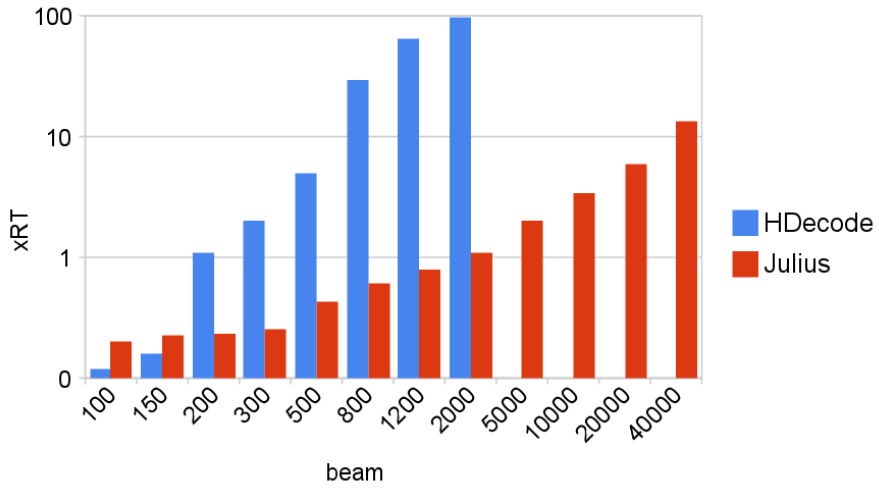

Figura 4.2: Variação do xRT com o aumento do *beam*.

4.4 Comparação entre Sistemas de Reconhecimento de Voz

De forma a medir a viabilidade da construção de sistemas LVCSR, a partir dos recursos descritos até o momento, uma comparação entre três sistemas foi realizada: HDecode, Julius e IBM ViaVoice. O processo de avaliação dos sistemas foi realizada em duas etapas: utilizando modelos independentes e dependentes de locutor.

Para os decodificadores HDecode e Julius, um processo de otimização de parâmetros foi realizado. Este processo visou obter o melhor desempenho possível em termos de WER e xRT. Onde se fixou o xRT em 1.2 como sendo o valor máximo aceitável. Os parâmetros utilizados em ambos os decodificadores são mostrados nas Tabela 4.4 e 4.5. A descrição dos parâmetros do HDecode e Julius podem ser encontrados em [42] e [69], respectivamente.

Tabela 4.4: Parâmetros utilizados nos testes com o HDecode.

Pruning beam width	250.0
Language Model scale factor	20.0
Word insertion penalty	22.0
Wordend beam width	100.0
Number of tokens per state	8
Acoustic scale factor	1.5

Tabela 4.5: Parâmetros utilizados nos testes com o Julius.

Pruning beam width for the first pass	2000.0
Pruning beam width for the second pass	200.0
Language model weight for the first and second passes	15.0
Word insertion penalties for the first and second passes	10.0
Score envelope width	300.0

Para os testes com modelos independentes de locutor utilizou-se o LapsBenchmark. Inicialmente, foi necessário realizar alguns procedimentos para os testes com o *software* IBM

ViaVoice, pois o *software* impõe uma etapa de adaptação de locutor antes de sua utilização. Sendo assim, a etapa de adaptação de locutor foi feita utilizando 6 locutores, 3 homens e 3 mulheres, cada um contribuindo com 1/6 do processo, que corresponde a um total 10 minutos de áudio. Para os outros decodificadores, esta estapa não foi necessária, pois o LapsAM-1.5 já é independente de locutor.

A Tabela 4.6 exibe os resultados dos testes. Infelizmente, não foi possível medir o xRT para o IBM ViaVoice, pois o procedimento adotado para realizar o testes dificultou a medição. Observamos que o HDecode e o IBM ViaVoice obtiveram quase a mesma WER, enquanto o julius apresentou uma maior WER, porém um menor xRT.

Decoder	**WER (%)**	**xRT**
Julius	32.87	0.9
HDecode	26.8	1.1
IBM ViaVoice	29.3	x

Tabela 4.6: Comparação dos sistemas utilizando modelos independentes de locutor.

