UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

Estudo e Implementação da Programação Genética para Síntese de Fala

por

EVANDRO FRANZEN

Dissertação submetida à avaliação, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação

Prof. Dr. Dante Augusto Couto Barone
Orientador

Porto Alegre, novembro de 2002.

CIP – CATALOGAÇÃO DA PUBLICAÇÃO

Franzen, Evandro

Estudo e Implementação da Programação Genética para Síntese de Fala / por Evandro Franzen. – Porto Alegre: PPGC da UFRGS, 2002.

113f.:il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2002. Orientador: Barone, Dante Augusto Couto.

1.Programação genética. 2. Síntese de fala. 3. Computação Evolucionária. I. Barone, Dante Augusto Couto. II. Título.

Agradecimentos

A meus pais pela dedicação, apoio e acompanhamento durante todos os anos de minha vida e por terem me proporcionado exemplos de comportamento e caráter.

A minha esposa pelos momentos de convívio, carinho e amor e pelo apoio durante as atividades profissionais e acadêmicas.

Ao Dr. Dante Barone pela orientação e por todo o conhecimento que tive oportunidade receber e compartilhar durante o convívio no período de desenvolvimento do mestrado.

A todos os colegas de aula, de trabalhos, professores e demais pessoas com as quais tive oportunidade de conviver durante o mestrado.

Às demais pessoas que me proporcionaram momentos de apoio, lazer e reflexão tão importantes para o bom andamento das atividades e para superação das dificuldades do dia a dia.

A Deus, por tudo.

Sumário

Lista de Abreviaturas

ADF Automatic Defined Functions

AG Algoritmos Genéticos

CE Computação Evolutiva

DAF Definição Automática de Funções

DSP Digital Signal Processing

HMM Hidden Markov Models

IA Inteligência Artificial

IPA International Phonetic Alphabet

LTS Letter To Sound

NLP Natural Language Processing

PE Programação Evolucionária

PG Programação Genética

PLN Processamento de Linguagem Natural

RNA Rede Neural Artificial

RPB Result Program Branch

TTS Text To Speech

VA Vida Artificial

Lista de Figuras

Lista de Tabelas

Resumo

Este trabalho descreve a aplicação da Programação Genética, uma técnica de Computação Evolucionária, ao problema da Síntese de Fala automática.

A Programação Genética utiliza as técnicas da evolução humana para descobrir programas bem adaptados a um problema específico. Estes programas, compostos de instruções, variáveis, constantes e outros elementos que compõe uma linguagem de programação, são evoluídos ao longo de um conjunto de gerações.

A Síntese de Fala, consiste na geração automática das formas de ondas sonoras a partir de um texto escrito. Uma das atividades mais importantes, é realizada através da conversão de palavras e letras para os sons da fala elementares (fonemas). Muitos sistemas de síntese são implementados através de regras fixas, escritas por programadores humanos.

Um dos mais conhecidos sistemas de síntese é o FESTIVAL, desenvolvido pela Universidade de Edimburgh, usando a linguagem de programação funcional LISP e um número fixo de regras.

Neste trabalho, nós exploramos a possibilidade da aplicação do paradigma da Programação Genética, para evoluir automaticamente regras que serão adotadas para implementação do idioma Português na ferramenta FESTIVAL, desenvolvido no projeto SPOLTECH (CNPq – NSF cooperação entre UFRGS e Universidade do Colorado).

A modelagem do problema, consiste na definição das regras de pronúncia do Português Brasileiro, que a implementação do sistema FESTIVAL pronuncia erradamente, já que o mesmo foi implementado primariamente para o idioma Inglês. A partir destas regras, o sistema de Programação Genética, desenvolvido neste trabalho, evolui programas que constituem boas soluções para a conversão de letras para fonemas.

A descrição dos resultados obtidos, cobre detalhes sobre a evolução das soluções, complexidade e regras implementadas, representadas pelas soluções mais bem adaptadas; mostrando que a Programação Genética, apesar de ser complexa, é bastante promissora.

Palavras-chave: Programação Genética, Síntese de Fala, Computação Evolucionária.

TITLE: “STUDY AND IMPLEMENTATION OF GENETIC PROGRAMMING TO SPEECH SYNTHESIS”

Abstract

This work describes the application of Genetic Programming, an Evolutionary Computation technique, to Automatic Speech Synthesis.

Genetic Programming uses some human evolution techniques in order to find fitted programs to a specific problem under solution. These programs, made of by instructions, variables, constants and other programming language elements, evolve through a set of generations.

Speech synthesis consists of the automatic generation of sound waveforms, from a written text. One of the major tasks involved is made through the conversion of words and letters to units corresponding to elementary speech sounds (phonemes) through fixed rules written by programmers.

One of the most know synthesis system is FESTIVAL developed at the University of Edimburgh, using LISP functional language, and a fixed number of rules.

In this work, we have explored the possibility of application of the Genetic Programming paradigm to evolve automatically rules that will be added to Festival tool implementation for Portuguese in the context of the SPOLTECH project (CNPq – NSF cooperation between UFRGS and University of Colorado).

The problem modelling consists of defining Brazilian Portuguese pronunciation rules that FESTIVAL implementation word “pronounce” wrongly, since as it was primarily implemented to English. From these rules, the Genetic Programming system developped in this work evolve programs which are good solutions to Brazilian Portuguese letter phoneme conversion.

The description of the obtained results covers details about evolution of solutions, complexity and implemented rules represented by the best fitted solutions; showing that the application of the Genetic Programming, besides being complex, is quite promising.

Keywords: Genetic Programming, Speech Synthesis, Evolutionary Computing.

1 Introdução

Apesar da evolução constante de hardware e software, grande parte das aplicações computacionais ainda é baseada em algoritmos e rotinas adaptadas para um problema específico. Além disso a imensa maioria destas ainda exige que o processo de interação entre humanos e sistemas ocorra através da linguagem escrita e em alguns casos através de instruções pré-definidas.

Assim, um dos grandes objetivos em anos recentes, tem sido a criação de sistemas computacionais que apresentem aspectos da inteligência humana, que possibilitem uma interação com alto grau de facilidade ou que tenham por exemplo a capacidade de resolução de problemas sem que sejam explicitamente programados para determinada atividade.

Para que tais objetivos sejam atingidos, algumas questões fundamentais devem ser avaliadas: Como poderão computadores resolver problemas, sem que sejam programados? Como os computadores podem encontrar o procedimento correto para resolução de um problema, sem que sejam dadas a eles todas instruções necessárias para isto? E ainda, como poderão computadores aprender a resolver por si próprios problemas cada vez mais complexos?

O aprendizado humano pode ser caracterizado como um processo adaptativo, no qual as atividades evoluem para permitir uma melhor compreensão e resolução de problemas. A Inteligência Artificial (IA) emprega diferentes técnicas e paradigmas na tentativa de simular o processo de aprendizado humano em computadores. A computação evolucionária é um ramo da IA no qual são definidas técnicas baseadas nos princípios da evolução humana. De acordo com Turing [KOZ 2000] a inteligência de máquina pode ser obtida através de uma abordagem biológica e pode haver uma relação direta com técnicas de busca.

Dentro da Computação Evolutiva diferentes técnicas tem sido propostas, entre elas podemos citar Vida Artificial, Programação Evolucionária e Algoritmos Genéticos, além da Programação Genética que é objeto de estudo neste trabalho. Estas técnicas apresentam diversas características comuns como a utilização e avaliação de um conjunto de soluções e operações de reprodução, cruzamento, seleção ou mutação; entretanto diferem na implementação destas características e essencialmente na forma de representação das soluções [FOG 2000].

A Programação Genética é uma técnica que tem como objetivo fazer evoluir um conjunto de programas de computador que são soluções candidatas a resolver um determinado problema. Através da evolução destas soluções é possível encontrar um conjunto de instruções que resolve o problema proposto [KOZ 92].

Processos de síntese e reconhecimento de voz, tradicionalmente são complexos e apresentam uma grande variabilidade, o que justifica a utilização e avaliação de diversas técnicas de IA para sua resolução. Como exemplo podemos citar as Redes Neurais Artificiais (RNAs) que são utilizadas com alto grau de sucesso em diversos sistemas de síntese e reconhecimento de voz [LAW 2000].

A síntese de voz normalmente é composta de um conjunto de etapas que envolvem desde a identificação do texto a ser sintetizado até a geração das ondas da fala. Uma etapa intermediária é a conversão do texto para unidades lingüísticas

equivalentes aos sons da linguagem; o problema é que esta relação não é direta e varia de um idioma para outro, exigindo assim que sejam implementadas regras para que o sistema de síntese possa efetuar essa etapa do processo.

Diversas aplicações tem utilizado a programação genética como forma de obter soluções para atividades complexas. Em [KOZ 92] são descritas aplicações da técnica à problemas de controle, planejamento de atividades, regressão simbólica ou descoberta de funções.

O objetivo deste trabalho é descrever a aplicação da programação genética a problemas de conversão de texto para fonema em processos de síntese de voz. Esta aplicação tem como meta principal descobrir programas que implementem as regras para conversão citadas acima, a partir de exemplos. O aprendizado e descoberta de regras através de técnicas evolutivas, em especial a Programação Genética, é um tópico pouco explorado em trabalhos científicos desenvolvidos no PPGC(UFRGS).

Além da originalidade da aplicação da PG à processos desta natureza, o aprendizado de regras, poderá contribuir para a construção de sistemas adaptáveis a diferentes idiomas, ou variações regionais de um mesmo idioma. Poucos sistemas hoje em dia permitem a utilização em diferentes contextos, sem que para tal uso seja necessário um trabalho extenso de implementação manual.

O trabalho desenvolvido está relacionado com o projeto Spoltech (CNPq-NSF) e tem como meta identificar potenciais utilizações da Programação Genética ao sistema para síntese implementado neste projeto. Uma descrição mais detalhada das atividades e resultados do Spoltech pode ser visualizada no capítulo 2.

O sistema será criado como uma ferramenta para aplicação da técnica a diversos problemas, não se restringindo apenas ao descrito neste trabalho, permitindo desta maneira a reutilização futura. A arquitetura do sistema para que tal objetivo seja atingido apresenta uma separação entre o núcleo e módulos de entrada, saída e validação. Para aplicações a outros problemas é necessário implementar funções relativas ao novo problema, as quais se encontram no módulo de entrada. Entretanto o foco deste trabalho é a execução da técnica visando a descoberta de programas que implementem regras para conversão de texto para fonema em língua portuguesa.

A dissertação está dividida em cinco capítulos. O capítulo 2 descreve os principais conceitos relacionados ao processo de síntese de voz, em especial a síntese texto para fala. São descritas as etapas do processo com ênfase na conversão de texto para fonema e nas particularidades deste processo na língua portuguesa. Serão descritas também regras implementadas de forma manual para avaliação dos resultados obtidos na execução da aplicação desenvolvida no trabalho.

O capítulo 3 apresenta uma descrição detalhada da programação genética e todos os mecanismos envolvidos em sua aplicação. É estabelecida ainda uma relação da técnica com outras semelhantes, além da descrição de resultados obtidos na sua aplicação a outros problemas.

No capítulo 4 são apresentadas as características do sistema implementado, incluindo a arquitetura básica e seus principais componentes. A partir de uma abordagem modular é possível aplicar a técnica a diferentes problemas sem que para tanto sejam necessárias modificações no núcleo do sistema. Diferentes aplicações podem exigir a criação e uso de novas funções de domínio, uso de diferentes conjuntos

de terminais, casos de fitness, variações que em sua maior parte podem ser realizadas sem um grande esforço de programação.

O capítulo 5 descreve o trabalho de modelagem da conversão de texto para fonema para sua resolução através da programação genética. Diversas questões são avaliadas e definidas, dentre estas estão a estrutura das soluções, forma de avaliação dos indivíduos, estrutura dos casos de fitness e dos conjuntos de funções e terminais. Embora a programação genética permita um alto grau de liberdade nas soluções produzidas, deve haver um mínimo de restrição à sua estrutura, caso contrário a busca seria inviabilizada devido a falta de convergência ou consumo exagerado de recursos computacionais.

Os resultados obtidos nas diversas execuções são descritos no capítulo 6. Para melhor compreensão, as regras selecionadas são agrupadas em quatro casos e para cada um deles são demonstradas e avaliadas as melhores soluções obtidas, gráficos comparativos de fitness médio da população, do melhor e pior indivíduo e da complexidade estrutural média da população e do melhor indivíduo. Através destas informações será possível efetivamente verificar a aplicabilidade da técnica ao problema.

No capítulo 7 serão emitidas considerações e conclusões finais sobre contribuições do trabalho desenvolvido e sugestões de alternativas futuras para o prosseguimento dos estudos.

O anexo 1 descreve os símbolos usados para representação dos fonemas no sistema Festival e no IPA (International Phonetic Alphabet). O anexo 2 tem por objetivo descrever as regras implementadas no sistema Festival, nas atividades do Projeto Spoltech. Estas regras permitem uma comparação com os resultados implementados pelas soluções citadas no capítulo 7. Os anexos 3 e 4 descrevem os códigos relativos à implementação de funções de domínio e de avaliação de fitness.

2 Lingüística computacional: Síntese de fala

2.1 Lingüística computacional

A transmissão de informação através de uma linguagem natural como Inglês, Português ou Alemão é um procedimento bem estruturado. Emissores e receptores devem compreender e conhecer os elementos que compõe o processo, do contrário o entendimento falha ou ocorre de forma incompleta [HAU 99].

A lingüística computacional à subárea da computação que envolve o estudo de fenômenos de percepção, cognição e investiga a produção e compreensão de linguagem falada em sistemas computacionais, envolvendo hardware e software. Nos dias atuais, a comunicação entre humanos e máquinas está altamente limitada a formas restritas, que envolvem linguagens de programação, comandos estruturados e dispositivos tradicionais como teclado, mouse e monitor de vídeo [HAU 99].

Diversas aplicações práticas estão diretamente ligadas à lingüística computacional, entre as quais podemos citar recuperação de informações em bases de dados, produção automática de texto e diálogo automatizado. Estas aplicações normalmente envolvem tarefas de decomposição de sinais em componentes, classificação de componentes e composição através de regras ou conhecimento.

A produção de fala de forma automática em sistemas computacionais é composta de duas atividades definidas como reconhecimento e síntese de fala. A atividade de reconhecimento de fala consiste em captar, reconhecer e interpretar sinais sonoros; em outras palavras, converter um sinal acústico, capturado por microfone ou telefone para um conjunto de palavras [LEM 2000]. A síntese de fala, por sua vez consiste na geração automática das formas de onda da voz, normalmente a partir de um texto escrito ou armazenado[LEM 2000]. Cada um destes processos é realizado a partir de um conjunto de atividades complexas do ponto de vista computacional e que requerem portanto o estudo e aplicação de técnicas avançadas para sua realização.

2.2 Aspectos básicos de um sistema de texto para fala

Um sistema de texto para fala (Text To Speech - TTS) tem como meta principal receber um texto de entrada, que pode ter sido inserido diretamente por um operador, obtido através de um scanner ou mesmo produzido de forma artificial por algum software e efetuar a síntese do texto [DUT 96]. Existem diferenças fundamentais entre sistemas para síntese de texto para fala e outras máquinas falantes como gravadores ou equipamentos que produzem sons a partir da concatenação de palavras ou sentenças isoladas com vocabulários limitados. Neste modelo de síntese, torna-se impossível armazenar todas palavras de uma determinada linguagem, portanto a tarefa central é a produção de fala através da automática fonetização de sentenças através da pronúncia [DUT 96].

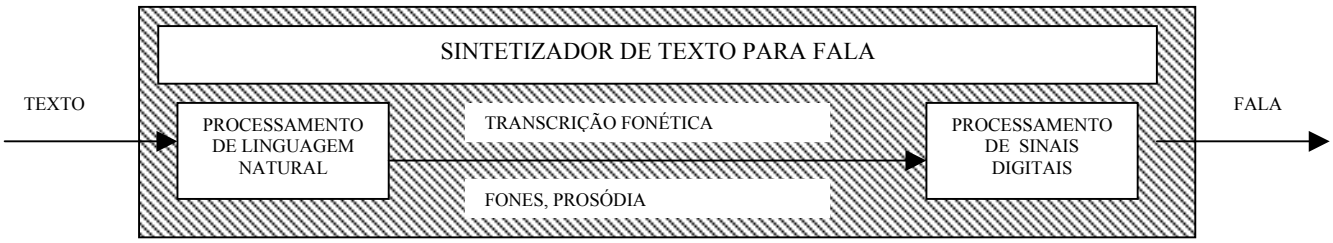


FIGURA 2.1 - Diagrama funcional de um sistema de texto para fala.

A fig. 2.1 mostra um diagrama funcional de um sintetizador de texto para fala, composto de dois grupos de atividades distintos. O módulo de processamento de linguagem natural (Natural Language Processing - NLP) deve ser capaz de produzir a representação fonética do texto, que inclui os sons da fala respectivos ao texto e também a entonação e ritmo necessários. O módulo de processamento de sinais digitais transforma a informação simbólica recebida em sinais de onda sonora que compõe a fala [DUT 96]. Estas duas fases são tradicionalmente chamadas de síntese de alto e baixo nível, respectivamente. Entre estas tarefas é possível distinguir claramente características segmentais e suprasegmentais, envolvidas. Qualidade segmental pressupõe a capacidade do módulo de linguagem natural de produzir sons da fala de alta qualidade, e suprasegmental refere-se à habilidade do módulo requerida para explorar a riqueza dos contornos prosódicos da fala. A naturalidade e inteligibilidade do resultado final está diretamente relacionada a ambos.

Para uma melhor compreensão das atividades a serem implementadas em um sistema para síntese de voz torna-se necessária antes de mais nada uma análise do processo da fala em seres humanos e da estrutura do aparelho fonador, responsável pela produção de fala [CHB 94].

Todos os seres humanos possuem mesmo que inconscientemente conhecimento acerca das regras para leitura e formação da fala em sua língua nativa; tais regras são transmitidas de forma simplificada nos primeiros anos escolares. O processo de leitura em humanos passa pela visão de unidades escritas, por uma atividade mental que traduz estas unidades em sons a serem produzidos pelo aparelho fonador em uma etapa final, como podemos verificar na fig. 2.2 [DUT 96]. Em uma comparação direta com as etapas de um sistema de texto para fala poderíamos considerar o cérebro como responsável pelo processamento de linguagem natural e o aparelho fonador como unidade de processamento de sinais.

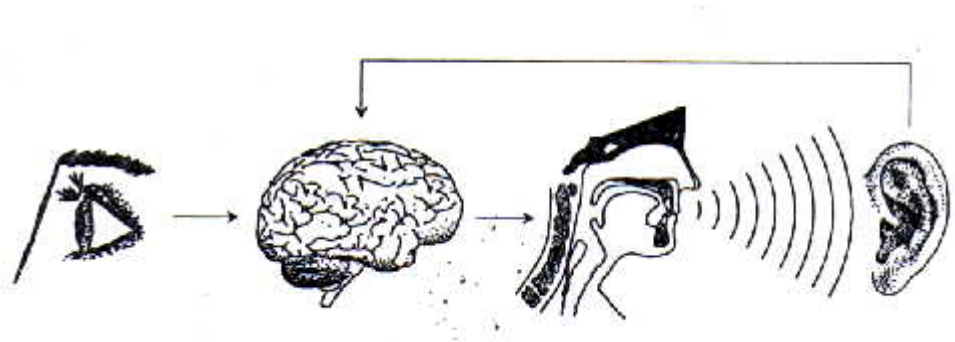


FIGURA 2.2 - Diagrama representando o fluxo do processo de leitura em humanos.

A compreensão do aparelho fonador é importante para entender os parâmetros envolvidos na produção da voz e por este motivo é ainda um tópico de diversas pesquisas na área de fonética [KLA 90]. As principais partes do aparelho fonador humano indicadas na fig. 2.3 são [CHB 94]:

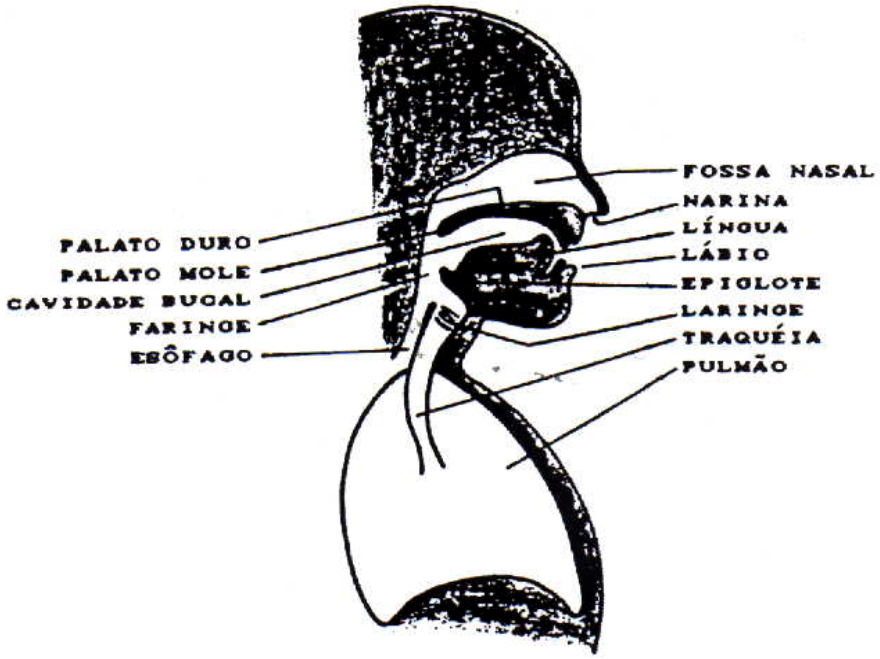


FIGURA 2.3 – Aparelho fonador humano

- pulmões, brônquios e traquéia são responsáveis pela corrente de ar, matéria-prima da fonação;
- na laringe encontram-se as cordas vocais, que produzem a energia sonora utilizada na fala
- cavidades supralaríngeas (faringe, boca e fossas nasais), funcionam como uma caixa de ressonância. A cavidade bucal pode variar profundamente em forma e volume, graças a movimentos de órgãos ativos, sobretudo da língua. Através da movimentação do palato mole (vélum), a cavidade nasal pode ser acoplada à cavidade bucal. Estas cavidades são também conhecidas como trato vocálico.

2.3 Fonética, fonologia e os sons da fala

Analisando de forma mais detalhada a funcionalidade do módulo de processamento de linguagem natural descrito na fig. 2.4, podemos concluir que o mesmo deve ser capaz de converter um texto em uma seqüência de fonemas, acompanhados da prosódia [DUT 96]. Esta transcrição obviamente não está explícita no texto a ser sintetizado e deve ser conseguida a partir de duas operações básicas, como mostra a fig. 2.4.

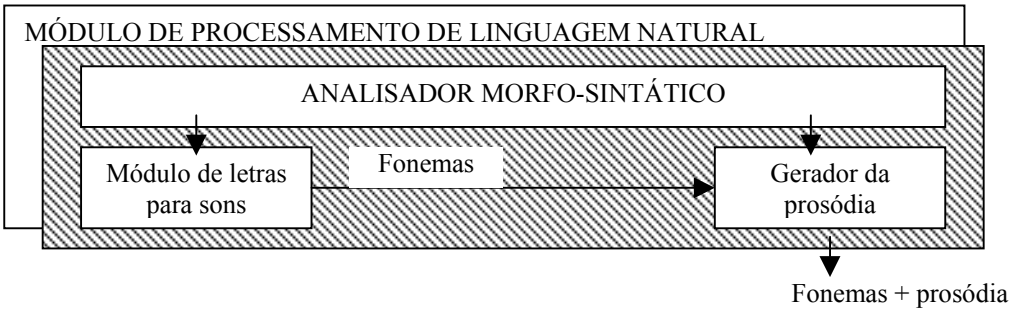


FIGURA 2.4 Diagrama funcional do módulo de processamento de linguagem natural

De acordo com [BIS 99] a fonética visa o estudo dos sons da fala do ponto de vista articulatório, verificando como os sons são articulados ou produzidos pelo aparelho fonador e do ponto de vista acústico, analisando a realização das propriedades físicas da produção e propagação dos sons. A fonologia por sua vez, dedica-se à compreensão dos sistemas de sons, sua descrição, estrutura e funcionamento, analisando a forma das sílabas, morfemas, palavras e frases, como se organizam e como se estabelece a relação "mente e língua". Estes sons são conhecidos como fonemas.

Os sons são gerados pela passagem do ar fornecido pelos pulmões através do trato vocálico, o que ocorre de três formas distintas originando sons sonoros ou vocálicos, fricativos e plosivos: i) Sons sonoros são produzidos pela elevação da pressão do ar nos pulmões, forçando sua passagem através do orifício das cordas vocais e causando assim sua vibração; ii) Sons fricativos são gerados pela formação de uma constrição em algum ponto do trato vocálico, normalmente nos lábios, forçando a passagem do ar através desta constrição com velocidade suficiente para produzir turbulência , criando assim uma fonte de "ruído branco"; iii) Sons plosivos resultam da constrição completa do trato vocálico em alguma parte, com acumulação, seguida de liberação abrupta de pressão. O ponto completo de fechamento pode ser efetuado em várias zonas de articulação e a excitação pode ou não causar vibração nas cordas vocais [CHB 94]. Uma descrição mais detalhada de sons e pontos de articulação pode ser encontrada em [SIL 99].

À medida que os sons gerados por uma das formas citadas propagam-se pelo trato vocálico, os mesmos apresentam alteração em seu espectro de freqüências, com ressonância em algumas delas. Estas freqüências são denominadas formantes do som, sendo o número de formantes variável de acordo com o som [CAM 80]. Juntamente com a freqüência fundamental, as formantes constituem os principais parâmetros acústicos da voz; tipicamente para uma voz masculina, a freqüência fundamental varia entre 60 e 240 Hz, enquanto as três formantes variam em torno de 500 Hz, 1500 Hz e 2500 Hz. Para a voz feminina, a freqüência fundamental tem valores entre 100 e 400 Hz, enquanto as formantes estão aproximadamente 10% acima das masculinas [KLA 90].