A segunda fase de testes utilizou modelos dependentes de locutor. Os testes foram realizados com dois locutores, cada um contribuiu com 10 minutos de voz para adaptação dos modelos e 20 frases para teste. Ambas as técnicas MLLR e MAP foram utilizadas no processo, que produziu dois novos modelos adaptados para cada locutor específicos, esse modelos foram utilizados nos testes com o HDecode e Julius. Para o IBM ViaVoice, o processo padrão de adaptação feito pelo próprio *software* foi adotado.

Os novos resultados são descritos na Tabela 4.7. Como esperado, a adaptação de locutor melhorou o desempenhos para todos os decodificadores. O HDecode mostrou resultados satisfatórios quando comparado ao *software* comercial IBM ViaVoice. Entretanto, o Julius ainda permaneceu com desempenho inferior.

Os resultados apresentados, mostram que, com os recursos produzidos, é possível a construção sistemas de reconhecimento de voz para o PB. Apesar dos resultados do Julius serem inferiores aos do HDecode em testes com grandes vocabulários, ainda sim ele foi escolhido como o decodificador do Coruja, os motivos dessa escolha são descritos abaixo:

Decoder	**WER (%)**	**xRT**
Julius	16.56	1.0
HDecode	12.4	0.9
IBM ViaVoice	17.3	x

Tabela 4.7: Comparação dos sistemas utilizando modelos dependentes de locutor.

- Diferente do HDecode, a licença do Julius é bastante flexível, permitindo a sua distribuição (o Julius vem integrado ao *software* Coruja, por exemplo) e a construção de aplicações comerciais.

- Como descrito em [70], o Julius pode obter resultados superiores aos do HDecode. Acredita-se que o Julius necessite de modelos treinados com mais quantidade de dados, isso foi observado durante o treinamento dos modelos, pois conforme aumentava-se os dados de treino, o melhora do Julius era maior do que a do HDecode.

- O Julius possui um menor custo computacional e mais velocidade. Como descrito em [71], o mesmo já foi implementado em microprocessadores, iPhones, etc.

- O Julius vem sendo melhorado constantemente. Em média a cada 2 meses, um nova versão é disponibilizada, já o HDecode está na mesma versão desde de 2006.

- Para o Japonês, o Julius já alcança uma WER menor que 7.9% [54].

- O Julius foi construído especificamente para aplicações em tempo real, onde ele próprio realiza a extração de parâmetros (semelhante ao *HCopy*), suporta entrada através do microfone, via rede de computadores, etc. O HDecode não possui nenhum suporte a tempo real e a entrada é realizada apenas através de arquivos MFCCs extraídos pelo *HCopy*.

Capítulo 5

Considerações Finais

Neste trabalho foram apresentados os recursos necessários para implementação de um software para reconhecimento de voz para Português Brasileiro. Foram discutidos também vários aspectos sobre reconhecimento automático de fala contínua para grande vocabulários. Vários conceitos importantes foram abordados e soluções para problemas comuns da área foram apresentados. Foram construídos e testados modelos acústicos, modelos de linguagem, *corpora* de voz e texto, além da construção de uma API.

A implementação do sistema apresentado neste trabalho foi baseada em cadeias ocultas de Markov. As principais ferramentas disponíveis na Internet foram utilizadas e todos os blocos que compõem um sistema de reconhecimento de voz foram descritos. Foi observado, que técnicas de adaptação de locutor MAP e MLLR, podem ser utilizadas, tanto para a adaptação dos modelos para locutores específicos, quando para resolução de problemas causados pelo descasamento acústico. O LapsStory mostrou-se eficiente para a construção de modelos acústicos quando combinado com tais técnicas.

Foi apresentada uma API (LapsAPI) desenvolvida na linguagem de programação C++ que permite comunicação entre as linguagens suportadas pela plataforma .NET. A API permite o controle em tempo-real do *engine* para reconhecimento de voz, Julius, e da interface de áudio do sistema. Os principais recursos desenvolvidos foram compilados em um único *software* chamado Coruja. o Coruja é composto pelo modelo acústico, LapsAPI, Julius e o dicionário fonético. Testes do Coruja utilizando gramáticas, mostraram a viabilidade do mesmo para tarefas de comando e controle, onde obteve-se WER de 4.1% com xRT de 0.2. Nos testes com

modelos de linguagem n-grama para grandes vocabulários, o Coruja apresentou resultados inferiores quando comparado ao HDecode e IBM ViaVoice. Apesar disso, como mostrado na Seção 4.4, o Julius possui várias vantagens e espera-se que com mais dados para treino, o Coruja pode superar ambos HDecode de IBM ViaVoice.