2.3.1 Transcrição fonética

Na maioria das linguagens, um texto escrito não corresponde a sua pronúncia; assim para descrever a pronúncia correta, um conjunto de representações simbólicas se faz necessário. Cada idioma possui um alfabeto fonético diferente e um conjunto de possíveis fonemas e suas combinações. Um conjunto de fonemas pode ser definido

como um número mínimo de símbolos necessários para descrever qualquer possível palavra em uma linguagem [LEM 2000].

Fonemas são unidades mentais abstratas das quais o som é a realização física; sua pronúncia depende dos efeitos contextuais e das características do locutor e suas emoções[LEM 2000]. Durante a fala contínua, os movimentos articulatórios dependem dos fonemas anteriores e posteriores. Os articuladores estão em diferentes posições dependendo do movimento anterior e da preparação para o seguinte, o que causa variações em como um fonema individual é pronunciado. Estas variações são chamadas alofones, constituindo um subconjunto dos fonemas e seu efeito é conhecido como coarticulação. Um exemplo clássico na maioria dos dialetos do idioma português brasileiro ocorre com os fonemas [t] e [d] que apresentam variação diante do [i], sendo possível perceber esta variação comparando-se palavras como "dia" e "dado". Cabe salientar entretanto que em regiões como por exemplo no nordeste do Brasil, a pronúncia é a mesma em ambas as situações.

O módulo para conversão de letra para fonema em sistemas de síntese tem portanto, a responsabilidade de determinar automaticamente a transcrição fonética do texto, que envolve primariamente a determinação de quais fonemas correspondem às letras. Pode-se imaginar a princípio que um simples processo de busca em dicionário seria suficiente, entretanto a partir de uma examinação detalhada percebe-se que diversas palavras podem ocorrer em diversos contextos, muitas das quais não constam em dicionários de pronúncia [DUT 96]. Além disso, a notação fonética nem sempre é perfeita, isto porque o sinal da fala é sempre contínuo e a notação fonética é sempre discreta.

Diversas convenções tem sido propostas com objetivo de permitir a transcrição fonética de sons em qualquer idioma. Uma das mais difundidas é o " International Phonetic Alphabet - IPA" proposto pela Sociedade Internacional de Fonética. O IPA entretanto emprega caracteres pouco comuns em máquinas de escrever e computadores, o que dificulta sua utilização em sistemas de texto para fala. O anexo 1 descreve os símbolos do IPA para a língua portuguesa.

O alfabeto fonético é usualmente dividido em duas categorias principais, vogais e consoantes. Vogais são sempre sons vozeados, pois são produzidos com as cordas vocais em vibração, enquanto que consoantes podem ser vozeadas ou não vozeadas. Vogais tem uma amplitude consideravelmente mais alta do que consoantes e também são mais estáveis e fáceis de analisar e descrever acusticamente [LEM 2000]. De acordo com [CAL 90] a língua portuguesa tem 26 fonemas segmentais. O acento é considerado como um fonema suprasegmental, como uma qualidade que se superpõe a certos segmentos.

2.3.2 Classificação das vogais

Vogais normalmente são classificadas segundo quatro critérios: quanto à região de articulação, quanto ao timbre, quanto ao papel das cavidades bucal e nasal e quanto à intensidade. Os três primeiros critérios são fundamentalmente de base articulatória e o último de base acústica[CHB 94].

a) Classificação quanto à região de articulação

Diz respeito ao ponto ou parte em que se dá o contato ou aproximação dos órgãos que cooperam para a produção dos fonemas. No caso das vogais, estes são a língua e

pálato. A vogal média [a] é produzida, mantendo-se a língua baixa, quase em posição de descanso e a boca entreaberta.

Para passar da vogal [a] para as anteriores [e], [é], [i] levanta-se gradualmente a parte anterior da língua em direção ao pálato duro, ao mesmo tempo que diminui-se a abertura da boca. Para emitir as vogais posteriores [o],[ó],[u] eleva-se a parte posterior da língua em direção ao véu palatino, arredondando progressivamente os lábios.

b) Classificação quanto ao timbre

Leva em consideração o maior ou menor grau de abertura dos lábios. Para vogal [a], a abertura é máxima e para vogais [e] e [i] o grau de abertura é mínimo.

c) Classificação quanto ao papel das cavidades bucal e nasal

É considerada aqui a posição da úvula durante a passagem de ar pelo trato vocálico. Quando a corrente sonora é impedida de ressoar na cavidade nasal devido à posição levantada da úvula, ocorre a produção das vogais orais ([a],[e],[é],[i],[o],[ó],[u]).

Estando as fossas nasais acopladas à cavidade bucal, através do abaixamento da úvula, parte da corrente sonora ressoa na cavidade nasal, produzindo as vogais nasais ([ã],[ẽ],[ĩ],[õ],[ũ]).

d) Classificação quanto à intensidade

Diz respeito, à qualidade física da vogal, que depende da força expiatória e da amplitude da vibração das cordas vocais. Vogais que encontram-se nas sílabas pronunciadas com maior intensidade chamam-se tônicas e caracterizam-se no idioma Português por um reforço da energia expiratória. Vogais que se encontram em sílabas não acentuadas denominam-se átonas.

TABELA 2.1 – Resumo da classificação das vogais [CHB 94]

Intensidade	Papel das cavidades bucal e nasal	Timbre	Região de articulação		
			Anteriores ou palatais	Médias ou centrais	Posteriores ou velares
Tônicas	Orais	Fechadas	[i]		[u]
		Semi-fechadas	[e]		[o]
		Semi-abertas	[é]		[ó]
		Abertas		[a]	
	Nasais	Fechadas	[ĩ]		[ũ]
		Semi-fechadas	[ẽ]	[ã]	[õ]
Átonas	Orais	Fechadas	[i]		[u]
		Semi-fechadas	[e]		[o]
		Abertas		[a]	

2.3.3 Classificação das consoantes

As consoantes da Língua Portuguesa tradicionalmente são classificadas em função de quatro critérios de base articulatória, ou seja, quanto ao modo de articulação, ao ponto de articulação, a função das cordas vocais e quanto ao papel das cavidades bucal e nasal. Vai-se analisar agora, em detalhes estas diferentes classes.

a) Classificação quanto ao modo de articulação

Diz respeito à maneira pela qual os fonemas consonantais são articulados. Vindo da laringe, a corrente de ar chega à boca, onde encontra obstáculos totais ou parciais da parte dos órgão bucais. Se o fechamento dos lábios ou a interrupção da corrente do ar é total, são produzidas as consoantes oclusivas ([p],[t],[k],[b],[d],[g]).

Se o fechamento for parcial são produzidas as consoantes constritivas, que podem ser: fricativas, vibrantes ou laterais. As fricativas ocorrem quando o trato vocálico é excitado por um fluxo de ar turbulento, passando a corrente expiratória pela constrição ([f],[s],[x],[v],[z],[j]). Vibrantes caracterizam-se pelo movimento vibratório rápido da língua [r], ou da úvula [R], que corresponde ao som de "rr". As laterais, são produzidas pela passagem da corrente expiratória pelos dois lados da cavidade bucal, em virtude de um obstáculo formado no centro desta, pelo contato da língua com os alvéolos dos dentes [l], ou com o pálato [L].

b) Classificação quanto ao ponto de articulação

Faz referência ao local onde os órgãos fonadores entram em contato para a emissão do som, podendo ser bilabiais ([p],[b],[m]), labidentais ([f], [v]), lingüodentais ([t],[d],[s],[z]), alveolares ([l],[r],[n]) ou velares ([k],[g])).

c) Classificação quanto a função das cordas vocais

Quanto às vibrações das cordas vocais, produzem-se consoantes sonoras, quando ocorre vibração ou consoantes surdas caso contrário.

d) Classificação quanto ao papel das cavidades bucal e nasal

Havendo uma saída do ar exclusivamente pela boca, as consoantes são ditas orais e havendo uma penetração de ar nas fossas nasais, pelo abaixamento da úvula, as consoantes são nasais ([m],[n],[ñ]).

A tab. 2.2 resume a classificação das consoantes.

TABELA 2.2 - Resumo da classificação das consoantes

Função das cavidades bucal e nasal		Orais						Nasais
Modo de articulação		Oclusivas		Constritivas				Oclusivas
				Fricativas		Vibrantes	Laterais	
Função das cordas vocais		Surdas	Sonoras	Surdas	Sonoras	Surdas	Sonoras	Sonoras
Ponto de articulação	Bilabiais	[p]	[b]					[m]
	Labio-dentais			[f]	[v]			
	Lingüo-dentais	[t]	[d]	[s]	[z]			
	Alveolares					[l]	[r]	[n]
	Palatais			[x]	[j]	[L]		[ñ]
	Velares	[k]	[g]				[R]	

2.4 Técnicas para síntese

As técnicas para síntese de fala, podem ser classificadas como acústica ou articulatória. Na síntese acústica, também chamada de funcional o processo concentra-se em aspectos do som produzido e na sua reprodução; normalmente através de amostras pré-gravadas [CAR 98]. Além das técnicas de síntese acústica, outra forma possível de produzir fala é a modelagem direta do sistema humano de produção de fala. Este método, chamado de síntese articulatória, envolve modelagem de articulações humanas, incluindo também cordas vocais. O estudo, portanto, concentra-se na causa da fala, no estudo da anatomia e fisiologia do aparelho fonador humano. Entretanto, embora este método permita uma alta qualidade no resultado final, possui uma grande complexidade potencial para sua implementação [LEM 2000].

As principais técnicas utilizadas nos sistemas atuais são a síntese por concatenação, por formantes, predição linear e fonte-filtro e serão descritas a seguir.

2.4.1 Concatenação

Sintetizadores concatenadores executam o processamento dos sinais da fala a partir de um banco de dados de falas naturais [CAR 98]. As principais características fonológicas da linguagem devem estar armazenadas no banco de dados em questão. A atividade básica envolvida no processo, consiste em concatenar segmentos e suavizar as transições entre eles. O segmento a ser usado varia e pode ser escolhido entre palavras, sílabas, fonemas, difones e demi-sílabas.

Para que a concatenação possa ser realizada, é necessário anteriormente um processamento do texto de entrada para identificação dos segmentos que o compõe

(sílabas, fonemas). A decisão de utilizar um ou outro segmento passa por uma análise que leve em consideração as seguintes características: i) Tamanho do vocabulário; ii) relação entre qualidade do resultado final; iii) capacidade de armazenamento e; iv) tempo necessário para recuperação dos segmentos a partir do banco de dados. Cada um dos segmentos citados possui semelhanças e diferenças e apresenta algumas vantagens e desvantagens quando do seu uso [CAR 98].

2.4.2 Síntese por sílaba

Assim como na utilização de palavras, o número de sílabas existentes na linguagem é um problema considerável na maioria dos idiomas. Entretanto, devido ao pequeno número de interrupções e da diminuição sensível do problema de coarticulação, verificado na síntese por fonemas em geral, o resultado final é de boa qualidade e nitidez [CAR 98].

2.4.3 Síntese por fonemas

Os fonemas são a menor unidade da fala e correspondem efetivamente aos sons produzidos. A utilização destes como segmento para síntese foi a primeira alternativa para contornar o problema do grande número de palavras e sílabas. Entretanto, mesmo que a unidade de concatenação escolhida não seja o fonema em si, a conversão de texto para sons da fala, em nível de fonema, é necessária. [CAR 98].

2.4.4 Síntese por Difone

O difone representa a transição entre segmentos adjacentes[KLA 90]. A utilização de fonemas, causa problemas de clareza na transição entre um fonema e outro. O emprego de difones, tem por objetivo minimizar este problema, que ocorre principalmente devido ao fenômeno da coarticulação. A coarticulação não é considerada um som e sim a adaptação da posição dos órgãos vocais, com objetivo de pronunciar o fonema seguinte, a partir da posição em que se encontram, em razão do fonema pronunciado.

A gravação dos difones engloba os segmentos da consoante com a vogal seguinte, bem como a mesma vogal com a consoante seguinte. Tais gravações, trazem embutidas a coarticulação necessária para maior clareza no resultado final.

2.4.5 Síntese por Demi-sílaba

A demi-sílaba é sutilmente diferente do difone. Enquanto em um difone, a separação é realizada no meio da vogal, na demi-sílaba ela é feita tão logo ocorra a transição da consoante para a vogal; ou seja, um pouco antes do corte difônico.

2.4.6 Formantes

As formantes são zonas onde se verificam marcas frequenciais de maior intensidade, podendo ser consideradas como identidades do fonema e da fala. São divididas em 3 bandas diferentes (200-1000, 500-2500, 1500-3500 Hz). Em cada uma das bandas o ponto de maior freqüência é denominado formante, não sendo portanto

necessário o armazenamento de toda a onda sonora representando o fonema; apenas as formantes que permitem caracterizá-lo. A quantidade de espaço necessária para o armazenamento neste caso é infinitamente inferior às alternativas anteriores.

2.4.7 Fonte-Filtro

Esta técnica baseia-se na premissa de que a região vocal pode ser modelada como um filtro linear que varia no tempo[CAR 98]. O filtro é excitado por uma fonte que gera pulsos de onda, modelando assim o som e produzindo o fonema específico. Em uma analogia com o aparelho vocal humano, podemos comparar a fonte ao pulmão e o filtro às cordas vocais, lábios e cavidades orais e nasais.

2.5 Conversão letra para fonema em Língua Portuguesa

Após a conceituação de fonemas e da classificação de acordo com aspectos relacionados à produção física dos sons, torna-se necessário abordar o problema sob o ponto de vista do processo de conversão, e de que maneira ele pode ser realizado computacionalmente. Tradicionalmente, existem duas estratégias possíveis para realização desta atividade, os quais são, processos baseados em dicionário ou baseados em regras.

A estratégia baseada em dicionário consiste no armazenamento de um máximo conhecimento fonológico em um léxico. Para não implicar em tamanhos que inviabilizem o processo, entradas são geralmente restritas à morfemas, derivações destes ou ainda regras que definem como componentes morfêmicas são modificadas, quando combinadas em palavras [DUT 96].

Outra estratégia adotada é baseada em conjuntos de regras para conversão de letras para sons (LTS – Letter To Sound), também chamadas de letra-fonema ou grafema-fonema. Nesta estratégia apenas palavras pronunciadas de forma diferente das regras em determinados contextos, devem ser armazenadas em um dicionário de exceções. Esta abordagem permite a expressão de diferentes regras de acordo com variações regionais, extremamente comuns na Língua Portuguesa. O conjunto de regras a serem mantidas é sensivelmente menor do que o dicionário léxico e ainda torna-se possível representar um número quase infinito de palavras e combinações derivadas destas regras, o que torna esta abordagem mais adaptada do que a conversão baseada em dicionário [DUT 96].

A conversão de letras para fonemas na maioria dos casos ocorre em nível de palavras; sendo assim, inicialmente um pré-processamento no texto de entrada deve ser realizado, visando traduzir ou eliminar elementos como abreviações, acrônimos, datas, números ou outros elementos semelhantes. A partir desta etapa, o texto será separado em palavras e cada uma delas passará pelo processo de conversão de letras para fonemas. Embora em alguns contextos possa haver uma influência da palavra anterior sobre o início da próxima, a escolha de palavras como unidades de conversão é a mais natural e simples do ponto de vista de sistemas de texto para fala automáticos [DUT 96].

Na maioria dos idiomas, assim como na língua portuguesa, não há uma correspondência exata entre número de letras e de fonemas. As regras para conversão tipicamente envolvem avaliação de fonemas anteriores e posteriores. Particularmente na Língua portuguesa, os seguintes casos podem ser citados como exemplos [CHB 94]:

- Uma mesma letra simbolizando mais de um fonema (e**x**ame, **x**ale, pró**x**imo).
- Um mesmo fonema representado por mais de uma letra (ca**s**a, co**z**inha, e**x**ame).
- Letras que por vezes não representam fonemas, funcionado como notações léxicas (ca**m**po, reg**u**e).
- Casos em que letras não representam fonemas e tampouco notações léxicas (**h**otel, di**sc**ípulo).

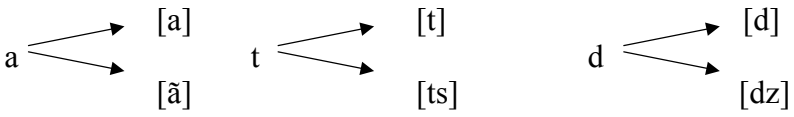
Vários autores tem proposto ao longo do tempo, conjuntos de regras para guiar a conversão; entretanto é impossível cobrir todos os aspectos e contextos verificados na linguagem. De acordo com [SIL 93] as regras para conversão podem ser reunidas em dois grandes grupos: símbolos gráficos com representação única e com representação múltipla. Este último grupo pode ser dividido em diversos subgrupos:, envolvendo elementos gráficos (letras) que podem representar mais de um elemento fônico (fonema) ou fonemas passíveis de representação por mais de uma letra. A grande maioria das regras concentram-se no segundo grupo; uma vez que existindo uma representação biunívoca entre letra e fonema, basta produzir como resultado a própria letra. Os diferentes subconjuntos de regras estão descritos abaixo[SIL 93]:

Grupo 1 - Relações biunívocas entre letra e fone. A combinação "nh" tem uma representação única, por este motivo encontra-se também neste grupo.

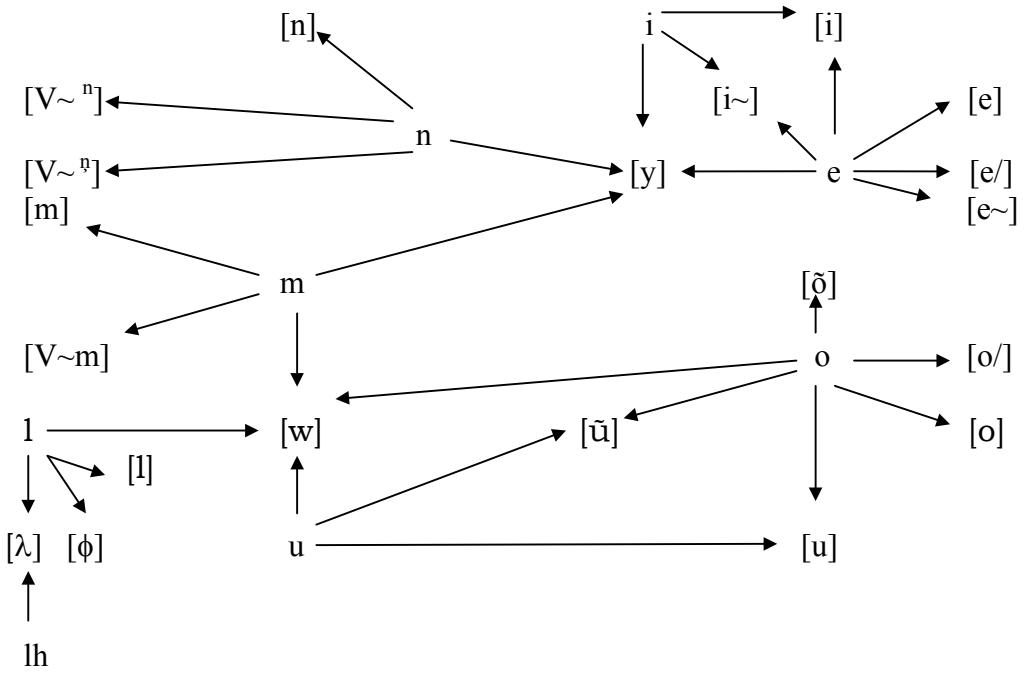
p → [p]
b → [b]
f → [f]
v → [v]
nh → [ñ]

Grupo 2 - Representações múltiplas

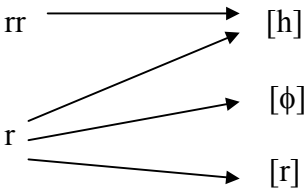
O subgrupo 1 descreve símbolos gráficos com valor fonológico múltiplo simples. Este grupo pode ser considerado como uma variação do anterior, onde algumas letras possuem além da representação direta do fonema básico, uma pequena variação em contextos ou até mesmo em regiões específicas.



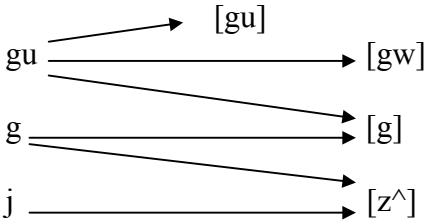
O subgrupo 2 descreve representações múltiplas de letras e fonemas. A maioria das letras neste grupo pode representar mais de um fonema. Alguns fonemas como [u], são representados por mais de uma letra e letras como "m" ou "n" podem simbolizar um fonema ou ainda influenciar a representação da letra anterior, como no caso da nasalização.



Subgrupo 3 - Representações da letra "r". A letra em questão pode ser considerada um caso especial, pois a ocorrência da mesma em seqüência exige a produção de um fonema diferente. Os fonemas neste subgrupo não possuem uma representação por mais de uma letra.

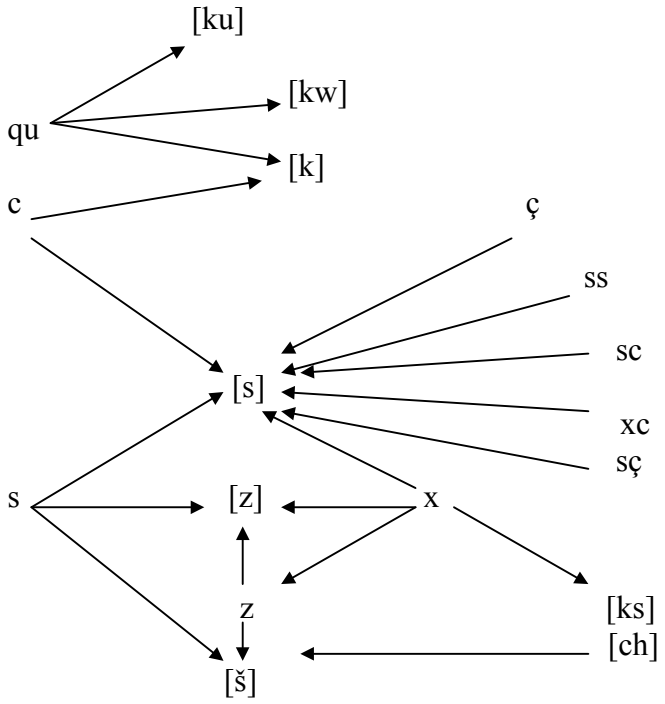


Subgrupo 4 - Representações das letras "g" e "j". Nesta situação, observa-se claramente uma influência das letras seguintes na produção do fonema correto. A letra "u" em certos contextos é representada em outros não. A letra "j" possui uma representação única, porém o fonema produzido por ela possui representação também pela letra "g".



Subgrupo 5 - Representações múltiplas de "q", "c", "s", "z". O nível de complexidade neste caso é grande, existem fonemas simbolizados por mais de uma letra

e a combinação de uma mesma letra simbolizando diferentes fonemas. A letra “x” pode representar diferentes fonemas, de acordo com o contexto que é utilizada.



A partir dos contextos e grupos descritos acima, analisando-se exemplos de palavras onde sejam verificadas as diferentes situações, é possível deduzir as regras para conversão de letras para fonemas. A descoberta das regras neste trabalho deverá portanto, envolver um número mínimo de situações de cada um dos grupos descritos.

2.6 Histórico e sistemas para síntese de fala

Diversos esforços tem sido realizados ao longo dos anos para criar sistemas para produção de fala. Inicialmente foram criados dispositivos mecânicos semelhantes a instrumentos musicais, cujo objetivo era a produção de sons de vogais. Estes primeiros dispositivos foram desenvolvidos por Kratzenstein em 1779. Este trabalho foi estendido e culminou 20 anos após com a criação da máquina de fala de Kempelen em Viena, que ainda constituía-se de um dispositivo mecânico, baseado na produção de sons a partir da pressão do ar acionado manualmente por operadores humanos.

A partir de 1922 foram descritos e produzidos os primeiros sintetizadores elétricos, sendo que o primeiro a ser considerado um sintetizador de fala foi o VODER (Voice Operating Demonstrator) introduzido por Homer Dudley e desenvolvido pelos laboratórios da empresa Bell. Após a demonstração do VODER o trabalho científico em síntese de voz passou a receber uma maior atenção e um incremento no interesse, pois a partir daí foi demonstrado que fala inteligível poderia ser produzida artificialmente [LEM 2000].

O primeiro sintetizador por formantes foi o PAT (Parametric Artificial Talker) introduzido em 1953 por Walter Lawrence, o qual consistia de componentes para ressonância conectados em paralelo. Um dispositivo de vidro em movimento convertia sinais desenhados em funções para controle de três freqüências formantes: i) Amplitude de vozeamento; ii) freqüência fundamental e; iii) amplitude de ruído [LEM 2000].

O primeiro sintetizador articulatório, o DAVO (Dynamic Analog of the Vocal tract), foi introduzido em 1958 por George Rosen no M.I.T. e era controlado por fitas com sinais de controle criados e gravados manualmente. Entre 1970 e 1980 uma considerável quantidade de sistemas comerciais de texto para fala tornou-se disponível, sendo que o primeiro circuito integrado para síntese de fala pode ser considerado o Votrax, que possuía um chip que implementava sintetizadores em cascata[LEM 2000].

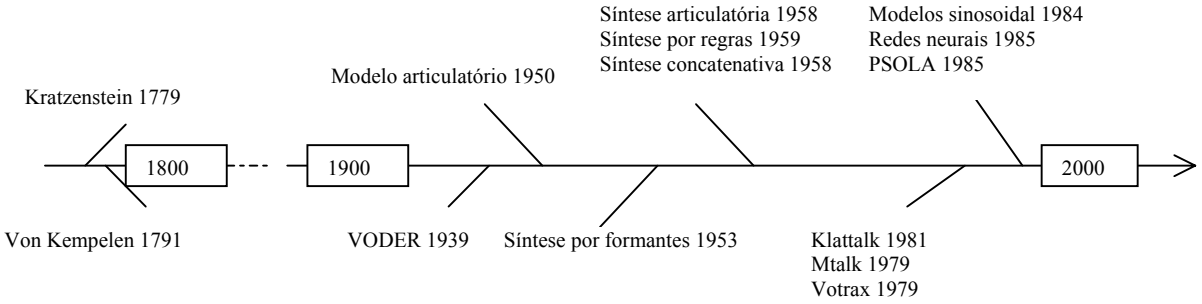


FIGURA 2.5 Evolução histórica da síntese de fala.

A fig. 2.5 ilustra a evolução dos sistemas para síntese. Tecnologias modernas para síntese de fala envolvem métodos e algoritmos sofisticados e complexos. Alguns destes métodos são os Modelos Ocultos de Markov(Hidden Markov Models - HMM), utilizados desde 1970 e as Redes Neurais Artificiais, cuja utilização é mais recente. A maioria das técnicas atuais emprega princípios de inteligência artificial e aprendizado de máquina, principalmente na fase de processamento de linguagem natural.

Diversas aplicações comerciais atuais possuem um grande potencial para uso da síntese de texto para fala. Dentre estas aplicações podemos citar serviços de telecomunicações, envolvendo respostas e atendimento telefônico automático, ensino de línguas apoiado por sistemas computacionais envolvendo diálogo, interfaces para usuários que possuem deficiências motoras ou visuais, multimídia e todas as formas de pesquisas e sistemas envolvendo comunicação entre humanos e máquinas [LEM 2000].

2.6.1 Sintetizadores atuais

Neste item serão descritos de forma sucinta alguns sintetizadores utilizados em larga escala.

a) Dosvox

É um dos mais conhecidos sintetizadores de fala em Língua Portuguesa [DOS 98]. Baseia-se na concatenação de fonemas e possui dois componentes: Subsistema de conversão e subsistema de conversão e amplificação.

O subsistema de conversão recebe uma palavra e através de um conjunto de regras, representadas por uma máquina de estados finitos, retorna os respectivos fonemas que compõe a mesma. O subsistema de conversão e amplificação emite os sons seqüencialmente dos fonemas obtidos pelo primeiro subsistema, em um dispositivo de saída [DOS 98].

b) Mbrola

O sintetizador de fala Mbrola [MBR 2000] foi elaborado com o objetivo de servir para síntese em qualquer idioma. A ferramenta baseia-se em um banco de dados fonético específico para cada idioma.

Não existe neste sistema a necessidade de tradução da escrita para fonemas, parte-se diretamente destes. O funcionamento básico consiste na concatenação de difones e na descrição de cada um destes quanto à prosódia.

2.6.2 Projeto Spoltech

Spoltech é um projeto de pesquisa em lingüistica computacional que tem por objetivo criar, desenvolver e prover tecnologias de síntese e reconhecimento de fala em língua portuguesa. O projeto é de cooperação internacional, através do CNPq (Brasil) e NSF(Estados Unidos), sendo composto por pesquisadores, estudantes do Instituto de Informática e Instituto de Letras da Universidade Federal do Rio Grande do Sul, do Departamento de Informática da Universidade de Caxias do Sul, CSLR/CU (University of Colorado, Boulder) e do CSLU/OGI (Oregon Graduate Institute). A coordenação do projeto, recai sobre o Prof. Dr. Dante A. C. Barone, orientador do presente trabalho [SPO 2001].

Para síntese de texto para fala é utilizado o sistema Festival, que consiste de um framework para construção de sistemas para síntese, desenvolvido pela University of Edimburgh(Escócia). A ferramenta Festival oferece a possibilidade da adição rápida e eficiente de novos módulos; sendo assim, não é necessário desenvolver a arquitetura completa do sistema de síntese e sim os módulos específicos para síntese no idioma em questão ou ainda módulos relativos a outras técnicas de síntese que estejam sendo pesquisadas.

O processo emprega a síntese acústica por concatenação, onde a unidade empregada no processo é o difone. Duas diferentes bases de difones foram criadas, sendo a segunda mais completa. A lista de difones é uma combinação de todos os fonemas observados no idioma. Portanto para possibilitar a definição da base de difones e construir o processo de síntese foram inicialmente definidos os fonemas do idioma português brasileiro [SPO 2001]. O Anexo 1 mostra a tabela de fonemas empregada na ferramenta Festival e sua representação equivalente no "IPA". Esta mesma notação foi utilizada na aplicação descrita neste trabalho.

O processo de conversão de letra para fonema é implementado através de um conjunto de regras descritas em um módulo LTS, conhecido como módulo de letras para sons. Abaixo podemos visualizar dois exemplos de regras criadas no ambiente do sistema Festival. A primeira define que ocorrendo a letra “m” após “o”, em final de palavra, simbolizado por #, será produzido o fonema [o~]. A segunda estabelece a produção do fonema [s] sempre que ocorrer a letra “c” antes de “e” ou “i”.

([o m] # = o~)	([c] EI = s)

As regras implementadas efetuam corretamente a conversão na maioria dos casos; entretanto alguns problemas ocorrem:

- Palavras com letras “e” e “o”. Em diversas situações, as palavras representam diferentes fonemas sem que haja possibilidade de definir regras aplicáveis a todas as situações. Para melhor conversão nestes casos foi realizada uma análise através de um dicionário de pronúncia com aproximadamente 20.000 palavras. A partir do levantamento das principais ocorrências verificadas, foram criadas regras para os casos mais comuns.

- Palavras homógrafas. Todas as palavras deste gênero constituem uma situação extremamente difícil de resolver por não haverem regras gerais que funcionem para qualquer contexto.
- Ocorrências da letra x. Nas regras implementadas não é possível detalhar todos os casos em que a letra aparece, por serem muitos.

A partir da descrição das atividades necessárias para síntese de fala, é possível perceber as dificuldades envolvidas para a realização completa deste processo. A atividade de conversão de letra para fonema é de vital importância e apresenta variações de acordo com contexto, idioma e região. Tais variações justificam a aplicação de técnicas avançadas com objetivo de diminuir a necessidade de alterações e adaptações manuais em sistemas computacionais para síntese de fala.

3 Programação genética

3.1 Princípios da evolução natural

A teoria da evolução humana foi proposta por Darwin em 1859 e é aceita até hoje como a teoria biológica mais importante para explicação da origem e perpetuação da vida. De acordo com Dobzanhansky [DOB 77], "Nada na biologia faz sentido, exceto a luz da evolução", tal afirmativa serve de exemplo da importância que a maioria dos pesquisadores atribui a teoria.

De acordo com Darwin [FOG 2000], a história da vasta maioria das espécies é totalmente contada por alguns poucos processos estatísticos que agem sobre populações e espécies. Estes processos são a reprodução, mutação, competição e seleção. A reprodução é o mais conhecido e óbvio processo para perpetuação da vida e para evolução; a mutação é garantida em um sistema que continuamente reproduz ele mesmo em um universo positivamente entrópico. A competição e seleção são conseqüências inescapáveis da expansão de populações que ocupam uma arena finita, uma vez que as espécies e indivíduos competem por recursos. A evolução portanto, é o resultado desta fundamental interação entre estes processos que atuam sobre populações, geração após geração [FOG 2000].

Algumas questões podem ser formuladas em relação a como tais processos agem sobre uma evolução: O que está sendo otimizado pela evolução? Quão extensiva é esta otimização? Quais são os benefícios de otimização resultantes: para indivíduos ou para as espécies como um todo? Quais os benefícios da seleção sexual? Os mecanismos evolucionários criam um emaranhado de efeitos que não podemos distinguir ou particionar facilmente, mas que governa todo o processo evolucionário e pode ser expresso por um pequeno conjunto de regras [FOG 2000].

Organismos vivos podem ser vistos com uma dualidade de genótipo (a codificação genética básica) e seu fenótipo (a maneira que ele responde, suas atividades, psicologia e morfologia). De acordo com a teoria da evolução, a seleção natural é a força evolucionária que prevalece na formação das características fenotípicas de um indivíduo na maior parte das situações encontradas na natureza. A variação deste fenótipo é um resultado de larga atuação dos processo de mutação e recombinação. A interação entre espécies e seu ambiente (incluindo outros organismos) determina o sucesso ou falha, que pode significar a morte de espécies[FOG 2000].

A seleção pode ser vista como guia para manutenção ou incremento da habilidade de sobreviver e reproduzir em um ambiente específico. Em um processo de evolução natural entretanto, há enormes dificuldades de medir diretamente esta habilidade. A reprodução sexual por sua vez, oferece uma maneira significativa para geração de diversidade genética e como conseqüência, diversidade fenotípica; há um incremento na taxa de exploração do espaço de estados genotípicos/fenotípicos possíveis [FOG 2000].

Uma das tendências da evolução, é a geração de organismos com maior complexidade, onde indivíduos tendem a apresentar uma maior capacidade de interação com o ambiente e maior facilidade de resolver problemas. Este incremento de complexidade, é necessário para aumentar a qualidade na atividade de antecipação a eventos ambientais [FOG 2000].

3.2 Computação evolucionária

A idéia de simular inteligência e aprendizado em computadores utilizando princípios evolutivos pode ser observada em um conjunto de técnicas propostas. Além da Programação Genética, podemos citar ainda a Programação Evolucionária, Vida Artificial, Estratégias Evolucionárias e Algoritmos Genéticos. Estas técnicas envolvem a evolução de programas, simples padrões numéricos, máquinas de estados ou indivíduos formados por strings de bits [FOG 2000].

3.2.1 Algoritmos Genéticos

Algoritmos Genéticos (AG) foram propostos inicialmente por John Holland [HOL 75] e desde então vem sendo aplicados com sucesso em uma grande quantidade de problemas, principalmente na área de otimização [DAV 91], [MIT 92].

Essencialmente a técnica engloba uma população de indivíduos codificados como cromossomos, tipicamente usando uma representação binária, cada indivíduo da população representa uma solução para o problema em questão. A utilização de representações em níveis de abstração mais altos tem sido investigada. Como estas representações são mais fenotípicas, facilitariam sua utilização em determinados ambientes, onde essa transformação "fenótipo - genótipo" é muito complexa [MIT 92].

Para medir a adaptação destes indivíduos ao ambiente (problema a ser resolvido) é definida uma função de fitness, que é aplicada a cada solução candidata. Um critério de seleção vai fazer com que, depois de muitas gerações, o conjunto inicial de indivíduos gere indivíduos mais aptos. A maioria dos métodos de seleção são projetados para escolher preferencialmente indivíduos com maiores notas de aptidão, embora não exclusivamente, a fim de manter a diversidade da população. Um método de seleção muito utilizado é o método da roleta, onde indivíduos de uma geração são escolhidos para fazer parte da próxima geração, através de um sorteio de roleta [MIT 92].

Neste método, cada indivíduo da população é representado na roleta proporcionalmente ao seu índice de aptidão. Assim, aos indivíduos com alta aptidão é dada uma porção maior da roleta, enquanto aos de aptidão mais baixa é dada uma porção relativamente menor da roleta. Finalmente, a roleta é girada um determinado número de vezes, dependendo do tamanho da população, e são escolhidos, como indivíduos que participarão da próxima geração, aqueles sorteados na roleta [MIT 92].

Indivíduo S_i	Aptidão $f(S_i)$	Aptidão Relativa
S_1 10110	2.23	0.14
S_2 11000	7.27	0.47
S_3 11110	1.05	0.07
S_4 01001	3.35	0.21
S_5 00110	1.69	0.11

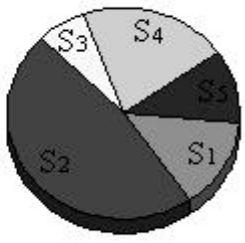


FIGURA 3.1 - Indivíduos de uma população e a sua correspondente roleta de seleção

Um conjunto de operações é necessário para que, dada uma população, se consiga gerar populações sucessivas que espera-se melhorem sua aptidão com o tempo. Estes

operadores são: cruzamento (fig.3.2) e mutação. Eles são utilizados para assegurar que a nova geração seja totalmente nova, mas possua, de alguma forma, características de seus pais, ou seja, a população se diversifica e mantém características de adaptação adquiridas pelas gerações anteriores. Para prevenir que os melhores indivíduos não desapareçam da população pela manipulação dos operadores genéticos, eles podem ser automaticamente colocados na próxima geração, através da reprodução elitista, que consiste em manter o melhor indivíduo de uma geração para outra [MIT 92].

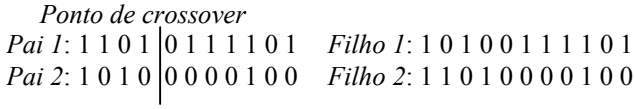


FIGURA 3.2 - Operação de crossover em AG

Um Algoritmo Genético opera sobre um conjunto de soluções, e desta maneira pode explorar diferentes regiões do espaço de busca, ao contrário de técnicas tradicionais como Subida de Montanha (*Hill Climbing*) e Recozimento Simulado (*Simullated Annealling*) [MIT 92].

O Algoritmo Genético tradicional é representado pelos seguintes passos:

{ t = 0;

inicia_população (P, t)

avaliação (P, t);

repita até (t = d)

{ t = t +1;

seleção_dos_pais (P,t);

recombinação (P, t);

mutação (P, t);

avaliação (P, t);

sobrevivem (P, t) } }

onde:

t - tempo atual;

d - tempo determinado para finalizar o algoritmo;

P - população

Neste caso, t pode ser definido como a geração atual e d como o número de gerações que serão processadas. Pode-se usar outros parâmetros para encerrar o processamento, sem se ater exclusivamente a um número fixo de gerações. O processo de avaliação, é baseado em uma medição do fitness, que corresponde ao grau de adaptação de cada indivíduo[MIT 92].

3.3 Programação Genética para descoberta de programas

A programação genética tem sido considerada por diversos autores como uma extensão ou variação dos algoritmos genéticos devido à semelhança entre as duas abordagens. A principal diferença entre ambas está localizada na forma de representação das soluções. Enquanto os Algoritmos Genéticos utilizam uma

representação abstrata e altamente estruturada, a Programação Genética apresenta como solução programas de computador em uma linguagem de programação específica. O objetivo central portanto na Programação Genética é descobrir uma seqüência de instruções que possa resolver o problema proposto.

3.3.1 Descoberta de programas

Muitos problemas semelhantes em aprendizado de máquina, inteligência artificial e processamento simbólico podem ser vistos como uma busca ou descoberta de um programa de computador que produza determinadas saídas a partir de um conjunto específico de entradas [BRA 92].

A partir da hipótese de que um programa correto possa ser induzido a partir de um conjunto de casos de testes, podemos considerar esta atividade como uma tarefa de indução, que por sua vez está sujeita a todos os méritos e armadilhas do raciocínio indutivo. Um dos principais problemas, é que uma hipótese indutiva está ligada diretamente a capacidade que casos de teste tem de representar o problema inteiro. A maioria dos problemas possui um conjunto grande de hipóteses possíveis [O'RE 95].

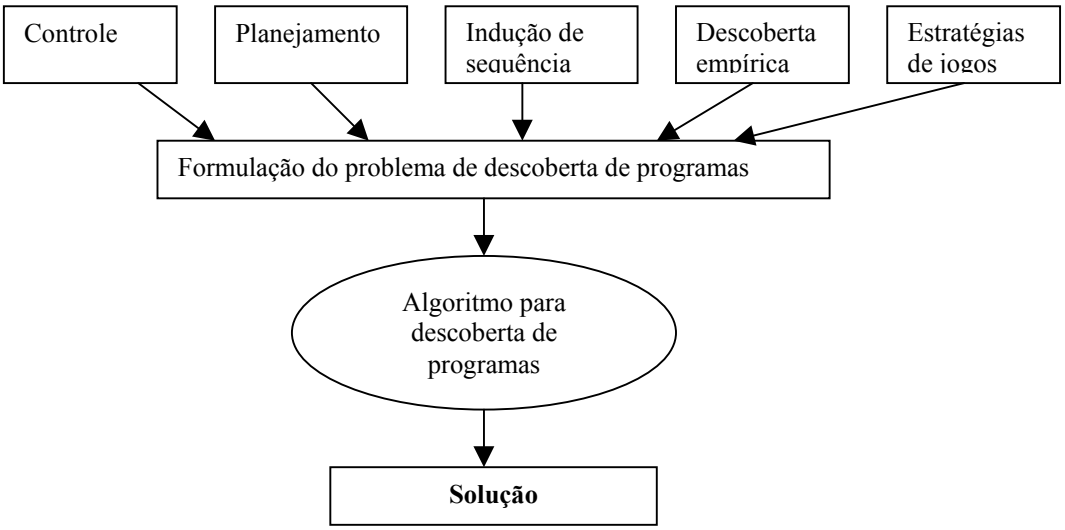


FIGURA 3.3 - Reformulação de problemas como uma descoberta de programas

O que torna a descoberta de programas especial e diferente de outras tarefas indutivas é que as soluções são expressas na forma de programas, que são soluções utilizáveis para especificação de uma atividade para uma classe geral de problemas, pela possibilidade de parametrização através do uso de variáveis. Assim, um mesmo programa é aplicável a uma classe geral e não a um único problema, bastando que se altere o conjunto de parâmetros de entrada, para que se obtenha outra saída [O'RE 95].

Vários são os domínios onde a descoberta automática de programas pode ser aplicada, dentre estes podemos citar: Estratégias de controle, planejamento, indução de seqüência, regressão simbólica, estratégias de jogos, descobertas empíricas, previsões e programação automática [O'RE 95].

3.3.2 Atributos de um sistema para descoberta automática de programas

De acordo com [KOZ 2000], um sistema para criação automática de programas deveria ser capaz de criar entidades que possuem a maioria ou preferencialmente todas as capacidades descritas nos atributos abaixo:

1. Iniciar com o que necessita ser feito: Instrução de alto nível especificando requerimentos para o problema

2. Dizer-nos como fazê-lo: Produzir um conjunto de instruções na forma de uma seqüência de passos que resolvem satisfatoriamente o problema

3. Produzir um programa de computador:

4. Determinar automaticamente o tamanho do programa: Determinar de forma automática o número de passos que devem ser executados sem que devam ser pré-especificados pelo usuário.

5. Reuso de código: Possuir a habilidade de automaticamente organizar grupos usáveis de código que podem ser reutilizados.

6. Reuso parametrizado: Criar unidades de código que possam ser reusadas com diferentes instanciações dos valores (parâmetros formais).

7. Armazenamento interno: Utilizar armazenamento interno, através de variáveis, vetores, matrizes, pilhas e outras estruturas de dados.

8. Iterações, laços e recursões:

9. Organização própria de hierarquias: Habilidade de organizar automaticamente grupos de passos em uma hierarquia

10. Determinar de forma automática a arquitetura do programa:

11. Usar ampla faixa de elementos de programação: Implementar elementos análogos aos usados por programadores humanos, e que incluem macros, bibliotecas, tipos, ponteiros, operações condicionais, funções lógicas, funções inteiras, múltiplas entradas e saídas e outros.

12. Operar em uma forma bem definida: Distinguir de forma correta entre o que o sistema precisa prover e o que ele produz.

13. Ser independente de problema: Independente, no sentido de que o usuário não necessita modificar os passos do sistema para cada novo problema.

14. Ampla aplicabilidade: Produzir soluções satisfatórias para uma ampla variedade de problemas em vários campos diferentes.

15. Escalabilidade: Permite aplicação em versões mais complexas para um mesmo problema.

16. Ser competitivo com resultados produzidos por humanos: Produzir resultados que sejam competitivos com aqueles produzidos por programadores, engenheiros e matemáticos.

3.4 Elementos principais da Programação Genética

O paradigma da Programação Genética segue a tendência da manipulação da representação do problema em Algoritmos Genéticos, pelo aumento da complexidade das estruturas que sofrem adaptação. O paradigma é recente e foi proposto por John R. Koza [KOZ 92]. As operações são realizadas diretamente em programas de computador hierarquicamente estruturados, que podem variar em forma e tamanho. A tarefa pode ser resumida na busca pelo programa de computador mais adaptado, no espaço de todos os possíveis programas.

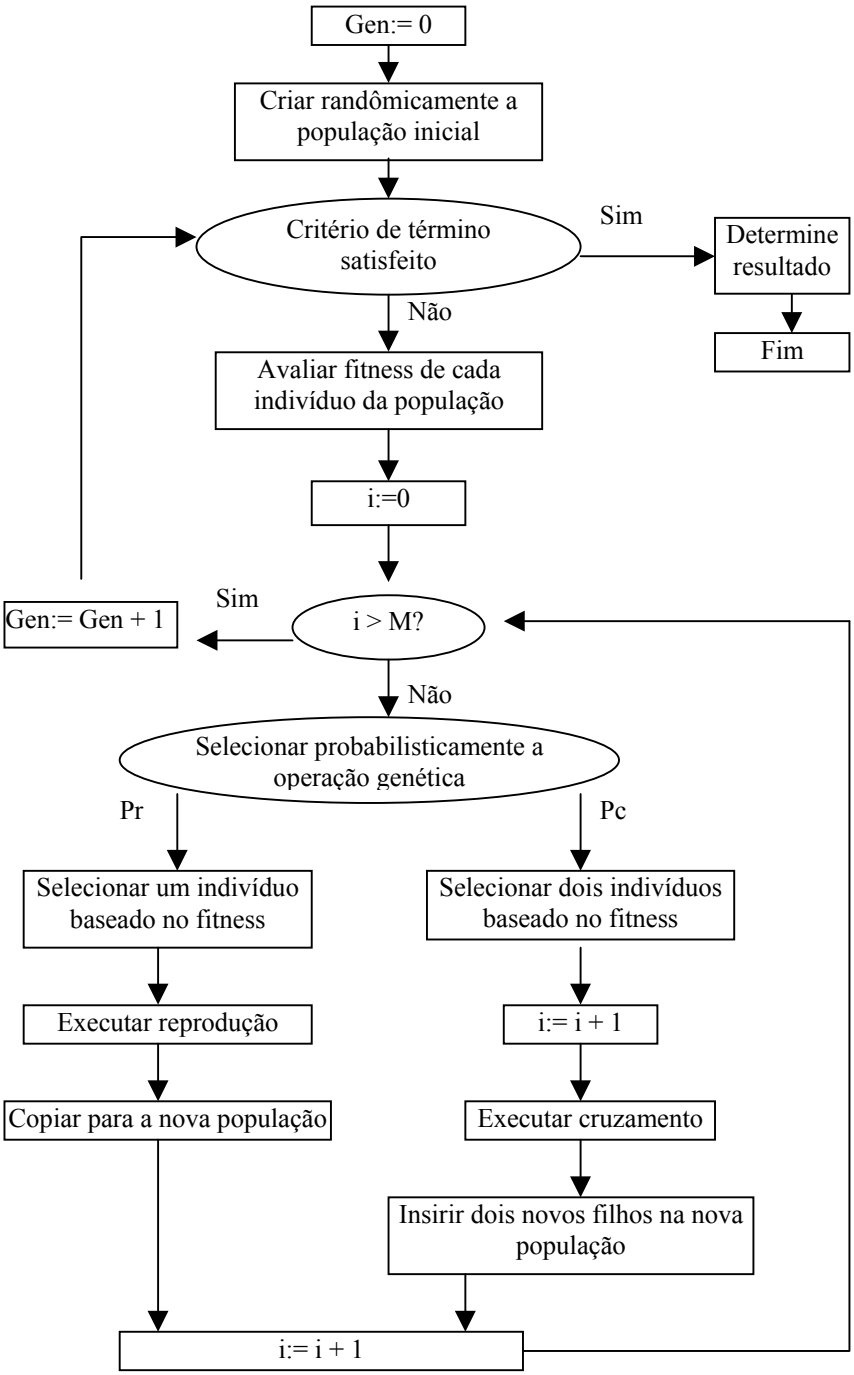


FIGURA 3.4 - Fluxograma da PG.

A fig. 3.4 ilustra os passos empregados na técnica. Na simbologia empregada, M é o tamanho da população, i o número de indivíduos a população e Pr e Pc significam probabilidades de operações de reprodução e cruzamento respectivamente. Estas probabilidades serão detalhadas posteriormente.

O processo inicia com a geração de uma população inicial de programas randomicamente criados, compostos de funções e terminais apropriados ao domínio do problema. Cada programa é uma solução em potencial e é avaliado em termos de sua adaptação ao ambiente. O princípio da reprodução e sobrevivência do melhor é então usado para criação de uma nova geração de programas a partir da atual. Programas encontrados nas gerações iniciais tendem a ser menos adaptados do que aqueles encontrados nas gerações seguintes [KOZ 92].

O estado deste processo consiste somente na atual população de indivíduos. A única informação disponível para evolução dos programas é a medida do ajuste de cada indivíduo. Não existem outras informações como por exemplo a estrutura interna, erros de representação ou efeitos que o programa possa ter tido sobre o ambiente. Tipicamente a solução para o problema é um indivíduo com melhor ajuste existente ao final da execução do algoritmo [KOZ 92]. A medida do ajuste de cada indivíduo, assim como nos Algoritmos Genéticos é chamada de fitness.

A variabilidade dinâmica dos programas de computador ao longo da busca da solução também se constitui em uma característica importante do paradigma, pois é extremamente difícil restringir o tamanho e a forma das soluções. Tais restrições poderiam diminuir a probabilidade de achar melhores soluções [KOZ 92].

Outra característica importante é que na programação genética o pré-processamento de entradas ou pós-processamento de saídas não é necessário ou exerce uma menor influência, uma vez que entradas, resultados intermediários e saídas são expressados diretamente em termos de instruções ou valores relativos ao domínio do problema [KOZ 92].

3.4.1 Estruturas que sofrem adaptação

Em qualquer sistema adaptativo, existem estruturas que são submetidas a um processo evolutivo, para que passem a funcionar melhor em seu ambiente. Em algoritmos adaptativos genéticos, cada estrutura é um ponto individual em uma população de soluções, constituindo-se assim em um ponto individual no espaço de busca do problema, ocorrendo uma busca paralela. O paradigma da Programação Genética, utiliza programas hierarquicamente estruturados que variam em tamanho e forma e são modificados durante o processo [KOZ 92].

O conjunto de possíveis estruturas na programação genética é formado pelo conjunto de possíveis composições a partir de um conjunto de funções disponíveis N_{func}, $F=\{f1, f2, f3...f_{Nfunc}\}$, exemplificadas a seguir e de um conjunto de terminais disponíveis N_{term} $T=\{t1, t2, t3...t_{Nterm}\}$. Cada função f_i requer um número $z(f_i)$ de argumentos $z(f_1)$, $z(f_2)$.... Ao definir o conjunto de funções possíveis, é necessário definir também um mapa de argumentos, que indica o número de argumentos necessários para cada uma das funções usadas para compor a estrutura [KOZ 92].