Como visto na Seção 3.4.3, o *software* Coruja vem sendo bastante utilizado por pesquisadores e programadores, a lista de discussão já conta com quase 30 membros.

A construção de um sistema LVCSR para o PB é um grande desafio. Por isso, o número de pesquisadores na área de processamento de voz cresce a cada ano, novos *corpora* para o PB vêm sendo desenvolvidos, além de várias ferramentas. Todos os recursos desenvolvidos neste trabalho estão disponibilizados na página do grupo *FalaBrasil* [13]. Espera-se que outros grupos de pesquisa e desenvolvedores de *software* possam se utilizar de tais recursos para desenvolvimento de novos sistemas e aprimoramento dos resultados.

5.1 Publicações Geradas

- Patrick Silva, Pedro Batista, Nelson Neto, Aldebaro Klautau. *An Open-Source Speech Recognizer for Brazilian Portuguese with a Windows Programming Interface*. In: The International Conference on Computational Processing of Portuguese - PROPOR 2010, Porto Alegre.

- Pedro Batista, Patrick Silva, Nelson Neto, Aldebaro Klautau. *A non-Visual Web-Browsing System using Speech Recognition for Brazilian Portuguese*. In: The International Conference on Computational Processing of Portuguese - Demos Session PROPOR 2010, Porto Alegre.

- Patrick Silva, Nelson Neto, Aldebaro Klautau. *Novos Recursos e Utilização de Adaptação de Locutor no Desenvolvimento de um Sistema de Reconhecimento de Voz para o Português Brasileiro*. In XXVII Simpósio Brasileiro de Telecomunicações - SBRT, 2009.

5.2 Trabalhos Futuros

- Construção de um módulo de adaptação de locutor. O módulo irá implementar as técnicas MLLR e MAP e poderá ser utilizado em conjunto com o Coruja.

- Aprimoramento dos modelos de linguagem. Continuação do processo de extração de textos da internet para expansão do LapsNews e melhora do processo de treino dos modelos de linguagem n-grama.

- Expansão do LapsStory. Atualmente composto por apensas 15 horas, busca-se a melhora do desempenho do Coruja com o aumento do *corpus* de treino do modelo acústico. Novos áudios estão sendo obtidos, o objetivo é construção de um *corpus* para PB com mais de 100 horas.

- Tornar a API do Coruja SAPI *compliant*. Isso permitirá o uso do Coruja em aplicações Windows como as ferramentas do pacote *Office*, entre outros.

- Construção de um conversor de gramáticas. O conversor terá como entrada uma gramática no formato XML (*extensible markup language*), padrão para os *engines* de reconhecimento de voz do windows, mais simples de implementar, e terá como saída a mesma gramática no formato do Julius.

- Construção de *software*s utilizando o Coruja.

Referências Bibliográficas

[1] L. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–86, Feb. 1989.

[2] F. Beaufays, H. Bourlard, H. Franco, and N. Morgan, "Neural networks in automatic speech recognition," *in The Handbook of Brain Theory and Neural Networks*, 2000.

[3] R. Fagundes and I. Sanches, "Uma nova abordagem fonético-fonológica em sistemas de reconhecimento de fala espontânea," *Revista da Sociedade Brasileira de Telecomunicações*, vol. 95, 2003.

[4] L. Pessoa, F. Violaro, and P. Barbosa, "Modelo de língua baseado em gramática gerativa aplicado ao reconhecimento de fala contínua," in *XVII Simpósio Brasileiro de Telecomunicações*, 1999, pp. 455–458.

[5] S. Santos and A. Alcaim, "Um sistema de reconhecimento de voz contínua dependente da tarefa em língua portuguesa," *Revista da Sociedade Brasileira de Telecomunicações*, vol. 17, no. 2, pp. 135–147, 2002.