Como exemplos de funções podemos citar:

- operações aritméticas (+, -);

- funções matemáticas (Sin, Cos, Exp);
- operações booleanas (AND, OR, NOT);
- operadores lógicos (IF-THEN-ELSE);
- operadores de iteração (DO-UNTIL).
- funções do domínio da aplicação, desenvolvidas para o problema.

O conceito de terminal está relacionado à representação de programas sob a forma de árvores. Um terminal não possui filhos, isto significa que não opera sobre nenhum argumento em caso de funções ou não constitui uma operação, como no caso de constantes e variáveis. Terminais podem ser átomos constantes como letras, números ou variáveis, que podem representar dados de entrada, sensores, variáveis de estado do sistema ou ainda funções sem argumentos [KOZ 92].

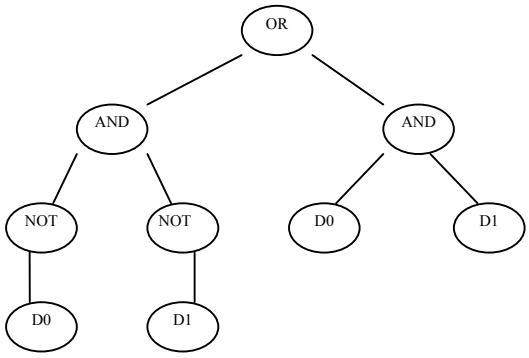


FIGURA 3.5 - Estrutura hierárquica correspondente a uma expressao-S em LISP.

Consideremos a fig. 3.5, que exemplifica um programa composto a partir de um conjunto de funções F={AND, OR, NOT} e do conjunto de terminais T={D0,D1}. O mapa de argumentos das função é Z={2, 2, 1}. A figura mostra a estrutura de um programa (OR (AND (NOT D0) (NOT D1))(AND D0 D1)), que foi gerado em uma população inicial, para resolução do problema de função de paridade para dois argumentos. Os pontos externos da árvore são terminais, sobre os quais atuam as funções que se encontram nos ramos internos da árvore [KOZ 92].

Para um funcionamento correto, dois requerimentos devem ser satisfeitos na escolha do conjunto de funções e terminais aplicáveis ao problema em questão: clausura e suficiência.

A propriedade de clausura, diz respeito ao fato de todos os indivíduos em uma população serem compostos de funções e terminais que executem corretamente, sem erros. Qualquer combinação entre terminais e funções, deve resultar em um programa válido. Um caso clássico de falta de clausura, pode ser considerado a combinação de uma função de divisão com a constante zero. Existe a possibilidade de que esta constante seja escolhida como segundo argumento da função em alguma solução. Tal combinação resulta em um erro de divisão por zero. Uma alternativa neste caso é a adoção de uma função de divisão protegida. Caso não seja possível satisfazer a propriedade, alguma alternativa deve existir, como alguma penalidade para o fitness ou simplesmente o descarte dos indivíduos cujo resultado não pode ser avaliado [KOZ 92].

Para satisfazer a propriedade de suficiência, o conjunto de funções e terminais escolhido deve ser grande e poderoso o suficiente para resolver o problema. Alguma combinação possível de funções e terminais, deve constituir-se em uma solução correta. Em muitos domínios a escolha do conjunto de funções e terminais pode ser uma tarefa difícil, dependente de processo criterioso de análise e modelagem.

3.4.2 Fitness

Cada indivíduo tem associado a si uma medida numérica, que é o resultado da interação dele com o ambiente; é em resumo uma medida do grau de adaptação deste indivíduo. Esta medida é que dirige diretamente o processo evolucionário, fazendo com que, quanto mais adaptado for um indivíduo maior a sua chance de permanecer ou ter suas características propagadas para as próximas gerações. A definição do fitness está ligada ao domínio do problema a ser resolvido [KOZ 92].

Podemos considerar quatro tipos básicos de fitness na programação genética: Fitness Básico, Fitness Padronizado, fitness de ajuste e fitness normalizado ou proporcional [KOZ 92].

O fitness básico é definido diretamente em função do domínio do problema, e na maioria das aplicações de acordo com um conjunto de casos de teste. Casos de teste são freqüentemente uma amostra do espaço de busca. No exemplo da função de paridade, o fitness básico poderia ser definido como o número de casos de fitness onde o valor retornado pelo programa é igual ao valor correto esperado [KOZ 98].

O fitness padronizado é relacionado com o fitness básico, e modifica este de forma que um valor mais baixo seja sempre melhor. Como exemplo para a função de paridade o fitness padronizado poderia ser definido como a soma a partir de N casos de teste da distância Hamming (erro) entre o valor retornado pelo programa e o valor correto esperado. O fitness padronizado é normalmente àquele que guia o processo de evolução. Indivíduos com menores valores estão mais próximos a resposta exata [KOZ 98].

O fitness de ajuste é usado somente em casos em que uma resposta exata é exigida para um problema, e o fitness proporcional é usado normalmente para definição da probabilidade de um indivíduo ser selecionado para a próxima geração ou para alguma operação genética [KOZ 98].

A avaliação do fitness é a atividade mais fundamental para uma boa evolução. Uma má função de avaliação, implica em processos sem boas soluções ou com convergências prematuras para respostas pouco adaptadas. A avaliação ocorre em uma amostra que represente o problema; esta amostra é composta de casos de fitness. Cada caso ou subconjunto de casos pode representar uma ou mais particularidades do problema. O grau de adaptação final de um indivíduo é calculado sobre todos os casos, o que torna a escolha dos mesmos, uma tarefa fundamental na modelagem do problema. Uma análise profunda do processo de avaliação de fitness pode ser encontrado em [BAN 98].

3.4.3 Operadores genéticos

Existem duas operações primárias para modificação de estruturas: Reprodução de um único indivíduo e cruzamento (*crossover*), que se constitui em uma recombinação

sexuada de dois indivíduos [KOZ 92]. Outra operação utilizada em menor escala é a mutação.

A reprodução é executada em dois passos: seleção de um indivíduo e cópia deste para a nova geração. Cada pai gera apenas um filho; o mecanismo da reprodução assexuada é a própria representação da teoria da evolução, da sobrevivência do mais adaptado. Indivíduos com alto grau de adaptação em geral são selecionados mais de uma vez para se reproduzirem.

A operação de cruzamento ocorre da seguinte forma: são escolhidos dois indivíduos para uma operação sexuada; em cada um dos pais é escolhido de forma randômica um ponto de cruzamento; a subárvore existente a partir do ponto escolhido no primeiro pai é inserida no ponto de cruzamento do segundo pai e vice-versa. As fig. 3.6 e 3.7 ilustram esta operação.

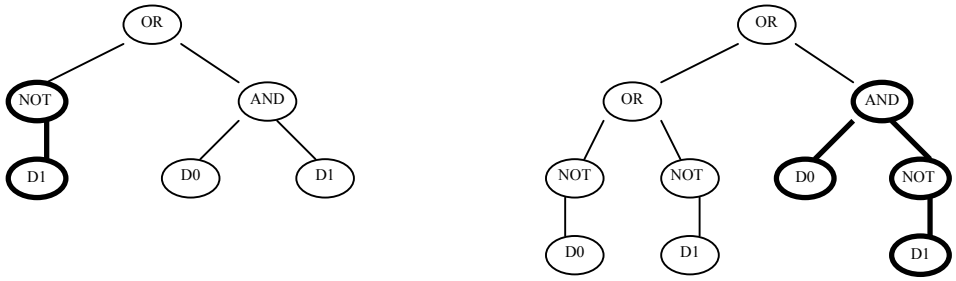


FIGURA 3.6 - Pais escolhidos para cruzamento e seus respectivos pontos de corte.

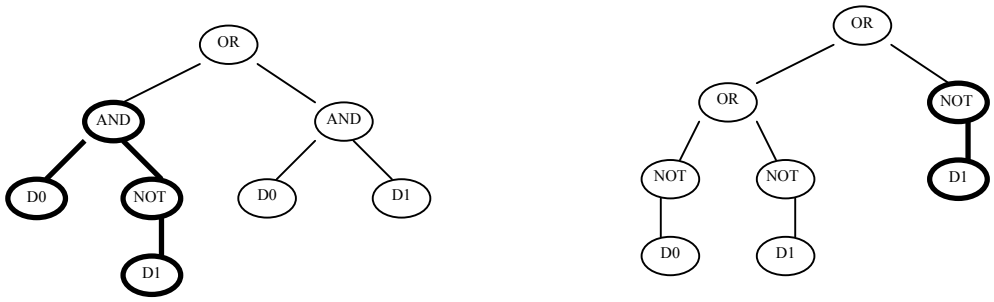


FIGURA 3.7 - Filhos gerados após a operação de cruzamento

A operação de cruzamento cria variações na população pela produção de filhos compostos de partes de cada um dos pais. A propriedade de clausura é mantida a partir do momento que somente funções e terminais disponíveis nos conjuntos possíveis continuam sendo usadas. Se um ramo raiz é escolhido, um indivíduo inteiro é inserido como subárvore de outro e caso dois ramos raiz sejam escolhidos, serão gerados dois filhos iguais a seus pais.

Operações de cruzamento, modificam a forma e tamanho dos programas, pois ao contrário do algoritmo genético convencional, onde um ponto de cruzamento é igual em dois pais, o que garante a manutenção do tamanho da string, na programação genética o ponto selecionado é em geral diferente para cada um dos pais.

A mutação tradicional consiste em selecionar um indivíduo e um ponto aleatório e substituir a subárvore a partir deste ponto por outra criada randomicamente. Se uma

operação de cruzamento ocorrer em pontos onde existam terminais, o efeito é o mesmo de uma mutação de um ponto.

3.4.4 Critérios e parâmetros de controle

A execução da técnica é dependente de um conjunto de parâmetros cujos valores são normalmente escolhidos no início do processo. A escolha de diferentes valores para cada um dos parâmetros afeta diretamente a evolução e os recursos consumidos pelo sistema [KOZ 92].

Um dos critérios importantes em uma técnica adaptativa é como saber e obter ao final da execução o resultado do processamento. Na programação genética, o resultado final do processo é um único indivíduo mais adaptado ao ambiente, ou seja o vencedor ao final é considerado a melhor solução possível para o problema. Em muitos casos se não houver uma estratégia elitista, não é possível garantir que o melhor indivíduo estará presente na geração final. Uma estratégia elitista privilegia a perpetuação do melhor indivíduo nas próximas gerações, altas taxas de reprodução na maioria dos casos, garantem esta característica [KOZ 92].

Os parâmetros mais utilizados para controlar o processo são: tamanho da população (M) e o número de gerações (N). O tamanho da população influencia diretamente no rendimento e no esforço computacional necessário para produzir resultados, e o número de gerações serve como um critério de término, caso nenhum outro critério seja estabelecido. Maiores populações também podem significar um maior espaço de busca e portanto maior probabilidade de encontrar boas soluções [KOZ 92].

O percentual de execução de cruzamento (Pc), percentual de reprodução baseada em fitness (Pr) e percentual de mutação (Pm) afetam a composição da próxima geração. Maiores taxas de cruzamento e mutação definem estratégias menos elitistas, e maior variabilidade das soluções da próxima geração. O percentual de cruzamento pode ser dividido ainda em percentual de cruzamento, em pontos de função e em qualquer ponto. Estes últimos implicam em mudanças mais ou menos radicais no tamanho das soluções. A definição a priori de pontos de corte e taxas de cruzamento, reprodução e mutação é uma tarefa complexa. Na maior parte dos casos, execuções iniciais poderão indicar quais as melhores combinações para um problema específico [KOZ 92].

Outros parâmetros que influenciam a variação do tamanho das soluções são a profundidade máxima tolerada na geração inicial e nas gerações seguintes. Limitar a profundidade significa que a árvore representativa de um programa não poderá ter em sua ramificação mais profunda uma quantidade superior a um determinado número de nodos fixado como limite. Árvores com maior profundidade significam soluções com um maior número de pontos. Em muitas aplicações um aumento do número de pontos é importante para a solução do problema, porém um acréscimo desmedido pode consumir de forma ilimitada os recursos e prejudicar a convergência [KOZ 92].

A criação da geração inicial está relacionada à escolha de uma dentre três alternativas possíveis: Crescente, Subida Parcial ou Completa. A estratégia Crescente, produz indivíduos com tamanhos irregulares. A Completa, cria soluções nas quais no mínimo um dos nodos atinge a profundidade máxima. Na Subida Parcial, são criados indivíduos com profundidades distribuídas entre tamanhos mínimos e máximos. Como exemplo podemos supor que a profundidade máxima é 6, neste caso ocorre uma distribuição igual de indivíduos com profundidades 2, 3, 4, 5 e 6. Esta última alternativa

permite uma boa variabilidade na população sem soluções de tamanhos exagerados e é portanto a escolha mais comum [KOZ 92].

Outra escolha a ser realizada no início do processo diz respeito ao método utilizado para selecionar indivíduos para operações genéticas. Existem diversas variações possíveis, entretanto as três alternativas tradicionais da técnica são: Seleção proporcional ao fitness, torneio e proporcional ao fitness com seleção excedente. A primeira é a alternativa empregada na maioria das execuções deste trabalho e é equivalente à "roleta", comum nos Algoritmos Genéticos; soluções são escolhidas usando como critério o grau de adaptação, quanto mais adaptada maior a probabilidade de perpetuar seu material genético para as próximas gerações. A seleção por torneio envolve a escolha de um determinado número de indivíduos (tamanho do torneio) de forma aleatória, dentre estes os melhores são submetidos à operação específica. A seleção excedente é tipicamente usada com grandes populações e permite gerar através das operações genéticas uma população maior do que o tamanho original, a seguir o número de indivíduos é reduzido novamente ao tamanho da população pelo descarte dos indivíduos menos adaptados.

3.4.5 Escolha da linguagem de programação

Após definir os principais conceitos relativos ao paradigma da Programação Genética, e de utilizar como base para exemplo representações usando a linguagem LISP (LISt Processing), torna-se necessário definir de forma mais prática o motivo da escolha de LISP para implementação da maioria dos exemplos encontrados em trabalhos sobre o assunto, e também se é possível a utilização de outras linguagens na implementação da técnica.

Qualquer programa de computador escrito em alguma linguagem (C, PASCAL...) pode ser visto como uma seqüência de aplicações de funções ou operações, a uma seqüência de argumentos ou valores. Compiladores usam este fato internamente para representar programas como uma árvore, posteriormente convertendo-a em código de *assembly* elementar. O paradigma é mais facilmente compreensível, se pensarmos em uma linguagem de programação que transparentemente trata um programa de computador como seqüências de aplicações de funções a argumentos.

Além disso, uma linguagem de programação que permite que qualquer estrutura seja direta e imediatamente avaliada como um programa, é extremamente benéfica, pois durante todo o processo programas são criados, modificados em sua forma e tamanho e são executados para avaliação do fitness. Esta funcionalidade, é diretamente obtida em LISP usando a função EVAL. Expressões LISP são escritas e avaliadas de uma forma pré-fixada (* 2 3), ao contrário de outras linguagens de programação tradicionais que utilizam a forma (3 * 2). Esta abordagem facilita a manipulação hierárquica de programas.

Embora a maior parte dos exemplos e discussões existentes baseiam-se na linguagem LISP, muitos autores tem utilizado C ou mesmo código de máquina em aplicações mais complexas ou direcionadas a um domínio específico. Um exemplo pode ser encontrado em [BANZ 95], onde em uma aplicação para controle em tempo real de movimento de um robô Kephera programas foram representados na forma de código de máquina binário, devido a necessidade de maior velocidade. Em [FRA 94] é possível encontrar uma implementação completa da técnica utilizando linguagem C.

3.5 Definição automática de funções

A partir da exposição dos conceitos gerais da programação genética nas seções anteriores, a partir desta será introduzido um conceito que estende a capacidade da programação genética, a descoberta automática de funções, que é fortemente baseada em conceitos de resolução hierárquica de problemas.

A descoberta automática de funções (*ADF - Automatic Defined Functions*) foi introduzida por John R. Koza em [KOZ 98] e tem por objetivo permitir ao paradigma da programação genética resolver problemas cada vez mais complexos através da descoberta de sub-rotinas ou funções que podem ser reutilizadas. Estas sub-rotinas permitem explorar regularidades existentes em problemas de diversas formas.

3.5.1 Resolução hierárquica de problemas

A resolução de problemas de forma hierárquica em um modelo top-down pode ser decomposta de três etapas distintas: i) Identificação de como um problema geral pode ser decomposto em subproblemas; ii) resolução de cada um dos subproblemas propostos; iii) resolução do problema geral através da integração das soluções de cada um dos subproblemas agora disponíveis [KOZ 98].

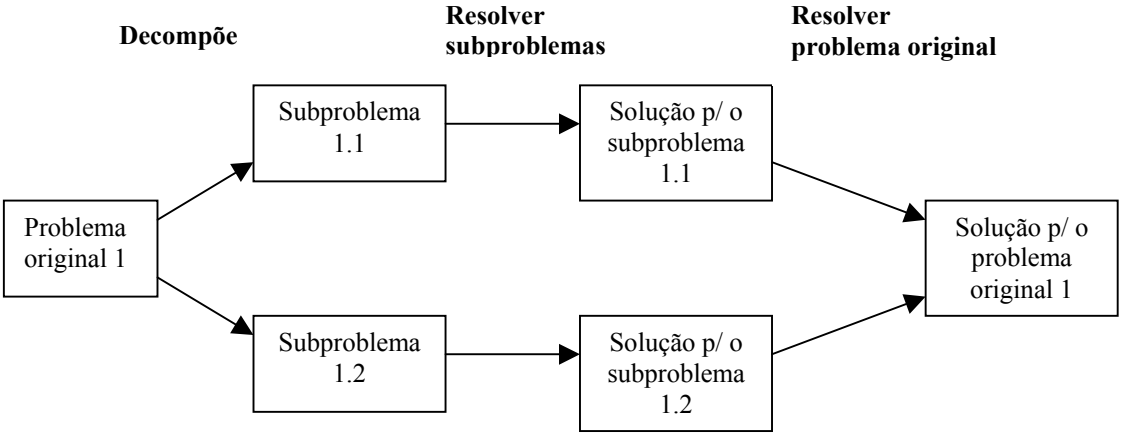


FIGURA 3.8 - Passos para resolução hierárquica de funções

Tradicionalmente, a decomposição hierárquica pode ser uma forma de reduzir o esforço computacional necessário para resolver o problema. Podemos ainda citar outras vantagens que podem emergir a partir de uma boa decomposição hierárquica de um problema: Aplicação recursiva, reuso idêntico, reuso parametrizado, generalização e abstração.

A aplicação recursiva ou reuso idêntico pode ocorrer no momento em que é possível utilizar parte da decomposição de um subproblema novamente na decomposição de outro subproblema. Neste caso, a solução invoca o mesmo passo que compõe parte da solução de algum subproblema anterior que trabalhe nas mesmas variáveis ou dados e execute as mesmas instruções; a vantagem é que não é necessário reescrever este subproblema ou procedimento.

Em um reuso parametrizado ou na generalização de uma solução, também é reutilizada uma etapa que compõe um subproblema, porém ao ser utilizado um mesmo procedimento para resolver o problema atual, serão usados diferentes dados de entrada para este procedimento. Em um exemplo, poderíamos considerar um problema da

derivada de uma soma $g(x) = x^3 + x^4$, que pode gerar a decomposição em dois subproblemas ($3x^2$) e ($4x^3$). Podemos desenvolver um procedimento geral para resolução de x^m e invocar este mecanismo em diferentes momentos com valores diferentes para m. O mecanismo, explora similaridades entre subproblemas, porém é necessário primeiro identificar estas similaridades para resolução por um mecanismo geral. Programadores humanos, utilizam extensivamente rotinas genéricas para resolução de classes iguais de problemas [KOZ 98].

A abstração, diz respeito a capacidade de identificar determinadas instruções ou variáveis que são irrelevantes para solução do problema, em determinado momento. Como exemplo, podemos considerar uma rotina que receba um conjunto de parâmetros e eventualmente desconsidere um ou mais destes parâmetros no momento da resolução do problema. Em uma chamada a rotinas genéricas, somente são enviadas as variáveis necessárias para resolução do subproblema, e não todas as variáveis que definem o estado do problema.

De acordo com as descrições acima, podemos resumir em cinco, os benefícios que reduzem o esforço computacional necessário a partir da decomposição hierárquica de problemas [KOZ 98]:

1. Eficiência associada ao processo de decomposição hierárquica;
2. Eficiência obtida pela aplicação recursiva do processo de decomposição;
3. Reuso idêntico de soluções de problemas já resolvidos;
4. Reuso parametrizado, que reutiliza as mesmas soluções para subproblemas diferentes;
5. Abstração, onde variáveis irrelevantes ampliam a capacidade e aplicabilidade de soluções para subproblemas.

3.6 O problema da definição automática de função

Tendo em mente os benefícios da resolução hierárquica de problemas, algumas questões práticas são levantadas: Como é possível implementar o processo de forma automática e independente do domínio? Como fazer uma decomposição de um problema em subproblemas? Como identificar quais são os subproblemas e como resolvê-los? Após terem sido resolvidos os subproblemas, como montar a solução para o problema geral a partir destas soluções? [KOZ 98]

Para resolver as questões acima, a primeira idéia fundamental é a de introduzir sub-rotinas no processo de descoberta de programas; desta forma a estrutura de uma solução passará a ser composta de um programa principal e uma ou mais sub-rotinas. Anteriormente, ao definir os conceitos relativos a programação genética, uma das definições necessárias referia-se ao conjunto de funções utilizadas. As funções neste caso eram em sua maioria funções disponíveis na linguagem, e funções do domínio da aplicação criadas antes do processo por algum programador humano.

O grande objetivo da definição automática de funções é evoluir simultaneamente funções ou sub-rotinas e um programa principal durante o mesmo processo de execução. As funções são automaticamente definidas, porque assim como programas chamadores, serão criadas aleatoriamente a partir do conjunto básico de funções e

terminais disponíveis e serão submetidas ao mesmo processo de evolução natural, daí origina-se o conceito de ADF. Durante o processo evolutivo, são criados programas contendo um ou mais ramos de definição de função e um programa principal, chamado de ramo de produção de resultados. O número de vezes que o ramo de produção de resultados chama algum dos ramos de definição de função para cada indivíduo em particular não é predeterminado e é resultante do efeito evolucionário [ROS 96].

A evolução concorrente de unidades funcionais e programas chamadores capacita a programação genética a realizar inteiramente o processo de resolução hierárquica de funções descrito anteriormente [KOZ 98].

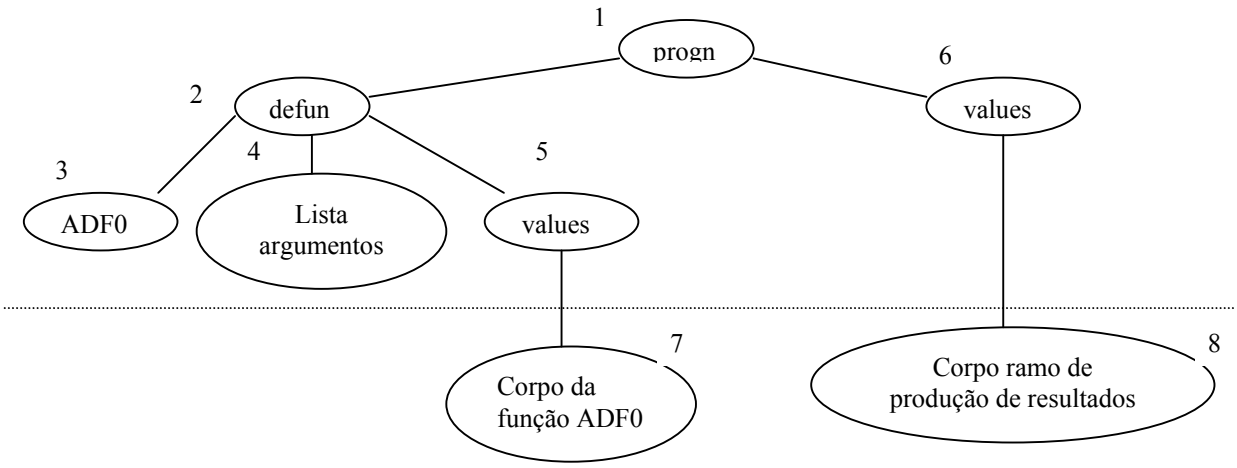


FIGURA 3.9- Exemplo de um programa geral com um ramo de produção de resultado e um ramo de definição de função

O programa geral mostrado pela fig.3.9 tem oito diferentes pontos. Os primeiros seis são chamados invariantes, operações de cruzamento ou mutação não são realizadas sobre eles, os mesmos permanecem fixos na estrutura de qualquer indivíduo. Pontos invariantes aparecem acima da linha horizontal pontilhada. Os dois últimos representam conjuntos de instruções variantes e constituem o corpo das funções que emergem a partir do processo evolucionário [KOZ 98].

Definição dos pontos invariantes:

- (1) A raiz do programa constitui-se da função conectiva *progn*, que indica a execução seqüencial dos ramos abaixo.
- (2) A instrução *defun* é a instrução LISP para iniciar a definição de uma função.
- (3) O nome ADF0 é um nome genérico atribuído a função que será definida dinamicamente. Neste caso é possível usar outro nome genérico sem modificar a estrutura.
- (4) Lista de argumentos que serão recebidos pela função.
- (5) A função *values* do ramo de definição de função identificando o valor retornado pelo ramo de definição de função.
- (6) A função *values* do ramo de produção de resultados, identificando o valor retornado pelo ramo de produção de resultado.

- Os pontos (7) e (8) são o conjunto de instruções criadas aleatoriamente e que são submetidas a evolução durante a execução.

Estes dois últimos tipos de ponto constituem-se efetivamente no resultado da extensão proposta . Cada indivíduo da população tem seu próprio ramo de definição de funções e de produção de resultados. De acordo com [KOZ 98], funções definidas automaticamente, podem:

- Executar cálculos similares aos quais programadores humanos fariam;
- Executar cálculos diferentemente aos criados por programadores humanos;
- Redundantemente definir uma função que é equivalente a uma função primitiva disponível no conjunto de funções para o problema;
- Ignorar algumas das variáveis no processo;
- Ser totalmente ignorados pelos ramos de produção de resultados;
- Definir um valor constante, independente de todas as variáveis;
- Retornar valores idênticos às variáveis, desta forma definindo redundantemente um terminal do problema;
- Chamar redundantemente outra função definida.

3.6.1 Criação de uma população inicial

A população inicial de indivíduos, randomicamente gerada precisa ser criada com cada indivíduo tendo a estrutura definida na fig. 3.9, podendo apenas existir mais de um ramo de definição de função. Um ramo de definição de função é criado a partir do conjunto de funções F_{adf} e do conjunto de terminais T_{adf}, disponíveis para resolução da função. O ramo de produção de resultados é uma composição randômica de funções a partir do conjunto de funções F_{rpb} e do conjunto de terminais T_{rpb}, disponíveis para solução geral do problema [KOZ 98].

É possível definir diferentes conjuntos de funções e terminais para cada um dos ramos, o que ocorre pelo fato do ramo de produção de resultados ter disponível além das funções tradicionais, a função geral ADF0 para uso [KOZ 98].

3.6.2 Preservação de estruturas em cruzamentos

A operação de cruzamento conforme detalhada nas características gerais da programação genética consiste em selecionar dois pais e escolher de forma randômica um ponto em cada um dos pais para corte. Parte do primeiro pai é inserido no segundo e vice-versa, desta forma dois novos filhos são criados. Com o uso da definição automática de funções é necessário que em operações de cruzamento a estrutura sintática da construção dos filhos seja mantida, para tanto alguns pontos não podem ser escolhidos para cruzamento [KOZ 98].

Como podemos verificar na fig. 3.9, os pontos numerados de 1 a 6 são pontos invariantes e nunca podem ser escolhidos como ponto de corte em operações de cruzamento; tal preservação garante que todos os indivíduos da população terão sempre um ramo de produção de resultados, e no mínimo um ramo de definição de função. Caso existam mais ramos de definição de função, haverá um número maior de pontos invariantes [KOZ 98].

A idéia básica do cruzamento com preservação da estrutura, é baseada na definição de um tipo para cada ponto em um indivíduo. Todas os pontos que compõe um ramo de definição de função possuem o número 7 em nosso exemplo e os pontos do ramo de produção de resultados tem atribuídos a si o número 8. Em uma operação de cruzamento, um ponto qualquer destes dois ramos é escolhido de forma aleatória. A partir desta escolha, o ponto a ser escolhido no segundo pai deve ser qualquer ponto que tenha o mesmo tipo daquele escolhido no primeiro pai, o que garante a produção de filhos com estrutura sintática válida [KOZ 98].

4 Abordagem do sistema: Arquitetura proposta

Para permitir a validação do paradigma da Programação Genética ao problema, foi necessário desenvolver um sistema baseado nos princípios básicos desta técnica. A proposta do trabalho não é desenvolver um aplicativo para realizar a síntese de forma completa e sim criar um sistema para aplicação da técnica ao problema descrito e em trabalhos futuros a outros.

O desenvolvimento utilizou a linguagem LISP (Allegro CL 3.0.2) . O ambiente de desenvolvimento possui diversos componentes para desenvolvimento de interfaces gráficas, além de ser uma linguagem orientada a objetos baseada na linguagem Common Lisp [SLA 98]. Outra escolha possível seria a utilização do próprio sistema Festival e a linguagem Scheme, que constitui um dialeto da linguagem LISP e está presente neste ambiente.

A justificativa para não utilização da ferramenta Festival baseiam-se em algumas características da linguagem Scheme:

- Falta de recursos para programação orientada a objeto. O núcleo da implementação da técnica utiliza classes e objetos e a sua implementação no referido ambiente demandaria em profundas mudanças na implementação original.
- Poucos recursos para criação de interfaces. Para uma boa avaliação dos resultados e da evolução e ainda para que a definição dos parâmetros de cada execução fosse flexível, o uso de recursos gráficos é desejável.
- Algumas funções originais da linguagem não estão presentes neste dialeto, ou apresentam uma sintaxe diferente.

Embora a aplicação da técnica tenha sido realizada em um sistema independente da ferramenta utilizada no projeto Spoltech, trabalhos futuros poderão permitir a utilização da técnica ou dos resultados no processo de síntese realizado no Festival. O aproveitamento destes resultados ou a adaptação do sistema desenvolvido no trabalho para o sistema Festival, dependerá de uma análise sobre qual a melhor alternativa ou como regras criadas como programas podem ser empregadas para conversão. Conforme citado anteriormente no item 2.6.2, as regras possuem formato diferente dos resultados que serão descritos posteriormente neste trabalho. Estas análises e adaptações não são objeto de estudo neste trabalho, a aplicação tem como objetivo investigar a possibilidade da utilização da técnica no processo de síntese.

Após a escolha da linguagem, o passo seguinte, foi definir as características do sistema a ser implementado. Os requisitos básicos, concentravam-se nas funcionalidades tradicionais da programação genética, na criação da população inicial e de todo o processo evolutivo. Outro requisito da implementação é que houvesse um alto grau de modularidade, para que fosse possível utilizá-la em trabalhos futuros, onde a técnica poderia ser aplicada a outros problemas.

Baseada nestes requisitos, foi proposta uma arquitetura na qual as funcionalidades da Programação Genética são implementadas em um núcleo principal, separado das interfaces de entrada e saída, que compõe os demais módulos.

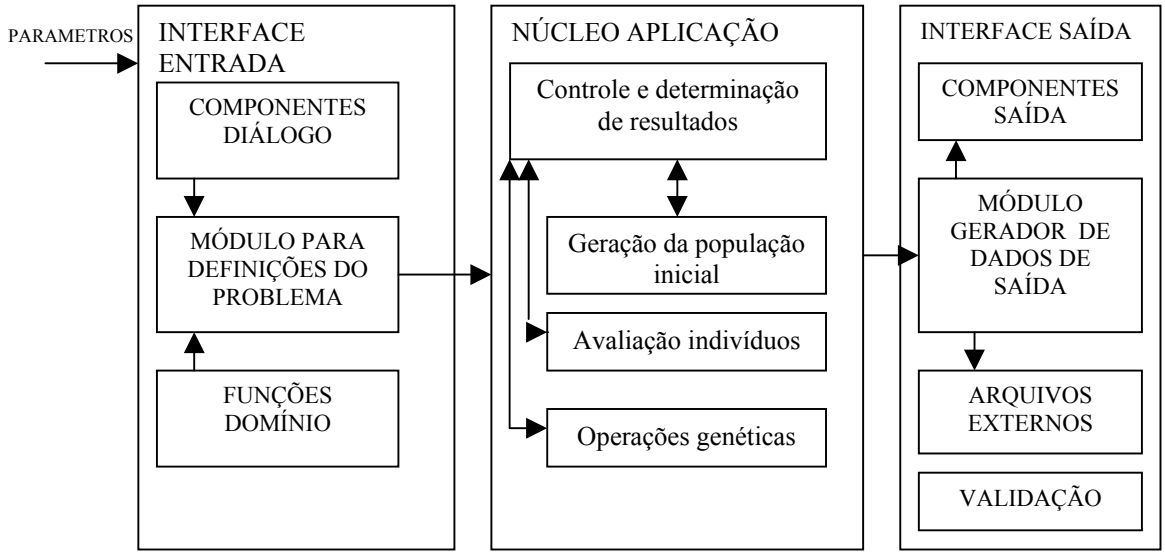


FIGURA 4.1 - Arquitetura do sistema

O núcleo da aplicação é baseado na implementação desenvolvida por Koza [KOZ 98], que foi utilizada para validação da técnica com problemas clássicos como indução de funções matemáticas, descoberta de funções booleanas entre outros. O módulo responsável pela execução e controle, recebe um conjunto de parâmetros, incluindo nome das funções de domínio. As funções definidas como responsáveis pela formação dos parâmetros básicos, conjunto de terminais, funções e de fitness são executadas retornando ao módulo de controle, todas as características do problema. Seguindo o princípio da modularidade, todas as funcionalidades relativas ao problema, estão isoladas do núcleo principal. Após a composição de todas as características do problema, o processo evolutivo é iniciado, e somente é abortado pelo módulo de controle quando a condição de término do problema é satisfeita.

O núcleo do sistema, possui ainda os módulos de geração da população inicial, operações genéticas e avaliação de indivíduos, que são executados pelo módulo de controle e recebem os argumentos necessários para correta execução O módulo de avaliação, recebe como argumento o nome da função para cálculo de fitness de um indivíduo e executa a mesma para cada indivíduo da população. Após a formação e avaliação de uma geração inteira, são chamadas as funções dos componentes do módulo responsável pela geração das informações para avaliação e acompanhamento do processo.

Os indivíduos são armazenados em uma matriz composta de objetos da classe *individual*, que possui a seguinte estrutura:

```
(defstruct individual
   program
  (standardized-fitness 0)
  (adjusted-fitness 0)
  (normalized-fitness 0)
  (hits 0)
  (complex-estrut 0))
```

Cada indivíduo possui 5 propriedades que armazenam além do código do programa, os vários fitness e a complexidade estrutural do mesmo. Quando um novo

indivíduo é criado, a classe é instanciada, as propriedades recebem os valores computados na avaliação e *program* recebe o conjunto de instruções.

Para a execução utilizando definição automática de funções, é definida outra estrutura adicional como segue:

```
(defstruct
 (adf-program
  (:print-function
   (lambda (instance stream depth)
    (declare (ignore depth))
    (format stream
      "(progn (defun ADF0 (ARG0 ARG1 ARG2)~
     ~%        (values ~S))~
     ~%      (defun ADF1 (ARG0 ARG1 ARG2)~
     ~%        (values ~S))~
     ~%        (values ~S))"
      (adf-program-adf0 instance)
      (adf-program-adf1 instance)
      (adf-program-rpb instance))
    )))
 adf0
 adf1
 rpb)
```

A estrutura acima, inclui uma função para retornar o indivíduo criado de acordo com a estrutura padrão, que inclui neste exemplo, duas funções automaticamente definidas. A utilização de mais funções implicaria na modificação da classe da qual se originam os indivíduos.

A atividade de desenvolvimento mais complexa, concentrou-se na implementação dos módulos relativos ao problema e na programação de interfaces de entrada e saída. É preciso salientar que módulos de interface são independentes do problema, conforme descrito nos requisitos do sistema.

Módulos relativos ao problema são responsáveis pela implementação de funções de domínio incluídas no conjunto de funções disponíveis para a execução, definição do conjunto de funções e terminais e finalmente pela definição e avaliação dos casos de *fitness*. A fig. 4.2 ilustra a estrutura do módulo que recebe dos componentes da interface as características do problema e as disponibiliza para o núcleo da aplicação.

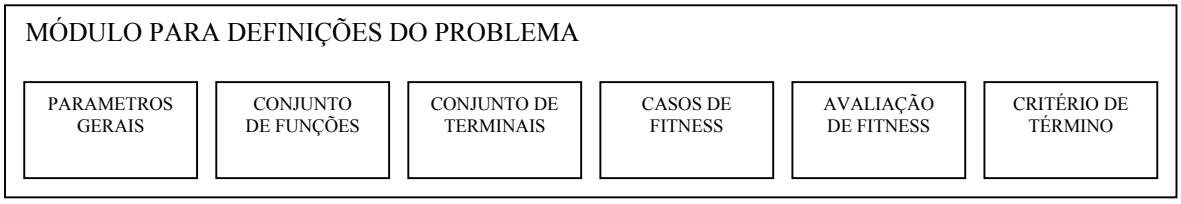


Figura 4.2 – Composição do módulo para definições do problema

Os parâmetros e especificações para cada execução são incluídos utilizando-se uma interface gráfica dividida em duas partes: Parâmetros gerais e especificação do problema. Na primeira, são incluídos os parâmetros básicos: número de gerações,

tamanho da população, profundidades máximas dos indivíduos, percentuais das operações de reprodução, cruzamento e mutação, tamanho do torneio e número de casos de fitness. O método para geração da população inicial, bem como o método de seleção são escolhidos a partir das alternativas descritas anteriormente, para cada um destes parâmetros. A interface foi desenvolvida com o intuito de escolher rapidamente entre as alternativas, reduzindo assim também a possibilidade de que seja informada uma opção incorreta. A fig. 4.3 mostra a interface e os parâmetros para uma execução.

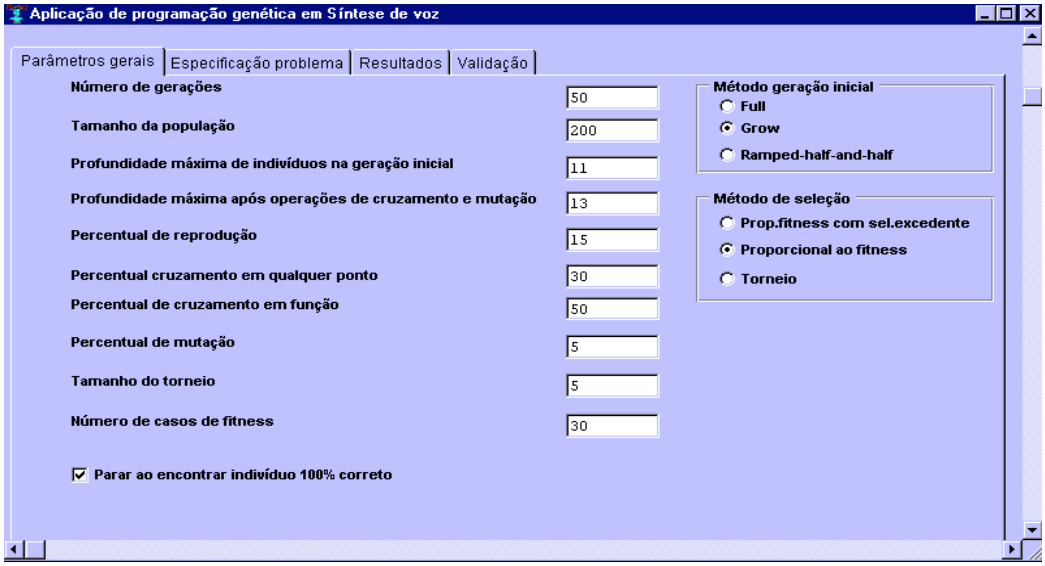


FIGURA 4.3 - Interface para definição de parâmetros gerais

A interface para definições relativas ao domínio do problema inclui controles para determinação do conjunto de terminais, funções, número de argumentos das funções e dos casos de fitness que incluem palavras e respectivos fonemas que constituem a resposta correta para cada caso. O número de funções automáticas indica se a estratégia de definição automática de funções será ou não utilizada na execução. Se o número de funções automáticas é igual a zero, a execução não emprega a estratégia, caso contrário são indicadas quantas funções automáticas as soluções irão implementar e ainda o conjunto de terminais e funções para cada um dos ramos de definição de função.

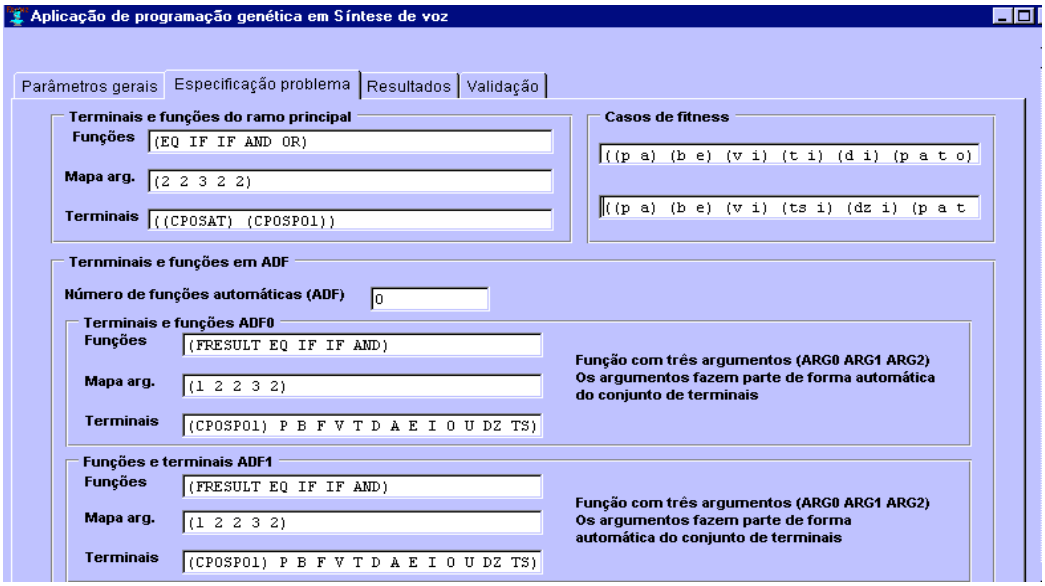


FIGURA 4.4 - Interface para definições relativos ao domínio do problema.

A avaliação e acompanhamento dos resultados, ocorre através de uma interface de saída composta por arquivos externos em formato texto e componentes gráficos em tela, atualizados durante a execução. O módulo gerador de dados de saída, descrito na arquitetura é responsável por enviar informações, tanto para a interface de saída quanto para arquivos externos.

Informações enviadas à arquivos externos possuem naturalmente um nível de detalhe maior por terem estes a finalidade de permitir a documentação dos resultados a posteriori, ao passo que a interface gráfica é usada para o acompanhamento do processo a cada geração, através de informações como geração atual, melhor fitness da geração, fitness, complexidade média da população e dados sobre o melhor indivíduo até a geração atual. Um acompanhamento do processo permite verificar se a evolução está realmente tornando as soluções mais adaptadas, ou se está ocorrendo a convergência para um mínimo local; fenômeno comum em algoritmos baseados em abordagem heurística. A fig. 4.5 ilustra o acompanhamento de uma execução através da interface.

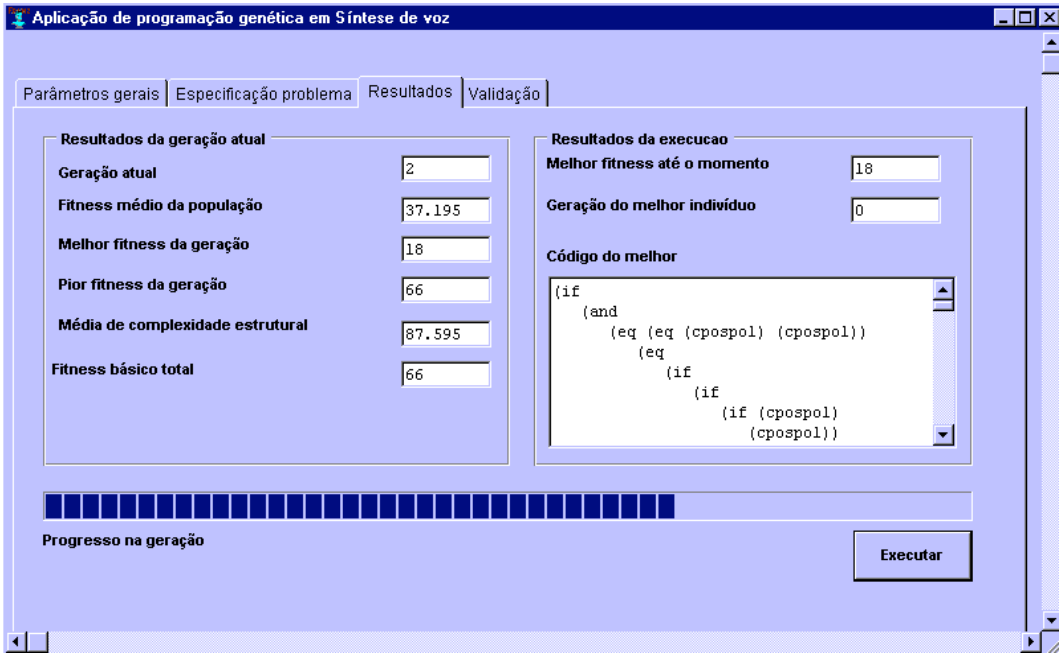


FIGURA 4.5 - Interface de saída para acompanhamento do problema.

Por fim, a arquitetura apresenta um módulo de validação de soluções produzidas durante uma execução. O módulo de validação não possui uma interação direta com o processo evolutivo, mas é de vital importância ao final de uma execução para verificar os resultados para cada caso, bem como se a solução produzida tem um bom desempenho em outras palavras, além daquelas utilizadas como casos de fitness.

A interface de validação inclui controles para informação de palavras usadas na validação juntamente com seus respectivos fonemas, do indivíduo a ser validado e mostra resultados produzidos por este indivíduo, que incluem o fitness e a resposta deste para cada palavra da lista de casos de validação.

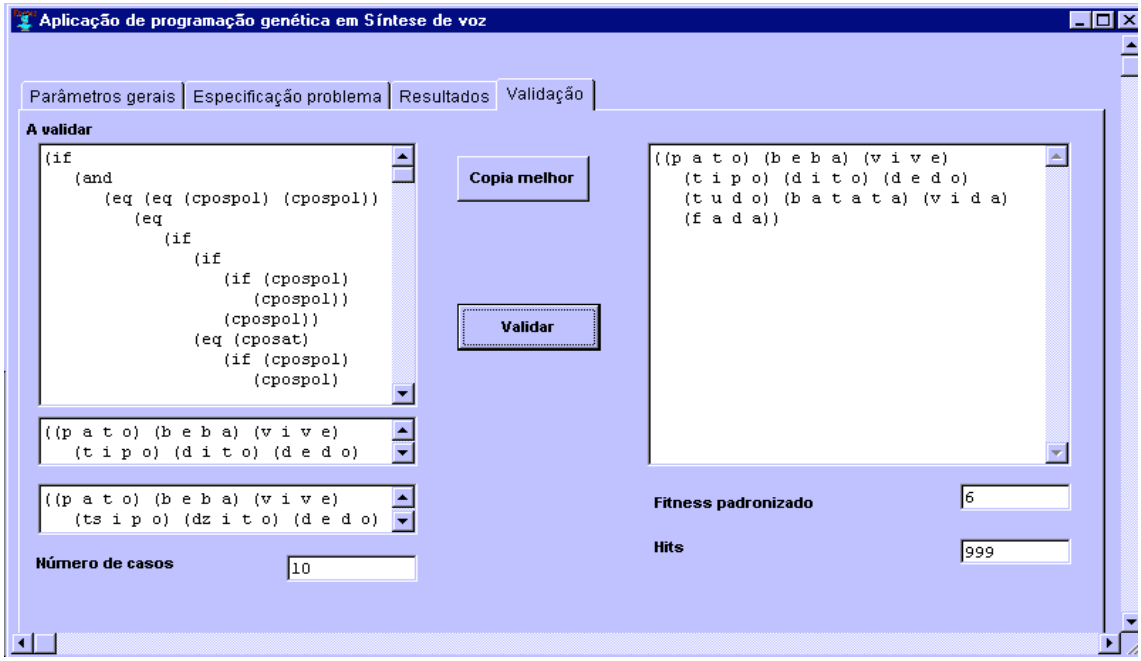


FIGURA 4.6 - Interface para validação de respostas produzidas na execução

Após uma execução, é possível obter diretamente o melhor indivíduo através do botão copiar. Para validação, usa-se outros casos de fitness que apresentem as mesmas regras que eram objeto da execução. O resultado da validação indica o fitness do indivíduo e as respostas produzidas. Esta interface serve também para verificar em soluções com pouca adaptação, quais as regras não descobertas e também problemas de convergência que venham a ocorrer em algumas execuções.

Após a descrição do sistema a ser utilizado para as execuções da técnica e para sua aplicação ao problema de conversão de letra para fonema, é necessária uma descrição detalhada da modelagem que tornará possível encontrar soluções. Todos os parâmetros e funções necessárias, ilustrados nas interfaces mostradas acima, são escolhidas a partir da análise descrita no próximo capitulo.

5 Modelagem do problema de conversão letra-fonema, usando programação genética

De acordo com a descrição dos contextos e regras que afetam o processo de conversão de letras para sons, é possível concluir que elas poderiam ser representadas através de programas de computador. Se fossem implementados por programadores humanos, é provável que este programas teriam a forma:

SE (letra atual é x) então o fonema é y SENÃO (subregra).

Sendo assim, o objetivo do trabalho de modelagem é fornecer a Programação Genética condições de compor soluções iguais ou semelhantes através de um processo evolutivo. De acordo com [KOZ 92] a preparação para execução da programação genética envolve cinco passos básicos:

- Determinação do conjunto de terminais
- Determinação do conjunto de funções
- Determinação da medição e casos de fitness
- Determinação dos parâmetros e variáveis de controle da execução, e
- Determinação do método para designação do resultado.

As duas últimas etapas podem variar de uma execução para outra sem qualquer dependência de uma estratégia específica para resolução do problema; entretanto a definição de funções, terminais e fitness está diretamente relacionada à uma forma de resolução proposta. Problemas mais simples do que o descrito neste trabalho, podem não necessitar de uma definição de estratégia ou podem possuir uma forma única e natural para obtenção de resultados. A estratégia, engloba a definição da estrutura das soluções e dos resultados. Problemas semelhantes foram descritos em [BAN 98] e [AND 94].

Um exemplo clássico de uma estrutura pré-definida para a solução, é o problema de regressão simbólica múltipla [KOZ 92]. Neste problema, uma solução deve retornar dois valores e não somente um. Para obter soluções corretas, uma função LIST2 foi utilizada como raiz; todas as soluções possuem esta como a função que é responsável por retornar os valores, o que garante a produção de uma resposta múltipla. Uma das soluções descritas no exemplo foi:

(LIST2 (- (X3 X1) (* X4 X2)) (+ (* X3 X2) (* X1 X4)))*

A opção pela estrutura das soluções, é dependente da estrutura das respostas aos casos de fitness. Caso a resposta exija a produção de mais de um valor, existem duas alternativas possíveis: i) A produção de uma resposta com todos os valores ou; ii) a avaliação sucessiva do indivíduo e a composição da resposta final com as diversas respostas unitárias produzidas pela solução. Entretanto, antes de efetuar uma escolha é necessário descrever o processo de definição de casos e avaliação de fitness.

5.1 Definição de casos e avaliação de fitness

A atividade de conversão letra-fonema abordada no trabalho, pode ser resumida como uma aplicação de regras sobre palavras ou letras, para descoberta de fonemas

correspondentes. Sendo assim, o passo inicial para determinação de uma ou mais estratégias para busca de soluções através da programação genética, é definir se casos de fitness consistirão em um conjunto de letras ou palavras. A utilização de letras isoladas fica completamente descartada a partir da constatação de que duas letras podem significar um único fonema, ou ainda que uma determinada letra produz diferentes fonemas, de acordo com a anterior ou posterior a ela.