[6] I. Seara et al, "Geração automática de variantes de léxicos do português brasileiro para sistemas de reconhecimento de fala," in *XX Simpósio Brasileiro de Telecomunicações*, 2003, pp. v.1. p.1–6.

[7] M. Schramm, L. Freitas, A. Zanuz, and D. Barone, "A Brazilian Portuguese language corpus development," *International Conference on Spoken Language Processing*, vol. 2, pp. 579–582, 2000.

[8] C. A. Ynoguti and F. Violaro, "Influência da transcrição fonética no desempenho de sistemas de reconhecimento de fala contínua," in *XVII Simpósio Brasileiro de Telecomunicações*, 1999, pp. 449–454.

[9] R. Teruszkin and F. Vianna, "Implementation of a large vocabulary continuous speech recognition system for Brazilian Portuguese," *Journal of Communication and Information Systems*, vol. 21, pp. 204–218, 2006.

[10] E. Silva, M. Pantoja, J. Celidônio, and A. Klautau, "Modelos de linguagem n-grama para reconhecimento de voz com grande vocabulário," in *III Workshop em Tecnologia da Informação e da Linguagem Humana*, 2004.

[11] P. Silva, N. Neto, A. Klautau, A. Adami, and I. Trancoso, "Speech recognition for brazilian portuguese using the spoltech and OGI-22 corpora," *XXVI Simpósio Brasileiro de Telecomunicações - SBrt 2008*, 2008.

[12] N. Neto, P. Silva, A. Klautau, and A. Adami, "Spoltech and ogi-22 baseline systems for speech recognition in brazilian portuguese," *International Conference on Computational Processing of Portuguese Language - PROPOR*, 2008.

[13] "http://www.laps.ufpa.br/falabrasil," Visited in June, 2010.

[14] A. M. da Cunha and L. Velho, "Métodos probabilísticos para reconhecimento de voz," Laboratório VISGRAF - Instituto de Matemática Pura e Aplicada, Tech. Rep., 2003.

[15] L. Rabiner and B. Juang, *Fundamentals of speech recognition.* Englewood Cliffs, N.J.: PTR Prentice Hall, 1993.

[16] Jelinke and Frederick, "Métodos estatísticos para reconhecimento de voz," *The MIT Press*, 1998.

[17] P. Woodland and D. Povey, "Large scale discriminative training of hidden Markov models for speech recognition," *Computer Speech and Language*, vol. 16, pp. 25–47, 2002.

[18] A. Lee, T. Kawahara, and K. Shikano, "Gaussian mixture selection using context-independent HMM," *In Proceedings IEEE-ICASSP*, 2001.

[19] M. Cohen, H. Franco, N. Morgan, D. Rumelhart, and V. Abrash, "Hybrid neural network/hidden markov model continuous speech recognition," *Proceedings of the International Conference on Spoken Language Processing*, 1992.

[20] "Advancing human language technology in Brazil and the United States through collaborative research on Portuguese spoken language systems," Federal University of Rio Grande do Sul, University of Caxias do Sul, Colorado University, and Oregon Graduate Institute, 2001.

[21] "http://www.ldc.upenn.edu/catalog/catalogentry.jsp?catalogid=ldc2008s04," 2008.

[22] C. A. Ynoguti, P. A. Barbosa, and F. Violaro, "A large speech database for Brazilian Portuguese spoken language research," *Lecture Notes in Computer Science (LNCS)*, vol. 2721/9, pp. 193–196, 2003.

[23] C. A. Ynoguti and F. Violaro, "A Brazilian Portuguese speech database," *XXVI Simpósio Brasileiro de Telecomunicações*, 2008.

[24] L. Pessoa, F. Violaro, and P. Barbosa, "Modelos da língua baseados em classes de palavras para sistema de reconhecimento de fala contínua," *Revista da Sociedade Brasileira de Telecomunicações*, vol. 14, no. 2, pp. 75–84, Dez 1999.

[25] E. Silva, L. Baptista, H. Fernandes, and A. Klautau, "Desenvolvimento de um sistema de reconhecimento automático de voz contínua com grande vocabulário para o Português Brasileiro," *XXV Congresso da Sociedade Brasileira de Computação*, 2005.