A definição dos casos de fitness, portanto recaiu sobre palavras. Considerou-se para efeito de medição de fitness uma palavra como um caso de fitness. A pontuação final do indivíduo, é medida a partir de todas as palavras que constituem o conjunto de casos. É importante salientar que o conjunto de palavras deve representar de forma significativa as regras a serem implementadas pelas soluções. Através desta afirmação, podemos concluir que se determinada letra estiver envolvida em diferentes regras, mas se apenas a ocorrência de uma ou outra destas regras estiver representada no conjunto de casos de fitness, as soluções somente poderão implementar aquelas que estiverem representadas pelos casos fornecidos na execução em questão.

O conjunto de casos de fitness é definido na interface de entrada da aplicação como uma lista de palavras, cada uma delas composta por uma lista de letras ((p a t o) (t i p o)...). As respostas corretas para cada caso, são igualmente especificadas como listas de fonemas ((p a t o) (ts i p o)...). A definição de casos e respostas respectivas em forma de lista foi escolhida pela facilidade encontrada na linguagem LISP para o processamento de listas; entretanto outras estruturas de dados podem ser utilizadas, não comprometendo o processo de busca de soluções.

Avaliando-se a resposta produzida por cada indivíduo, o sistema calcula o fitness básico utilizando as seguintes regras:

- Caso o fonema produzido seja igual ao fonema esperado na posição específica, são computados três pontos.
- Caso o fonema produzido seja diferente na posição, mas seja um fonema esperado em qualquer outra posição da resposta, é computado um ponto.

O fitness padronizado deve sempre indicar melhores soluções tendo menores valores, tendendo a zero. Neste problema, é calculado subtraindo-se o fitness básico obtido do maior valor possível para o mesmo. Assim, melhores soluções sempre terão valores menores.

5.2 Estratégias para descoberta automática de programas para conversão letra-fonema

Após uma descrição detalhada da estrutura de casos de fitness e suas respostas corretas, torna-se possível analisar alternativas para a estrutura dos indivíduos. Para que haja uma avaliação correta e padronizada do fitness, é importante a definição de um formato básico das respostas produzidas. Como exemplo, podemos citar outras aplicações como no caso da obtenção de uma equação matemática para resolver determinado problema. Se a resposta esperada for um número e a medição de fitness constituir em avaliar a diferença entre o número produzido e o esperado, é vital que as soluções candidatas produzam como resultado um valor numérico, do contrário a avaliação de fitness torna-se uma atividade complexa e sujeita a erros.

Na modelagem proposta, tanto as palavras, quanto seus respectivos fonemas que constituem a resposta correta para cada caso, são informados como listas. Sendo assim, a resposta avaliada na função que mede o fitness de um indivíduo deverá naturalmente vir em forma de uma lista de elementos. Surgem duas abordagens distintas: deixar a cargo da solução produzir a lista completa dos fonemas ou deixar a cargo das soluções a produção de cada fonema individualmente. Nesta última, a função de avaliação de fitness executa uma solução diversas vezes de acordo com o número de letras de cada caso de fitness. Dentro de cada abordagem existem ainda diferentes alternativas, sendo algumas delas descritas a seguir.

Seguindo a primeira abordagem, o programa será executado uma única vez para cada palavra e deverá produzir a lista completa de fonemas correspondentes. A obtenção de respostas em forma de lista pode ocorrer de duas maneiras: i) inserindo-se funções que retornam valores em forma de lista como por exemplo LIST, CONS, ou; ii) inserindo-se uma função de domínio do problema, que tem por objetivo alterar uma variável de estado, que representa a resposta para o problema. Na primeira alternativa, a resposta produzida pelo programa é avaliada, enquanto que na segunda uma variável global, modificada apenas pela função responsável pela composição da lista de fonemas será avaliada.

A simples inserção de funções que retornam uma lista no conjunto de funções, não permite afirmar que a resposta virá na forma de uma lista; não há garantia de que estas funções serão usadas, tampouco que serão escolhidas como função inicial. Uma forma de contornar este problema é a utilização de uma estrutura rígida, onde uma função semelhante a LIST2, citada no exemplo da regressão simbólica, consistiria sempre na função raiz de qualquer solução. Porém o número de elementos da lista, varia de acordo com o número de fonemas que compõe uma palavra; como o conjunto de casos de fitness pode ser composto de palavras de diversos tamanhos, a fixação do número de argumentos de funções que irão produzir a resposta é um obstáculo considerável. Restringir o tamanho dos casos é indesejável do ponto de vista da exploração das variações existentes no idioma; além disso, não há uma correspondência exata no número de letras e fonemas.

A segunda alternativa é a utilização de uma função para modificação de uma variável de estado que represente a resposta para cada caso. A função implementada (FRESULT) tem por objetivo receber um parâmetro que é o fonema a acrescentar na resposta e modificar uma variável que representa o estado da resposta (*lvresult*). A função, deve fazer parte do conjunto de funções, com um argumento. Sempre que for executada em uma solução, será inserido ao final da lista o argumento recebido, caso este não possua valor nulo. O código da função está descrito abaixo.

```
(defun FRESULT (pfone)
(setf lvtemp '())
(if (not (eq pfone nil)) (setf lvtemp (cons pfone lvtemp)))
  (setf vcontdo (- (length *lvresult*) 1))
  (loop until (< vcontdo 0) do
    (setf lvtemp (cons (nth vcontdo *lvresult*) lvtemp))
    (setf vcontdo (- vcontdo 1)))
   (setf *lvresult* lvtemp)
(values pfone))
```

Além da função que modifica a resposta, uma função de repetição e funções de incremento de posição devem fazer parte do conjunto de funções ou terminais. Como a palavra é composta por diversas letras, as regras devem ser aplicadas seqüencialmente a cada uma delas; portanto, fica a cargo do processo evolutivo descobrir como controlar a seqüência de aplicação, além de compor as respectivas regras. Para atingir este objetivo é criada uma função de repetição que executa diversas vezes um determinado conjunto de instruções recebidas como argumentos, de acordo com o tamanho da palavra que é objeto de avaliação. Uma função de repetição específica para o domínio do problema faz-se necessária, na medida que usando funções de repetição tradicionais, facilmente verifica-se a ocorrência de um laço infinito ou ainda a incidência de indivíduos cuja avaliação de fitness torna-se extremamente lenta e complexa. As funções de incremento, aumentam o valor de uma variável de estado (*vposletra*) que define a posição atual, influenciando funções que retornam a letra na posição atual, bem como as letras nas posições anteriores e posteriores. Um exemplo de uma possível solução que apresenta esta estrutura, está descrito abaixo.

```
(DOLFON (IF (IF (FRESULT (CPOSAT)) (IF (IF T15 T13)
(INCPOS) (CPOSAT)) (FRESULT T16)) T4 T6))
```

Na segunda abordagem avaliada, as soluções não produzem uma lista em uma execução única para cada palavra, mas são executadas várias vezes de acordo com o número de letras. Nesta abordagem, existem duas alternativas possíveis, uma que mantém a função que modifica a variável de estado, representando a resposta e outra, que utiliza o retorno do programa avaliado e insere-o na lista. A função de repetição em qualquer uma das alternativas é retirada do conjunto de funções.

Na primeira alternativa, a função FRESULT é mantida no conjunto de funções, e pode até mesmo ser chamada diversas vezes por cada solução. É comum nesta situação, a produção de respostas com um número de elementos superior ao esperado. Esta abordagem permite por exemplo que na execução de uma solução candidata, a mesma insira mais de um valor na resposta. Abaixo, segue um exemplo desta estrutura.

```
(IF (IF (FRESULT (CPOSAT)) (IF T11 T11 T15)) T6 T8)
```

A última estratégia das três possíveis é semelhante à anterior, porém a função FRESULT é retirada do conjunto de funções e é usada na avaliação de fitness. Nesta situação, o programa também é avaliado diversas vezes e a resposta do programa é inserida pela função de avaliação de fitness na resposta. A partir da não utilização de uma função específica para composição da lista de fonemas, a responsabilidade das soluções fica sendo somente a de elaborar a regra e retornar o fonema específico para cada letra. A função de avaliação de fitness tem a responsabilidade de compor a resposta final, usando os valores retornados a cada execução de uma solução. Nesta abordagem é possível garantir que somente um único valor de resposta será produzido pelo indivíduo.

Diversas execuções iniciais permitiram verificar vantagens e desvantagens de cada alternativa. Todas elas foram capazes de encontrar boas soluções; entretanto a utilização de funções de repetição, apresentou um maior tempo de processamento. Outra constatação básica foi a dependência da função de produção de resultados nas primeiras duas estratégias. A não utilização ou utilização em excesso desta instrução causa diversos problemas como produção de respostas nulas ou com tamanho superior ao esperado. A última alternativa mostrou um maior equilíbrio no consumo de recursos e

na evolução e ainda maior clareza nas soluções finais e por todos estes motivos foi escolhida; entretanto, é necessário salientar que qualquer uma das alternativas anteriores seria uma escolha válida.

Além da definição de uma estrutura de solução, outra definição importante é a forma como as soluções receberão as entradas do problema, que são constituídas pelas letras que formam palavras.

Se a entrada para o problema fosse a lista inteira, soluções deveriam possuir a habilidade de extrair sucessivamente cada letra e aplicar as regras. Entretanto, a atividade de extrair a partir de uma lista um elemento envolve funções da linguagem como FIRST, SECOND, ou ainda uma função para obter um elemento em uma posição específica como por exemplo NTH. Esta necessidade de extração sucessiva deve ser realizada seqüencialmente, caso a estrutura dos indivíduos siga a abordagem na qual é usada uma função de repetição como parte do conjunto de funções.

Após algumas simulações e avaliações iniciais foi possível verificar a dificuldade de encontrar soluções capazes de extrair elementos de lista, usando somente as funções básicas da linguagem. O uso de funções tradicionais como FIRST, SECOND torna-se inviável, uma vez que extraem sempre um elemento em posição específica. Uma função como NTH para ser utilizada corretamente deve utilizar como um de seus parâmetros a posição da letra na lista, posição esta que deve ser incrementada seqüencialmente. Embora esta alternativa seja viável, o objetivo do processo não é descobrir programas capazes de extrair elementos de listas e sim soluções que implementem regras sobre estes elementos. Além disso, uma estratégia que exige das soluções a extração de elementos de lista a partir de funções primitivas se mostra extremamente custosa em termos de desempenho e recursos do sistema.

Para obter melhor rendimento do sistema e maior facilidade para obter seqüencialmente cada elemento (letra) da lista (palavra), foram implementadas funções de domínio do problema que tem por objetivo retornar a posição atual (CPOSAT), anterior(CPOSAN1), posterior(CPOSPO1), levando em consideração uma variável que define a posição atual na lista. Além destas funções, foram criadas ainda funções para incremento da posição atual da lista em uma (INCPOS) e duas (INCPOS2) unidades. Estas funções são importantes, uma vez que o sistema deverá aplicar as regras sucessivamente sobre cada uma das letras ou poderá de acordo com a solução criada optar pela não aplicação a uma determinada letra. Cabe portanto às soluções descobrir a forma e seqüência correta de uso destas funções. Diversos outros problemas descritos em [KOZ 92],[KOZ 98] utilizam funções de domínio, o que caracteriza a inserção de conhecimento prévio no processo de busca de soluções.

As funções para incremento de posição fazem parte do conjunto de terminais apenas na abordagem que deixa a cargo da solução a repetição e incremento da posição atual. Nas abordagens em que a função de avaliação de fitness executa diversas vezes cada indivíduo, o incremento é feito antes de cada avaliação.

As entradas poderiam também ocorrer sob a forma de variáveis. Na avaliação, a própria função de avaliação extrairia a letra atual, as letras anteriores e posteriores e atribuiria à variáveis globais a cada avaliação. As soluções processariam variáveis e não as funções de entrada citadas acima. Entretanto esta implementação não é viável, principalmente quando a repetição fica a cargo das soluções. Por este motivo, a escolha recaiu sobre funções de entrada, que funcionam para todas as estratégias descritas anteriormente.

6 Resultados

O conjunto de regras para conversão letra para fonema em língua portuguesa é extremamente amplo. Míriam Barbosa da Silva em [SIL 93] descreve um conjunto de mais de oitenta regras que cobrem os diversos contextos em que letras são empregadas. O item 2.5 descreveu anteriormente os diversos subgrupos que concentram estas regras.

A aplicação da técnica ao problema, envolveu a seleção de subconjuntos dessas regras. A aplicação utilizando como base o conjunto completo das regras poderia exigir um excesso de funções, terminais e casos de fitness. Além disso, as soluções também tenderiam a apresentar uma complexidade muito grande, dificultando uma análise mais detalhada, necessária para compreensão do trabalho realizado. A descoberta de uma solução única para todas as regras, além de complexa poderia exigir um tempo inaceitável para ser encontrada.

Os diferentes casos aos quais a técnica foi aplicada foram elaborados para refletirem os diferentes contextos como fonemas simbolizados por mais de uma letra, letras simbolizando diferentes fonemas, letras que não representam fonema algum e ainda influências das letras anteriores e posteriores na produção do fonema relativo à letra na posição atual. Os casos foram definidos também com o objetivo de comparar os resultados em situações de maior e menor complexidade, uma vez que a capacidade de encontrar soluções para problemas de complexidade crescente é uma característica desejável em qualquer técnica que simule um processo de aprendizado em máquinas.

A documentação dos resultados obtidos é descrita através dos seguintes aspectos:

- Citação e exemplificação das regras a serem descobertas;
- Definições para a execução, envolvendo conjunto de terminais, funções, casos de fitness e demais parâmetros de controle;
- Evolução do melhor, pior e fitness médio da população;
- Regras implementadas pelas soluções encontradas;
- Evolução da complexidade estrutural da população e melhor solução;
- Validação da solução;

Para cada subconjunto, foram realizadas diversas execuções para obtenção de resultados com bom grau de representatividade. Devido à natureza probabilística do processo, as diferentes execuções apresentaram um grau de variabilidade, tanto em valores de fitness quanto na combinação de funções e terminais. Visando a uma análise mais detalhada, para cada caso será descrita uma execução que ilustra bons resultados. É preciso salientar entretanto, que bons resultados não foram obtidos somente na execução descrita, em diversas outras foram obtidos resultados igualmente aceitáveis, com indivíduos com diferentes estruturas e graus de adaptação.

Alguns dos parâmetros de controle foram mantidos sem modificação em todas as execuções. A manutenção destes parâmetros permite uma comparação isenta entre as diversas alternativas e casos documentados. O tamanho da população e o número de gerações foram os parâmetros que sofreram variações em determinadas execuções e serão descritos nas definições de cada caso, os demais parâmetros empregados, estão resumidos na tab. 6.1.

TABELA 6.1 – Parâmetros utilizados em todas as execuções

Parâmetro	Valor/Opção utilizada
Probabilidade de cruzamento em qualquer ponto	0.40
Probabilidade de cruzamento em ponto de função	0.45
Probabilidade de Reprodução	0.10
Probabilidade de Mutação	0.05
Profundidade máxima na geração inicial	6
Profundidade máxima para operações genéticas	10
Método para geração da população inicial	Subida Parcial
Método de seleção de indivíduos	Proporcional ao Fitness
Critério para término do processo	Atingir o número máximo de gerações ou descoberta de uma solução 100% correta

A definição destes parâmetros foi realizada após execuções iniciais e após revisões de problemas citados por outros autores [KOZ 92], [KOZ 98]. Após esta análise foi possível levantar algumas questões importantes sobre estes parâmetros: i) A probabilidade de cruzamento em pontos de função acarreta em uma maior alteração nos indivíduos, enquanto cruzamentos em qualquer ponto torna-se as modificações mais aleatórias; ii) probabilidades de reprodução muito altas fazem com que o processo encontre rapidamente soluções com um bom grau de adaptação, mas apresentam um problema da convergência prematura, ou seja, após encontrar uma solução bem adapta esta não mais evolui; iii) as profundidades devem ser limitadas até um nível aceitável, caso contrário os indivíduos tem sua complexidade aumentada indefinidamente.

Nos problemas abordados em [KOZ 92], profundidades máximas na geração inicial e após operações genéticas, obtinha-se que os valores empregados eram respectivamente 6 e 17 nós. Os valores definidos neste trabalho, mantém o tamanho para a geração inicial, porém utilizam profundidade de apenas 10 nós para gerações subseqüentes. Analisando as regras a serem implementadas nos subconjuntos que serão descritos posteriormente, será possível perceber que em nenhuma das situações são necessários valores superiores; entretanto pode ser necessário um aumento destes parâmetros caso o conjunto de regras seja mais complexo.

A evolução das soluções é demonstrada através de um gráfico, englobando o melhor fitness da geração, o melhor fitness da execução, o pior fitness e ainda a média da população.

O código da melhor solução para cada caso será descrito na íntegra com objetivo de possibilitar uma avaliação detalhada da estratégia e forma como as regras são

implementadas. O conjunto completo de instruções entretanto, é de difícil entendimento, sendo assim as regras serão descritas também em uma linguagem natural.

Outro fator importante para avaliação dos resultados é o número de pontos (funções e terminais) das soluções. Este o qual é chamado de complexidade estrutural. A evolução desta medida será descrita também por um gráfico, mostrando complexidade do melhor e pior indivíduo, bem como a média da população.

A validação é realizada para comprovar que a solução produz efetivamente as respostas corretas ou não, conforme as regras implementadas. Para evitar qualquer tendência associada ao processo de treinamento, a validação será realizada com palavras não utilizadas no treinamento.

A seguir são detalhadas as execuções para cada subconjunto.

6.1 Caso 1 - Relação um para um e letras "t" e "d" antes de "i"

O primeiro subconjunto de regras ao qual a técnica será aplicada engloba as regras mais simples para conversão:

- relação direta entre letra e fonema, onde determinadas letras representam sempre o mesmo fonema independente do contexto. Estas letras no caso são “p”, “b” e “v”.
- letras “t” e “d” quando ocorrem antes de “i”, representam diferentes fonemas. A notação para estes fonemas será ts e dz, nos demais casos o fonema é utilizado o símbolo correspondente à própria letra. Esta variação é comum em determinadas regiões do Brasil, como por exemplo nas regiões Sul e Sudeste.

6.1.1 Resultados utilizando somente funções básicas

A execução descrita inicialmente emprega apenas funções básicas da linguagem, nenhuma função de domínio é utilizada. O objetivo é avaliar a capacidade de solução usando exclusivamente funções tradicionais da linguagem. Os parâmetros para esta execução estão descritos na tab. 6.2.

TABELA 6.2 - Parâmetros para o caso 1, usando somente funções básicas

Parâmetro	Definição para o problema
Conjunto de funções	(EQ IF IF AND OR)
Número de argumentos das funções	(2 2 3 2 2)
Conjunto de terminais	((CPOSAT) (CPOSPO1) t d i ts dz)
Casos de fitness	((p a) (b e) (v i) (t i) (d i) (t a t o) (d a d o) (d i d i) (d i t i))
Resposta esperada	((p a) (b e) (v i) (ts i) (dz i) (t a t o) (d a d o) (dz i dz i) (dz i ts i))
Número de gerações	G = 50
Tamanho da população	M = 500

Os casos de fitness devem constituir-se em uma amostra representativa das regras a serem implementadas pela solução. Com relação às regras em questão, é necessário que as combinações das letras “t” ou “d” antes de “i” apareçam com razoável

freqüência, uma vez que todas as combinações restantes caracterizam a regra geral. Tanto a sílaba (v i) quanto (b e) representam a regra geral, mesmo que usem letras diferentes. A regra geral para este caso representa um ponto de convergência prematura.

O conjunto de funções possui uma função para comparação de igualdade (EQ), a função IF que pode ser empregada com dois ou três argumentos e por este motivo aparece duas vezes no conjunto. As funções AND e OR são necessárias para comparações mais complexas que envolvam duas condições.

O conjunto de terminais é composto apenas por uma função que retorna a letra na posição atual e por outra que retorna a letra na próxima posição. Esta definição garante o conceito de suficiência, pois é possível implementar uma solução correta apenas com estas duas entradas, sem necessidade das letras nas demais posições. Segue-se no exemplo o princípio de que funções sem argumentos fazem parte do conjunto de terminais.

O fitness básico máximo para um indivíduo é de 78 e é computado através da soma da multiplicação de todos os fonemas esperados por 3. O fitness padronizado é calculado subtraindo-se o fitness do indivíduo deste valor máximo. O gráfico da evolução do fitness demonstra a evolução do fitness durante o processo, no gráfico são demonstrados o melhor fitness da geração, da execução, a média de fitness da população e o fitness do pior indivíduo. A fig. 6.1 permite visualizar a evolução das soluções para esta execução.

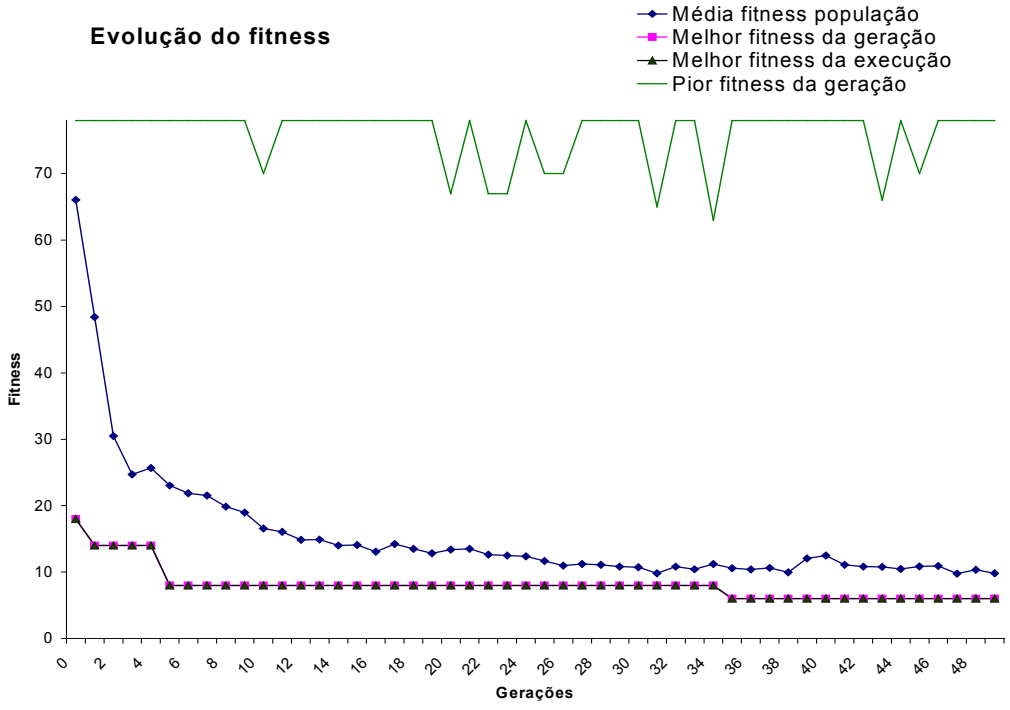


FIGURA 6.1 - Evolução de fitness para o caso 1, usando somente funções básicas

Os valores de fitness do melhor indivíduo da geração e da execução permanecem o mesmo durante todo o processo (fig.6.1). Neste exemplo o fitness do melhor indivíduo encontrado na geração 35 foi 6 e permaneceu durante o restante das gerações. A média do fitness da população cai de forma constante e a partir da obtenção da melhor resposta permanece estável, sofrendo apenas pequenas variações. Melhorias no melhor fitness ocorrem de forma brusca; uma combinação correta de instruções implica

em respostas corretas para diversos fonemas. Este comportamento é uma característica intrínseca do problema, uma vez que uma mesma regra é aplicável a vários casos.

Após a realização de três diferentes execuções, em nenhuma delas foi encontrada uma resposta 100% correta para o problema, a melhor solução tem um percentual aproximado de 93% de acerto. O código do melhor indivíduo da execução documentada para o caso, foi:

(IF (OR (AND (EQ T5 T4) (IF (OR (AND (EQ T5 T4) (CPOSPO1)) (CPOSAT)) (IF (CPOSAT) T7 (IF (AND (CPOSAT) (AND T6 (CPOSAT))) T5 T5)) (OR (CPOSAT) (EQ (IF T3 T5) (IF T6 (EQ T5 T4) T7))))) (EQ (IF (CPOSAT) T5) (IF T6 (OR (IF (EQ (IF (CPOSPO1) (CPOSAT) (OR (CPOSAT) T5)) (IF T4 T4 (CPOSAT))) (CPOSPO1) T7) T3) T7))) (AND (AND T6 (CPOSAT)) T7) (OR (OR (OR (CPOSAT) (CPOSAT)) (EQ (CPOSPO1) (OR T7 (IF T6 (OR T3 (OR (CPOSAT) (OR (CPOSAT) T7))) (CPOSAT))))) (OR (IF (EQ T5 T4) (CPOSAT) T7) T3)))

Abaixo, um resumo das regras implementadas pelo indivíduo:

Se letra na posição atual é igual a "d" (T4) e a letra na próxima posição é igual a "i" (t5)
    Retorna dz
Senão
    Retorna a letra na posição atual

O código envolve diversos pontos e combinações que não influenciam na produção da resposta. Diversas comparações como por exemplo (EQ T5 T4) produzem sempre valores nulos ou em outras situações como em (IF T4 T4 (CPOSAT)) produzem sempre o mesmo valor, o que neste caso corresponde à posição atual.

O pior fitness da geração permanece o pior valor possível durante todo o processo. Indivíduos representando a pior solução, na maioria dos casos possuem códigos que produzem valores nulos independentemente de valores de entradas.

Para o melhor indivíduo acima a complexidade estrutural é de 92 pontos. A fig. 6.2 mostra a evolução da complexidade estrutural ao longo das gerações.

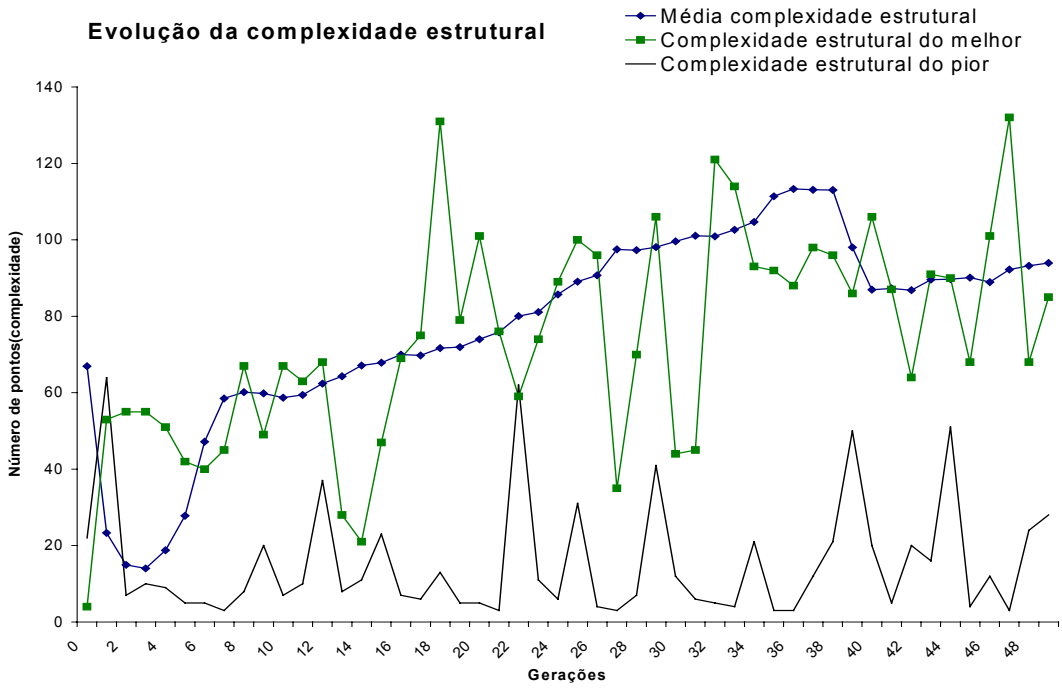


FIGURA 6.2 – Complexidade estrutural para o caso 1, usando somente funções básicas

Existe uma variação da complexidade tanto para o pior quanto para o melhor indivíduo. A complexidade do melhor indivíduo entretanto tende a aumentar ao longo das gerações; é comum observar que indivíduos que se constituem em boas soluções apresentam número cada vez maior de pontos. A média da população apresenta um crescimento constante e na maior parte do tempo estável; com isto, é possível concluir que a melhoria do fitness neste caso, é acompanhada por um aumento na complexidade estrutural.

A validação foi realizada utilizando-se 10 casos com um fitness total de 126. A adaptação apresentado nestes casos foi de aproximadamente 98%, com um fitness de 3. A tab. 6.3 resume os resultados da validação.

TABELA 6.3 – Validação para o caso 1, usando somente funções básicas.

Palavras usadas	Resultados esperados	Resultados produzidos
((p a t o) (b e b a) (v i v e) (t i p o) (d i t o) (d e d o) (t u d o) (b a t a t a) (v i d a) (f a d a))	((p a t o) (b e b a) (v i v e) (ts i p o) (dz i t o) (d e d o) (t u d o) (b a t a t a) (v i d a) (f a d a))	((p a t o) (b e b a) (v i v e) (t i p o) (dz i t o) (d e d o) (t u d o) (b a t a t a) (v i d a) (f a d a))

6.1.2 Inserção de funções de domínio

Nesta execução, o subconjunto de regras a serem descobertas permanece o mesmo descrito no item 6.1.1, mas são inseridas funções que implementam detalhes da regra, que chamamos de funções de domínio. Um dos objetivos do trabalho é comparar a influência das funções de domínio no processo. As definições necessárias para a execução estão resumidas abaixo (tab. 6.4).

TABELA 6.4 - Definições para o caso 1 com funções de domínio

Parâmetro	Definição para o problema
Conjunto de funções	(EQ IF IF AND FEHTD FRTDI)
Número de argumentos das funções	(2 2 3 2 2 1)
Conjunto de terminais	((CPOSAT) (CPOSPO1))
Casos de fitness	((p a) (b e) (v i) (t i) (d i) (t a t o) (d a d o) (d i d i) (d i t i))
Resposta esperada	((p a) (b e) (v i) (ts i) (dz i) (t a t o) (d a d o) (dz i dz i) (dz i ts i))
Número de gerações	G = 50
Tamanho da população	M = 500

Além das funções usadas na execução anterior, fazem parte do conjunto duas funções que implementam detalhes da exceção à regra geral. A função FEHTD recebe dois argumentos e responde com verdadeiro, caso o primeiro seja "t" ou "d" e o segundo seja a letra "i". A função FRTDI recebe um argumento, caso seja a letra t retorna ts, caso seja a letra d retorna dz. Em ambas as funções, não havendo condição verdadeira o retorno da função é o valor NIL (falso). O código das funções de domínio pode ser visualizado no Anexo 3.

O conjunto de terminais não possui as letras ("t", "d" e "i"); isto porque os testes envolvendo as letras são realizados por uma função de domínio; tampouco é necessário que variáveis representem os fonemas ts e dz, uma vez que a função FRTDI produz um ou outro de acordo com o argumento recebido. O conceito de suficiência sem estes terminais é satisfeito de forma plena.

A fig. 6.3 mostra a evolução das soluções para esta execução. A média de adaptação da população como um todo evolui rapidamente nas primeiras gerações e permanece estável a partir da geração 8. Os valores de fitness do melhor indivíduo da execução e da geração atual, também aqui permanecem os mesmos durante todo o processo, ou seja, não há um decréscimo do melhor valor em determinada geração.

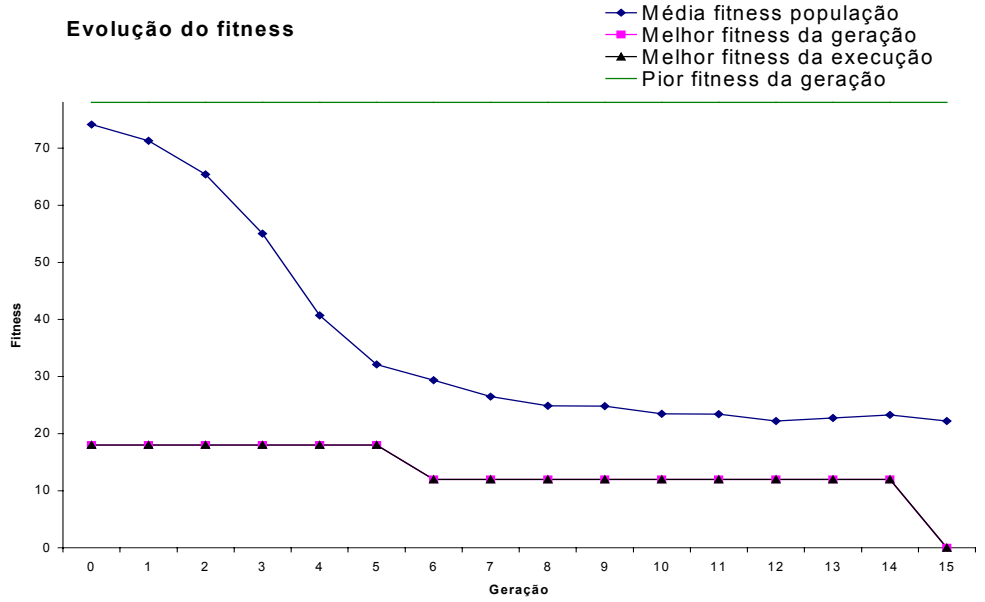


FIGURA 6.3 - Gráfico de fitness para o caso 1 usando funções de domínio

Ao contrário da execução descrita na alternativa anterior, nesta foi possível encontrar indivíduos que resolvessem completamente o problema. Um indivíduo com maior grau de adaptação possível é encontrado já na geração 15, satisfazendo desta forma o critério de término, que determina que ao ser encontrada uma solução 100% correta, a execução é encerrada. O código da solução 100% correta encontrada foi:

(IF (CPOSAT) (IF (FEHTD (CPOSAT) (CPOSPO1)) (FRTDI (IF (FRTDI (CPOSAT)) (CPOSAT) (CPOSAT))) (CPOSAT)) (CPOSAT))

Nesta solução é possível perceber claramente as seguintes regras:

Se a letra na posição atual não é um valor nulo
    Se a letra na posição atual é igual a "t" e a letra na próxima é igual "i"
        Retorna ts ou dz, de acordo com a letra na posição atual (Aplica função FRTDI sobre letra na posição atual)
    Senão
        Retorna a letra na posição atual
Senão
    Retorna a letra na posição atual

Embora o indivíduo apresente ainda instruções que não contribuem para produção de respostas corretas, é possível verificar que as soluções encontradas nesta execução tendem a se aproximar mais de implementações desenvolvidas por programadores humanos. Devido ao fato de funções como FRTDI e FEHTD implementarem parte da regra, o trabalho do processo evolutivo é facilitado e consequentemente, a melhor solução é encontrada rapidamente.

É importante salientar que a maioria das soluções ainda implementa a regra geral, ou seja produz sempre a letra na posição atual, o que significa uma tendência ao mínimo local, uma convergência prematura de boa parte dos indivíduos.

A complexidade estrutural do melhor indivíduo encontrado é de 14 pontos. A fig. 6.4 descreve a evolução da complexidade estrutural na execução.

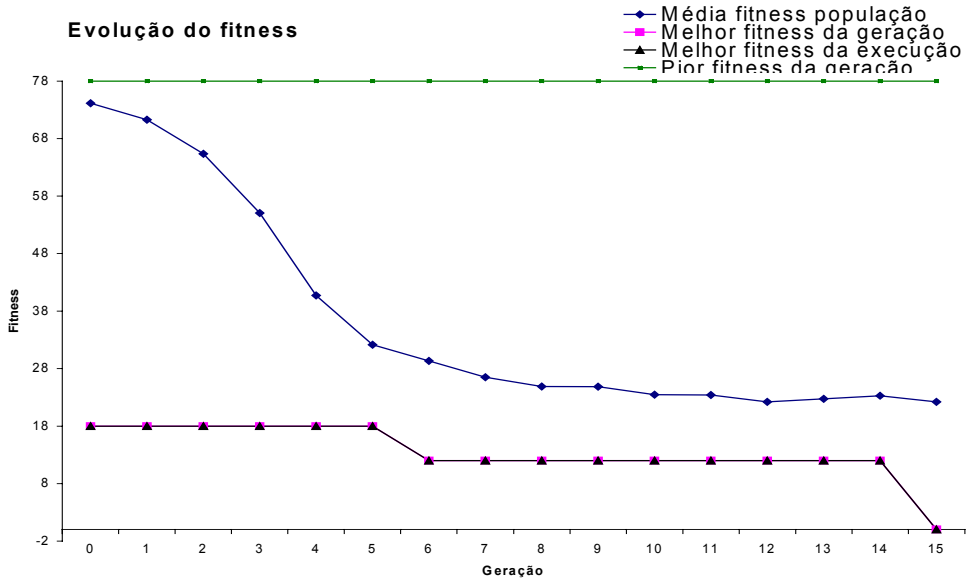


FIGURA 6.4 - Evolução da complexidade estrutural para o caso 1 usando funções de domínio

A variação da complexidade de melhores e piores indivíduos também ocorre nesta situação; porém os valores apresentados são muito inferiores àqueles verificados sem as funções de domínio. Até a geração 15, na qual o processo foi finalizado, a complexidade média dos indivíduos era de aproximadamente 23 pontos, enquanto na execução documentada anteriormente era de 68 pontos. A diferença de pontos entre os melhores indivíduos é 78 e mesmo levando-se em consideração que o melhor indivíduo havia sido obtido anteriormente na geração 35 é extremamente significativa.

Validando-se a solução com outras palavras obtêm-se sempre respostas corretas para as regras envolvidas no treinamento. A tab. 6.5 ilustra o processo de validação.

TABELA 6.5 – Validação para o caso 1, usando funções de domínio

Palavras usadas	Resultados esperados	Resultados produzidos
((p a t o) (b e b a) (v i v e) (t i p o) (d i t o) (d e d o) (t u d o) (b a t a t a) (v i d a) (f a d a))	((p a t o) (b e b a) (v i v e) (ts i p o) (dz i t o) (d e d o) (t u d o) (b a t a t a) (v i d a) (f a d a))	((p a t o) (b e b a) (v i v e) (ts i p o) (dz i t o) (d e d o) (t u d o) (b a t a t a) (v i d a) (f a d a))

6.2 Caso 2 - Nasalização de vogais

Neste caso, o conjunto de regras a se obter envolve três situações básicas:

- relação direta entre letra e fone;
- ocorrência especial das letras "t" e "d" antes de "i" conforme o caso anterior e;
- nasalização de uma vogal quando da ocorrência de "m" ou "n" após uma vogal em fim de palavra ou antes de uma consoante. Neste caso, as letras citadas não simbolizam um fonema específico e funcionam como elemento

de nasalização da letra anterior. As notações (ax~, e~, i~, o~, u~) são usadas para representação destes fonemas.

Analisando as regras acima, é possível observar que na maior parte dos casos, as letras citadas continuam tendo como representação fonêmica a própria letra. As variações ocorrem em contextos específicos, envolvendo segmentos anteriores e posteriores. As soluções deverão ser capazes de concluir que em determinado contexto nenhum fonema deve ser gerado, ou outra representação que não a própria letra deve ser retornada. A tab. 6.6 resume as definições para o caso.

TABELA 6.6 - Definições para o caso 2

Parâmetro	Definição para o problema
Conjunto de funções	(EQ IF IF AND AND NOT FEHVOGAL FNASAL FEHMN FEHTD FRTDI)
Número de argumentos das funções	(2 2 3 2 3 1 1 1 1 2 1)
Conjunto de terminais	((CPOSAT) (CPOSPO1) (CPOSAN1) (CPOSAN2) (CPOSPO2) NIL)
Casos de fitness	((a n t a n t) (e n t e n t) (i n t i n t i) (o n t o n t) (u n t u n t) (a n d a n d) (e n d e n d) (i n d i n d) (o n d o n d) (u n d u n d) (a m p a m p) (e m p e m p) (i m p i m p) (o m p o m p) (u m p u m p) (a m b a m b) (e m b e m b) (i m b i m b) (o m b o m b) (u m b u m b) (a m a) (a n a))
Resposta esperada	((ax~ t ax~ t) (e~ t e~ t) (i~ ts i~ ts i) (o~ t o~ t) (u~ t u~ t) (ax~ d ax~ d) (e~ d e~ d) (i~ dz i~ d) (o~ d o~ d) (u~ d u~ d) (ax~ p ax~ p) (e~ p e~ p) (i~ p i~ p) (o~ p o~ p) (u~ p u~ p) (ax~ b ax~ b) (e~ b e~ b) (i~ b i~ b) (o~ b o~ b) (u~ b u~ b) (a m a) (a n a))
Número de gerações	G = 200
Tamanho da população	M = 1,000

A complexidade das regras neste exemplo é maior do que a apresentada no caso 1. A regra para nasalização das vogais, possui uma implementação extensa, pois para cada uma das vogais, o fonema correspondente à nasalização é diferente. Execuções iniciais que utilizaram apenas funções básicas demonstraram uma imensa dificuldade em gerar soluções bem adaptadas. Outra constatação importante é que sem funções de domínio, o conjunto de terminais seria maior pois deveria possuir todos os fonemas representando as nasalizações e todas as vogais a serem usadas nas comparações.

Partindo destas conclusões, foram acrescidas funções criadas a partir de uma análise das regras a serem implementadas. Podemos classificar as funções em dois grupos, as que devem ser utilizadas como condição e as que retornam valores para composição da resposta. No primeiro grupo estão FEHVOGAL, que recebe um argumento e retorna um valor lógico, indicando se o mesmo é ou não vogal e FEHMN cuja tarefa é retornar verdadeiro se o argumento recebido é uma das letras “m” ou “n”, além da função FEHTD utilizada no caso 1. No segundo grupo estão FNASAL que retorna o fonema correspondente à nasalização quando o argumento recebido for uma vogal e FRTDI, descrita em execuções anteriores.

O conjunto de terminais é formado pelas letras nas duas posições anteriores (CPOSAN1 e CPOSAN2) e nas duas posteriores (CPOSPO1 e CPOSPO2), além da letra na posição atual (CPOSAT).

O fitness básico máximo que uma solução pode atingir é de 261 pontos, seguindo o princípio de cálculo da multiplicação de cada fonema esperado por 3. O fitness padronizado do melhor indivíduo, encontrado na geração 165, foi 16 (fig.6.5).

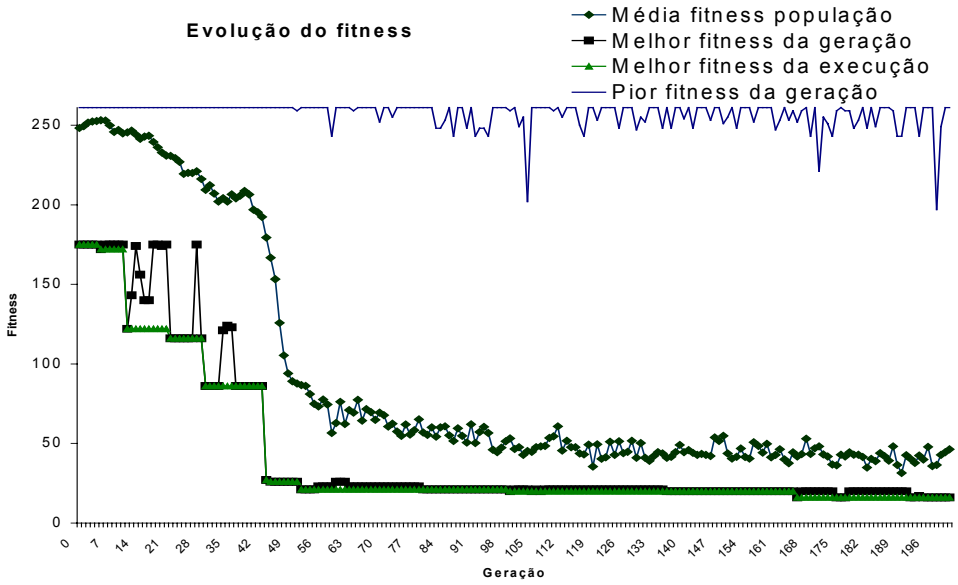


FIGURA 6.5 - Evolução do fitness para o caso 2

O código do melhor indivíduo foi:

(IF (IF (AND (CPOSPO1) (IF (IF (IF (CPOSPO2) (CPOSAN1) (AND (CPOSPO1) (IF (CPOSAT) (CPOSAT) (CPOSAT)) T6)) (FEHVOGAL (CPOSAN2)) (CPOSAN1)) (IF (FEHVOGAL (CPOSAN1)) (CPOSAN1) (CPOSAN2)) (IF (CPOSPO2) (CPOSAN1) (CPOSAN1))) T6) (IF (CPOSPO2) (FEHVOGAL (CPOSAN2)) (CPOSAN1)) (CPOSAT)) (IF (AND (FEHVOGAL (CPOSAN2)) (FEHVOGAL (CPOSAN2)) (CPOSAN1)) (CPOSAT) (IF (AND (FEHVOGAL (CPOSAT)) (IF (CPOSPO2) (CPOSAN1) (CPOSAN1)) (AND (CPOSPO1) (CPOSAT) T6)) (CPOSAN2) (IF (FEHVOGAL (CPOSPO1)) (CPOSAT) (FNASAL (CPOSAT))))) (IF (AND (IF (CPOSAN2) (CPOSAN1) (IF (CPOSAT) (FEHVOGAL (IF (CPOSPO2) (CPOSAN1) (FNASAL (AND (CPOSAT) (IF (CPOSPO2) (CPOSAN1) (CPOSAN1)) (CPOSAN1))))) (CPOSAN1))) (IF (AND (FEHVOGAL (CPOSAT)) (CPOSAT) (CPOSAN1)) (AND (FEHMN (CPOSPO1)) (CPOSAN1) (CPOSAN1)) (CPOSAN1)) (CPOSAT)) (CPOSAN2) (CPOSAN1)))

O indivíduo descrito acima obtém um bom fitness e constitui-se portanto, em uma solução aceitável para o problema. As regras implementadas são:

Se a letra na posição anterior não é nula e a letra na segunda posição anterior é vogal
Retorna a letra na posição atual
Senão
Se constante nula (não executa condição verdadeira)
Senão
Se letra na posição posterior é vogal
Retorna a letra na posição atual
Senão
Retorna nasalização da letra na posição atual

A primeira questão a ser ressaltada é que a maior parte das instruções observadas não realizam nenhuma atividade, essencialmente porque retornam valores sempre verdadeiros ou nulos, independente dos parâmetros de entrada. Este é o caso dos seguintes conjuntos de instruções:

(IF (AND (CPOSPO1) (IF (IF (IF (CPOSPO2) (CPOSAN1) (AND (CPOSPO1) (IF (CPOSAT) (CPOSAT) (CPOSAT)) T6)) (FEHVOGAL (CPOSAN2)) (CPOSAN1)) (IF (FEHVOGAL (CPOSAN1)) (CPOSAN1) (CPOSAN2)) (IF (CPOSPO2) (CPOSAN1) (CPOSAN1))) T6)

(AND (FEHVOGAL (CPOSAT)) (IF (CPOSPO2) (CPOSAN1) (CPOSAN1)) (AND (CPOSPO1) (CPOSAT) T6))

Nestes dois exemplos, o retorno será sempre nulo porque é executada uma função AND sobre três parâmetros, sendo que o último (T6) é um terminal que tem valor nulo fixo. A primeira das duas partes descritas é a condição do segundo IF da solução completa. Isto faz com que a primeira parte referente a condição verdadeira nunca seja executada. A função portanto retornará sempre a letra na posição atual (CPOSAT).

O código que efetivamente produz as respostas é:

(IF (AND (FEHVOGAL (CPOSAN2)) (FEHVOGAL (CPOSAN2)) (CPOSAN1)) (CPOSAT) (IF (AND (FEHVOGAL (CPOSAT)) (IF (CPOSPO2) (CPOSAN1) (CPOSAN1)) (AND (CPOSPO1) (CPOSAT) T6)) (CPOSAN2) (IF (FEHVOGAL (CPOSPO1)) (CPOSAT) (FNASAL (CPOSAT)))))

Nos casos que a posição atual é uma consoante, a sua nasalização é um valor nulo. Valores nulos não são inseridos na resposta final pela função de avaliação do fitness.

O melhor indivíduo da geração em determinadas momentos apresenta fitness pior do que o melhor da execução. Embora tal fato represente um decréscimo de adaptação da melhor solução, esta variação pode levar a combinações diferentes, que em gerações seguintes poderão resultar em soluções melhores.

Observando a fig. 6.5, é possível constatar que o pior fitness permanece no limite mais baixo possível durante muitas gerações, passando a variar de forma mais acentuada quando melhores respostas passam a ser obtidas. Durante a maior parte da evolução entretanto, existe algum indivíduo que não produz nenhum fonema correto.

Um exemplo é o indivíduo abaixo, encontrado na geração 180, que produz sempre um valor nulo.

(AND (CPOSPO1) (FEHVOGAL (CPOSAN2)) T6)

É possível perceber que a complexidade média aumenta progressivamente enquanto complexidades do melhor e do pior indivíduos oscilam. Valores relativos ao pior indivíduo, tendem a variar mais fortemente no início e diminuir a partir de gerações seguintes. O melhor indivíduo, embora apresente variações no número de pontos tende a possuir também uma complexidade crescente assim como a média da população (fig.6.6). A complexidade dos indivíduos neste subconjunto tende a crescer mais do que no caso anterior.

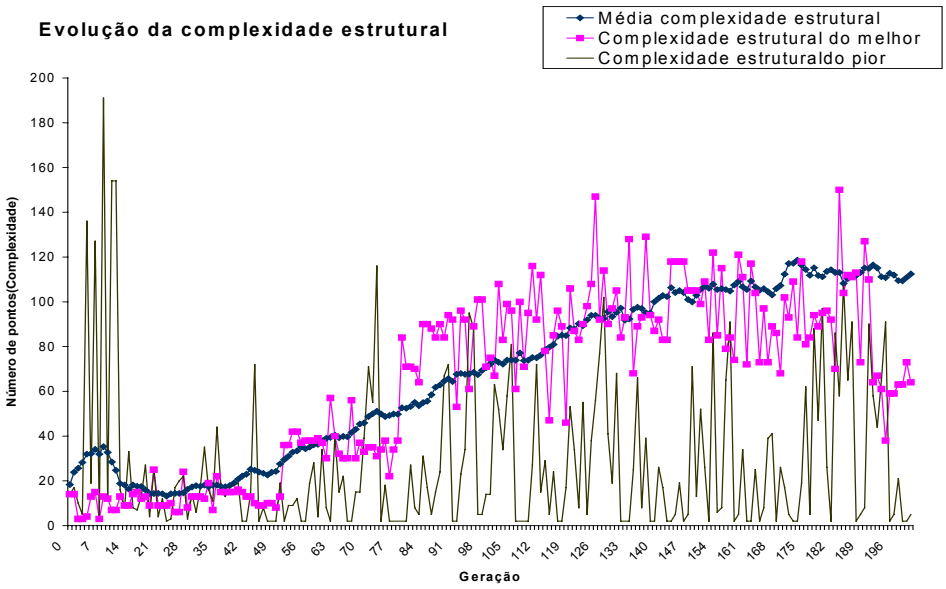


FIGURA 6.6 - Gráfico da complexidade estrutural para o caso 2

A complexidade estrutural do melhor indivíduo foi de 97 pontos. O número de pontos dos indivíduos neste caso apresenta valores superiores àqueles verificados no caso anterior. O aumento da complexidade das regras normalmente implica em soluções maiores, uma vez que a técnica tende a buscar melhores soluções através de combinações exaustivas.

Embora bem adaptada aos casos de fitness usados para o treinamento, a solução encontrada apresenta alguns problemas quando validada com outras palavras. Percebe-se na tabela abaixo que nas 30 palavras utilizadas na validação, a solução apresentou na maioria delas a produção de nasalização de vogais em final de palavra ou mesmo antes de consoantes diferentes de “m” ou “n” (tab.6.7).