[26] Kurzweil and Raymond, "Quando o HAL vai entender o que estamos falando?, em O legado de HAL: o computador de 2001 - Uma odisséia no espaço como sonho e realidade," *The MIT Press*, 1998.

[27] "http://www.cs.waikato.ac.nz/ml/weka/," Visited in March, 2008.

[28] C. Hosn, L. A. N. Baptista, T. Imbiriba, and A. Klautau, "New resources for Brazilian Portuguese: Results for grapheme-to-phoneme and phone classification," *In VI International Telecommunications Symposium, Fortaleza*, 2006.

[29] A. Siravenha, N. Neto, V. Macedo, and A. Klautau, "Uso de regras fonológicas com determinação de vogal tônica para conversão grafema-fone em português brasileiro," *7th International Information and Telecommunication Technologies Symposium*, 2008.

[30] D. Silva, A. de Lima, R. Maia, D. Braga, J. F. de Moraes, J. A. de Moraes, and F. R. Jr, "A rule-based grapheme-phone converter and stress determination for Brazilian Portuguese natural language processing," *VI International Telecommunications Symposium*, 2006.

[31] L. Rabiner and R. Schafer, *Digital Processing of Speech Signals*. Prentice-Hall, 1978.

[32] S. Davis and P. Merlmestein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *IEEE Trans. on ASSP*, pp. 357–366, Aug. 1980.

[33] J. Picone, "Signal modeling techniques in speech recognition," *Proceedings of the IEEE*, vol. 81, no. 9, pp. 1215–47, Sep. 1993.

[34] Ênio dos Santos Silva, "Desenvolvimento de um Reconhecedor Automático de Voz com Suporte a Grandes Vocabularios para o Português Brasileiro," Tech. Rep., 2005.

[35] C. Hosn, "Conversão grafema-fone para um sistema de reconhecimento de voz com suporte a grandes vocabulários para o português brasileiro," Master's thesis, Universidade Federal do Pará, Centro Tecnológico, 2006.

[36] L. R. Welch, "Hidden Markov models and the Baum-Welch algorithm," *IEEE Information Theory Society Newsletter.*, vol. 53, pp. 10–12, 2003.

[37] G. D. Forney, "The viterbi algorithm," *Proceedings of the IEEE*, vol. 3, pp. 268–278, 1973.

[38] P. Woodland and S. Young, "The HTK tied-state continuous speech recognizer," *In: Proc. Eurospeech'93, Berlim*, 1993.

[39] S. Young and P. Woodland, "State clustering in hmm-based continuous speech recognition," *Computer Speech and Language*, vol. 8, pp. 369–384, 1994.

[40] M. Hwang and X. Huang, "Shared distribution hidden markov models for speech recognition," *IEEE Trans Speech and Audio Processing*, vol. 1, pp. 414–420, 1993.

[41] L. Bahl, P. Souza, P. Gopalakrishnan, D. Nahamoo, and M. Picheny, "Context dependent modeling of phones in continuous speech using decision trees," in *DARPA Speech and Natural Language Processing Workshop*, 1994, pp. 264–270.

[42] S. e. Young, *The HTK Book*. Microsoft Corporation, Version 3.0, 2000.

[43] O. JJ, "The use of decision trees with context sensitive phoneme modelling," Master's thesis, Cambridge University Engineering Department, 1992.

[44] P. Woodland, J. Odell, V. Valtchev, and S. Young, "Large vocabulary continuous speech recognition using htk," *IEEE Int'l Conf. Acoustics, Speech, and Signal Processing, vol. 2, pp. 125–128, Adelaide*, 1994.

[45] R. Lippmann, "Speech recognition by machines and humans," *Speech Communication*, vol. 22, pp. 1–15, 1997.

[46] J. PICONE, "Ece 8463: Fundamentals of speech recognition." [Online]. Available: http://www.ece.msstate.edu/research/isip/publications/courses/ece_8463/

[47] F. Jelinek, "Continuous speech recognition by statistical methods," in *Proceedings. of. the IEEE, vol. 64 no. 4*, 1976, pp. 532–555.

[48] T. Vintsyuk, "Speech discrimination by dynamic programming," *Kibernetika (Cybernetics)*, vol. 4, pp. 81–88, 1968.

[49] S. Young, N. Russell, and J. Thornton, "Token passing: a conceptual model for connected speech recognition systems," 1989. [Online]. Available: svr-ftp.eng.cam.ac.uk

[50] A. Stolcke, "SRILM - an extensible language modeling toolkit," *International Conference on Spoken Language Processing*, 2002.

[51] "http://julius.sourceforge.jp/en/," Visited in May, 2009.

[52] B. T. Lowerre, "The harpy speech recognition system." Ph.D. dissertation, Pittsburgh, PA, USA, 1976.

[53] D. B. Paul, "The lincoln large-vocabulary stack-decoder based HMM CSR," in *HLT '94: Proceedings of the workshop on Human Language Technology*. Morristown, NJ, USA: Association for Computational Linguistics, 1994, pp. 399–404.

[54] A. Lee, T. Kawahara, and K. Shikano, "Julius - an open source real-time large vocabulary recognition engine," *Proc. European Conference on Speech Communication and Technology*, pp. 1691–1694, 2001.

[55] J. J. Godfrey, E. C. Holliman, and J. McDaniel, "SWITCHBOARD: Telephone speech corpus for research and development," *Texas Instruments Inc. Dallas*.

[56] R. J. Cirigliano, C. Monteiro, F. L. de F. Barbosa, F. G. V. R. Jr, L. Couto, and J. Moraes, "Um conjunto de 1000 frases foneticamente balanceadas para o Português Brasileiro obtido utilizando a abordagem de algoritmos genéticos," *XXII Simpósio Brasileiro de Telecomunicações*, 2005.

[57] "acdc.linguateca.pt/cetenfolha/," Visited in June, 2010.

[58] R. Kneser and H. Ney, "Improved backing-off for m-gram language modeling," *IEEE International Conference on Acoustics, Speech and Signal Processing*, 1995.

[59] L. de Abreu Borges, "Sistemas de adaptação ao locutor utilizando autovozes," Master's thesis, Escola Politécnica da Universidade de São Paulo, 2001.

[60] C. J. Leggetter and P. C. Woodland, "Speaker adaptation of hmms using linear regression," Cambridge University, Tech. Rep., 1994.

[61] L. C. Sousa, "Adaptação de locutor em sistemas de reconhecimento de fala contínua empregando eigenvoices," Master's thesis, Universidade Estadual de Campinas, 2004.

[62] P. C. Woodland, "Speaker adaptation: Techniques and challenges," in *Proc. IEEE Workshop on Automatic Speech Recognition and Understanding*, 2000, pp. 85–90.

[63] C. J. Leggetter and P. C. Woodland, "Flexible speaker adaptation using maximum likelihood linear regression," in *Proc. ARPA Spoken Language Technology Workshop*, 1995, pp. 104–109.

[64] J. L. Gauvain and C.-H. Lee, "Maximum a posteriori estimation for multivariate gaussian mixture observations os markov chains," *IEEE Transactions on Speech and Audio Processing*, vol. 02, pp. 291–298, 1994.

[65] C. H. Lee and J. L. Gauvain, "Speaker adaptation based on MAP estimation of HMM parameters," *IEEE ICASSP*, pp. 558–561, 1993.

[66] S. G. Ralf and R. Kompe, "A combined MAP + MLLR approach for speaker adaptation," *Proc Sony Res Forum*, vol. 9, pp. 9–14, 2000.

[67] "http://julius.sourceforge.jp/en_index.php?q=en_grammar.html," Visited in September, 2009.

[68] P. Silva, N. Neto, and A. Klautau, "Novos recursos e utilização de adaptação de locutor no desenvolvimento de um sistema de reconhecimento de voz para o Português Brasileiro," *In XXVII Simpósio Brasileiro de Telecomunicações*, 2009.

[69] A. Lee, *The Julius Book*. 0.0.2 ed. - rev 4.1.2, 2009.

[70] T. Rotovnik, M. S. Maucec, B. Horvat, and Z. Kacic, "A comparison of HTK, ISIP and Julius in Slovenian large vocabulary continuous speech recognition," *7th International Conference on Spoken Language Processing*, 2002.

[71] A. Lee and T. Kawahara, "Recent development of open-source speech recognition engine julius," *Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, 2009.