A regra implementada não avalia se a letra após uma vogal é efetivamente “m” ou “n” e falha exatamente nesta situação. O fitness obtido na validação é de 127, dividindo-se este valor pelo total possível, que é 462, obtém-se uma corretude aproximada de 73%.

TABELA 6.7 – Validação da melhor solução para o caso 2.

Palavras usadas	Resultados esperados	Fonemas produzidos
((a p e n a d o) (t e n t a) (m a n a d a) (a n t e n a d o) (a m i d o) (d i n d o) (t i n t a) (m i n a d a) (a d m i t e) (b a n d o) (a b a t i d o) (p o n t o) (b o n d e) (t a m p a) (t o m a) (d o m a d a) (m u n d o) (t e m p o) (p i m e n t a) (t e m i d o) (p a t i n a) (a d i d o) (t u m b a) (d i n a) (b e t u m e) (a m a d a) (t e m a) (a b a d i a) (d a n a d o) (t o b a t a))	((a p e n a d o) (t e~ t a) (m a n a d a) (ax~ t e n a d o) (a m i d o) (dz i~ d o) (ts i~ t a) (m i n a d a) (a d m i t e) (b ax~ d o) (a b a ts i d o) (p o~ t o) (b o~ d e) (t ax~ p a) (t o m a) (d o m a d a) (m u~ d o) (t e~ p o) (p i m e~ t a) (t e m i d o) (p a ts i n a) (a dz i d o) (t u~ b a) (dz i n a) (b e t u m e) (a m a d a) (t e m a) (a b a dz i a) (d a n a d o) (t o b a t a))	((ax~ p e n a d o) (t e~ t ax~) (m ax~ n a d a) (ax~ t e~ n a d o) (ax~ m i d o) (d i~ d o~) (t i~ t ax~) (m i~ n a d a) (ax~ m i~ t e) (b ax~ d o~) (ax~ b a t i d o) (p o~ t o~) (b o~ d e~) (t ax~ p ax~) (t o~ m a) (d o~ m a d a) (m u~ d o~) (t e~ p o~) (p i~ m e t ax~) (t e~ m i d o) (p ax~ t i n a) (ax~ d i d o) (t u~ b ax~) (d i~ n a) (b e~ t u m e) (ax~ m a d a) (t e~ m a) (ax~ b a d i ax~) (d ax~ n a d o) (t o~ b a t a))

6.3 Caso3 - Combinação "rr"

Neste caso, o conjunto de regras a se obter envolve os seguintes contextos:

- relações direta entre letra e fone;
- ocorrência especial das letras "t" e "d" antes de "i" conforme o caso inicial e;
- casos da letra "r". Quando ocorrer duas vezes seguida a letra "r" representa o fonema x, nas demais situações a representação é a própria letra.

O conjunto das regras neste caso tem uma complexidade menor do que no caso anterior, porém representa uma situação diferenciada na qual uma mesma letra pode representar mais de um fonema quando ocorrer duas vezes em seqüência. De forma análoga ao caso anterior a segunda ocorrência da letra não deverá representar fonema algum. As definições para este caso são resumidas na tab. 6.8.

TABELA 6.8 - Definições para o caso 3

Parâmetro	Definição para o problema
Conjunto de funções	(EQ IF IF AND AND NOT FEHTD FRTDI)
Número de argumentos das funções	(2 2 3 2 3 1 2 1)
Conjunto de Terminais	((CPOSAT) (CPOSPO1) (CPOSAN1) (CPOSAN2) (CPOSPO2) r x NIL)
Casos de fitness	((a r r a) (e r r e) (i r r i) (o r r o) (a r t i) (o r d i) (o r p o) (u r b a) (o r a r) (t o r r e) (t u r r a))
Resposta esperada	((a x a) (e x e) (i x i) (o x o) (a r ts i) (o r dz i) (o r p o) (u r b a) (o r a r) (t o x e) (t u x a))
Número de gerações	G = 100
Tamanho da população	M = 1,000

As únicas funções de domínio utilizadas foram as mesmas empregadas no caso 1. Por se tratar de um conjunto de regras mais simples, funções de domínio foram utilizadas em menor escala, será possível avaliar novamente a capacidade do sistema em derivar regras a partir de terminais e funções básicas da linguagem.

O conjunto de terminais, é composto por funções representando as letras nas posições anteriores e posteriores, pela constante NIL, pela letra "r" e pelo fonema x.

O fitness básico máximo que uma solução pode atingir é de 120 pontos. O fitness padronizado do melhor indivíduo, encontrado na geração 81, foi 6. A fig. 6.7 demonstra a evolução das soluções ao longo do processo.

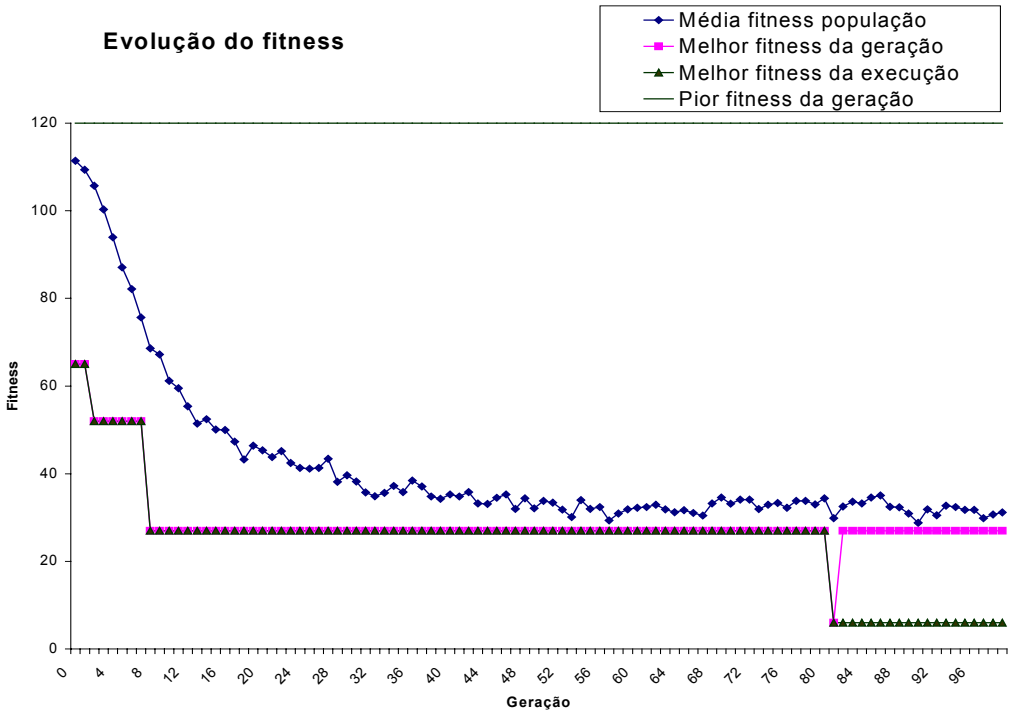


FIGURA 6.7 - Gráfico da evolução do fitness para o caso 3

Nesta execução, o fitness da melhor solução é encontrado na geração 81 e não é mais obtido nas gerações seguintes. Para este subconjunto de regras, foram realizadas além desta, outras duas execuções, nelas o melhor fitness permanece até o final ou por algumas gerações. O código da solução mais adaptada foi:

(IF (IF (FEHTD (CPOSAT) (IF (IF (CPOSAT) (CPOSAT)) (IF (IF (CPOSPO1) T8 (CPOSAN1)) T7 (CPOSAT)))) (AND (IF (EQ (CPOSAT) (IF (CPOSPO1) (CPOSAT) (CPOSAT))) (FEHTD (CPOSAT) (CPOSAT))) (FEHTD (CPOSAT) (IF (CPOSAT) (IF (IF (FEHTD (CPOSAT) (CPOSAT)) (FEHTD (CPOSAT) (CPOSPO1)) (CPOSAT)) (NOT T7) (AND (CPOSAT) (CPOSAN1) (CPOSAT)))))) (CPOSAT)) (IF (NOT (EQ (CPOSAT) (CPOSPO1))) (IF (EQ (CPOSAN1) (CPOSAT)) (IF (CPOSPO1) T7 (CPOSAT)) (CPOSAT))))

A maior parte do código, a exemplo das soluções anteriores é desnecessária. Um exemplo é a condição do IF inicial que resultará sempre em um valor verdadeiro. Segue abaixo o resumo das regras descobertas na solução.

Se a letra na posição atual é igual a letra na posição posterior
Retorna valor nulo
Senão
Se a letra na posição anterior é igual a letra na posição atual e a letra na posição posterior não é um valor nulo
Retorna o fonema x (T7)
Senão
Retorna a letra na posição atual

A solução implementa apenas regra correspondente a ocorrência de "rr" e a regra geral. O caso das letras "t" e "d" não é resolvido corretamente. Outras soluções encontradas durante a execução implementam estas últimas regras, mas falham na ocorrência "rr". Um exemplo é a solução abaixo, encontrada na geração 79. Neste indivíduo a combinação de "rr" leva a produção do fonema r, incorretamente.

(IF (IF (FEHTD (CPOSPO1) (CPOSPO1)) (AND (IF T8 (CPOSAN1) (IF (IF (IF (CPOSAT) (IF (EQ (CPOSAT) (CPOSAT)) (FRTDI (CPOSAT)) (CPOSAT))) (CPOSAT) (FRTDI (CPOSAT))) (EQ (CPOSAT) (CPOSPO1)) (CPOSAT))) (IF (IF (CPOSPO1) (CPOSAN1) (CPOSAT)) (CPOSAT) (IF (CPOSAT) (IF (IF (CPOSAT) (FEHTD (CPOSAT) (EQ (CPOSAN1) (CPOSAT))) (CPOSAT)) (CPOSAT))))) (CPOSAT)) (IF (NOT (EQ (CPOSAT) (CPOSPO1))) (IF (FEHTD (CPOSAT) (CPOSPO1)) (FRTDI (CPOSAT)) (CPOSAT))))

A complexidade estrutural do melhor indivíduo descrito foi de 60 pontos, porém é variável durante a maior parte do processo. A média da complexidade estrutural da população tende a manter-se em valores próximos a complexidade do melhor indivíduo a partir da metade do total de gerações.

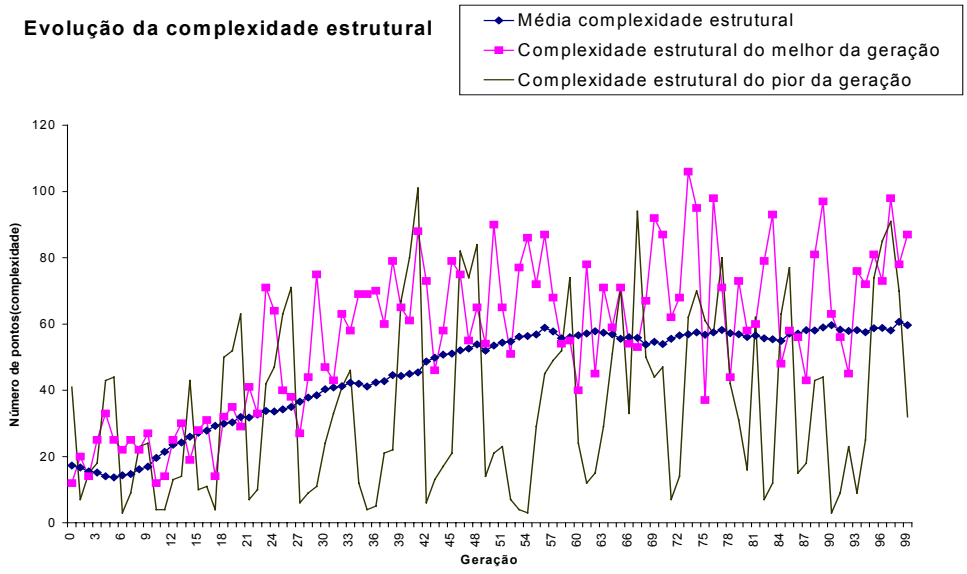


FIGURA 6.8 - Gráfico da complexidade estrutural para o caso 3

Os resultados do processo de validação mostrados abaixo confirmam a análise do código do melhor indivíduo. O fitness obtido na validação é 12, representando um acerto aproximado de 91%, relativo ao total do fitness da validação que é 144 (tab.6.9).

TABELA 6.9 - Resultados da validação da melhor solução para o caso 3

Palavras	Respostas esperadas	Fonemas produzidos
((t a r r o) (b r i t a) (a r r i b a) (o t o r r i n o) (p i r o u) (p a r a r) (a t i r a) (a d i r) (t i r a) (d i t o))	((t a x o) (b r i t a) (a x i b a) (o t o x i n o) (p i r o u) (p a r a r) (a ts i r a) (a dz i r) (ts i r a) (dz i t o))	((t a x o) (b r i t a) (a x i b a) (o t o x i n o) (p i r o u) (p a r a r) (a t i r a) (a d i r) (t i r a) (d i t o))

6.4 Caso 4 - Regras envolvendo as letras "c", "s" e "z"

O último subconjunto de regras possui a maior complexidade a ser enfrentada pela técnica. As regras a serem implementadas são as seguintes:

- relação direta entre letra e fone, ocorrência especial das letras "t" e "d" antes de "i";
- a letra "c" quando utilizada antes das letras "e" ou "i" é representada pelo fonema s, nos demais casos pelo fonema k. Não há a representação do caso de "ch", portanto a regra foi ignorada neste subconjunto.
- a letra "s" representa o fonema z em duas situações, quando ocorrer entre duas vogais ou ainda quando utilizada antes de consoantes vozeadas ("d", "b", "m", "g", "n").
- o emprego de "ss" resulta na produção de apenas um fonema s.
- a letra "z" tem como conseqüência o fonema s quando ocorre após vogal em fim de palavra, antes de consoante. Sempre que vier seguida de uma vogal o fonema correspondente é a própria letra.

Neste subconjunto são apresentados diversos contextos verificados na língua portuguesa como: i) a produção de diferentes fonemas pela mesma letra; ii)um mesmo fonema sendo representado por letras diferentes e; iii) uma situação na qual o final de palavra influencia a letra anterior. As definições para o caso estão na tab. 6.10.

TABELA 6.10 - Definições para o caso 4

Parâmetro	Definição para o problema
Conjunto de funções	(EQ IF IF FEHSCVO FEHTD FRTDI FEHSS FEHVSV FEHVZNIL FEHCEI)
Número de argumentos das funções	(2 2 3 2 2 1 2 3 3 2)
Conjunto de terminais	((CPOSAT) (CPOSPO1) (CPOSAN1) c s z k NIL)
Casos de fitness	((c a s a) (c e s s a) (a c i) (a c e) (a s s o) (e c o) (a s a) (o s s o) (e s d) (a s b) (t i s m o) (t i s s o) (d i s s o) (d i c a) (z e r o) (c e r z o) (a z) (o z) (c e c i) (c i c e) (o s a) (i s o) (e s a) (a s m) (o s n) (i s b) (i z) (u z) (d i) (t i))
Resposta esperada	((k a z a) (s e s a) (a s i) (a s e) (a s o) (e k o) (a z a) (o s o) (e z d) (a z b) (ts i z m o) (ts i s o) (dz i s o) (dz i k a) (z e r o) (s e r z o) (a s) (o s) (s e s i) (s i s e) (o z a) (i z o) (e z a) (a z m) (o z n) (i z b) (i s) (u s) (dz i) (ts i))
Número de gerações	G = 100
Tamanho da população	M = 1,000

O conjunto de funções apresenta novas funções de domínio. A complexidade e o conjunto de regras envolvidas requer determinados detalhes que são implementados por estas funções. Cada função de domínio é orientada a uma regra específica; porém caberá ao processo descobrir como empregá-la corretamente e quais os argumentos significativos.

A função FEHSCVO recebe dois argumentos e retorna verdadeiro caso o primeiro argumento for a letra "s" e o segundo for uma consoante vozeada ("d", "b", "m", "g", "n"). A função FEHSS retorna verdadeiro somente se ambos os argumentos recebidos forem a letra "s". FEHVSV recebe três argumentos e verifica se a primeira e terceira letras são vogais e a segunda é a letra "s", retornando verdadeiro em caso positivo. A função FEHVZNIL recebe igualmente três argumentos e testa se a primeira letra é vogal, a segunda a letra "z" e a terceira possui valor nulo. Por fim a função FEHCEI recebe dois argumentos e testa se o primeiro é a letra "c" e o segundo é "i" ou "e". Em todas as funções, o valor retornado caso os testes não sejam verdadeiros é a constante NIL.

O conjunto de terminais é composto pelas letras na posição atual, posterior e anterior. Não é exigido em nenhuma das regras a avaliação de uma letra na segunda posição anterior ou posterior; portanto a suficiência é mantida mesmo retirando-se estas entradas. Os demais componentes são as letras e fonemas envolvidos nas regras.

O fitness básico máximo que uma solução pode atingir é de 288 pontos. O fitness padronizado que orienta o processo é calculado subtraindo-se o fitness individual da solução de 288. A definição dos casos de fitness neste conjunto é especialmente delicada. Como o conjunto de regras a implementar é mais amplo, deve-se ter um cuidado extremo para que todas as regras possuam uma amostragem satisfatória sem que o conjunto de casos torne-se excessivamente extenso.

O fitness padronizado do melhor indivíduo, encontrado na geração 60, foi 30. A fig. 6.9 demonstra a evolução das soluções para o caso.

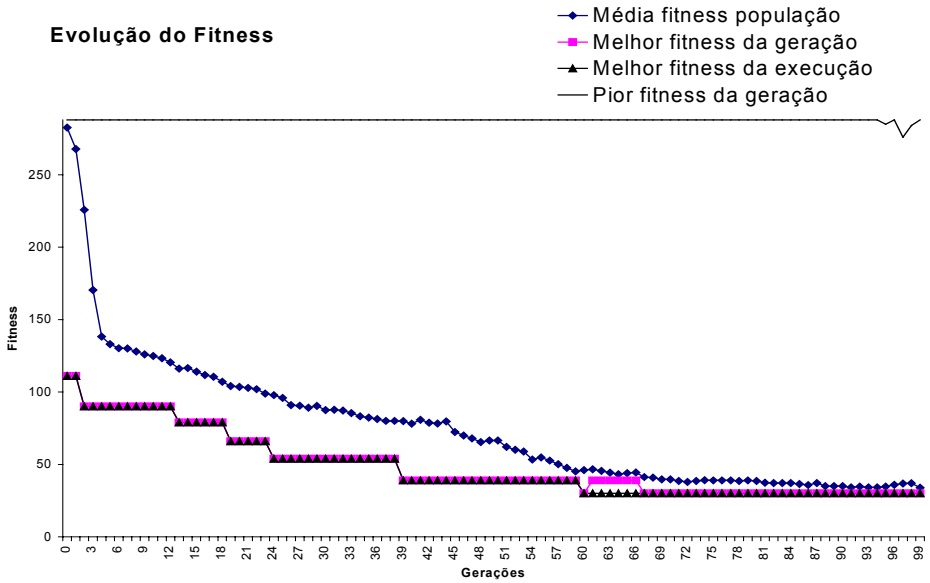


FIGURA 6.9 - Gráfico da evolução do fitness para o caso 4

É importante observar que algumas das funções de domínio foram ignoradas pela melhor solução, entretanto em algumas soluções bem adaptadas tanto nesta quanto em outras execuções elas foram utilizadas. O código da melhor solução é:

(IF (IF (EQ (IF T4 (CPOSAT) T5) T4) T6) (IF (FEHCEI (CPOSAT) (CPOSPO1)) T5 (IF (IF T4 (IF (IF T4 (CPOSAT) (IF T6 T5)) (IF (IF (EQ (CPOSPO1) T5) (IF (EQ (CPOSPO1) T5) T6)) (CPOSAT) (IF T4 (FEHTD (CPOSAT) (CPOSPO1)) T7)) T7) (CPOSAN1)) (CPOSAT) T7)) (IF T4 (IF (IF (EQ (IF (IF (EQ (CPOSPO1) T5) T6) (FEHVZNIL T5 T5 T8) (IF T4 (CPOSAT) T4)) T5) T6) (IF (EQ (IF (IF (EQ (CPOSAN1) T5) T6) T4 (IF T4 (CPOSAT) T7)) T5) T6) (IF (CPOSAT) (CPOSAT) T7)) (FEHVZNIL (CPOSPO1) T4 (CPOSPO1))))

Este código resulta no seguinte conjunto de regras:

Se a letra na posição atual é igual a "c"
Se a letra na posição atual é igual a "c" e a letra na próxima posição é igual a "i" ou a letra na próxima posição é igual a "e"
Retorna s
Senão
Retorna k
Senão
Se a letra na posição atual é igual a "s" e as letras na posição anterior e na posição posterior forem diferentes de "s"
Retorna z
Senão
Se a letra na posição atual é igual a "s" e a letra na posição posterior é igual a "s"
Retorna s
Senão
Se a letra na posição posterior não é igual "s" e a posição anterior é igual a "s"
Retorna valor nulo
Senão
Retorna a letra na posição atual

A complexidade estrutural do melhor indivíduo, mostrada na fig. 6.10, foi de 89 pontos. A complexidade estrutural apresentada neste caso é em média maior do que nos casos anteriores. O indivíduo mais bem adaptado apresentou um número de pontos menor do que na solução verificada no caso 2; neste caso entretanto, foi obtido um indivíduo com adaptação maior. A tendência verificada é de um aumento constante, pois o sistema possui como resultado uma solução ainda distante da ideal e vai aumentando a complexidade dos indivíduos na tentativa de obter melhores valores de fitness.

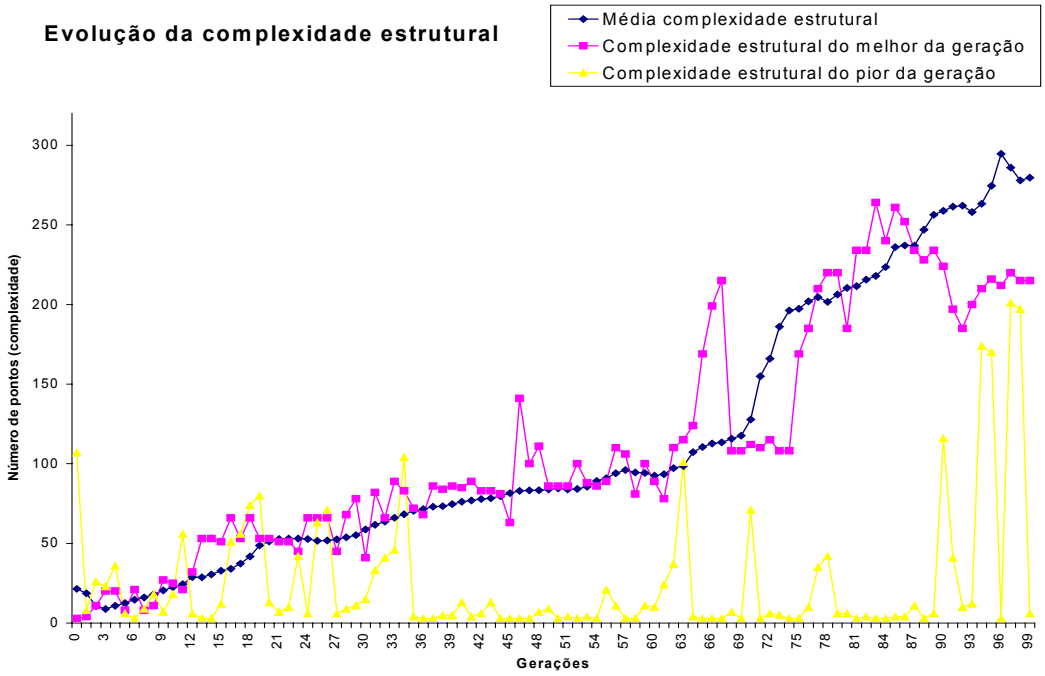


FIGURA 6.10 - Gráfico da complexidade estrutural para o caso 4

A validação do indivíduo apresenta um acerto aproximado de 90%, com um fitness padronizado igual a 18. O total do fitness para estas palavras é 189.

TABELA 6.11 - Resultados da validação da melhor solução para o caso 4

Palavras	Respostas esperadas	Fonemas produzidos
((c a c a) (c e c e) (c i c i) (c o c a) (a s s a) (i s s o) (e s s a) (a s a) (o s a) (u s e) (a s b) (e s d) (i s m) (t i t i) (d i d i) (z i z a) (z e z o) (a z) (u z))	((k a k a) (s e s e) (s i s i) (k o k a) (a s a) (i s o) (e s a) (a z a) (o z a) (u z e) (a z b) (e z d) (i z m) (ts i ts i) (dz i dz i) (z i z a) (z e z o) (a s) (u s))	((k a k a) (s e s e) (s i s i) (k o k a) (a s a) (i s o) (e s a) (a z a) (o z a) (u z e) (a z b) (e z d) (i z m) (t i t i) (d i d i) (z i z a) (z e z o) (a z) (u z))

6.4.1 Caso 4 - Funções sem argumentos

Até o presente momento todas as execuções utilizaram funções de domínio que operavam sobre argumentos recebidos. Os argumentos corretos deveriam ser selecionados pelo processo evolutivo; o recebimento de argumentos incorretos tornava as funções sem efeito ou produzia resultados insatisfatórios. A descoberta de argumentos corretos não é um processo trivial uma vez que existem inúmeras combinações possíveis.

TABELA 6.12-Definições para o caso 4 usando funções de domínio sem argumentos

Parâmetro	Definição para o problema
Conjunto de funções	(EQ IF IF FEHSS)
Número de argumentos das funções	(2 2 3 2)
Conjunto de terminais	((CPOSAT) (CPOSPO1) (CPOSAN1) (FEHCEI) (FEHSCVO) (FEHTD) (FRTDI) (FEHVSV) (FEHVZNIL) c s z k NIL)
Casos de fitness	((c a s a) (c e s s a) (a c i) (a c e) (a s s o) (e c o) (a s a) (o s s o) (e s d) (a s b) (t- i s m o) (t- i s s o) (d i s s o) (d i c a) (z e r o) (c e r z o) (a z) (o z) (c e c i) (c i c e) (o s a) (i s o) (e s a) (a s m) (o s n) (i s b) (i z) (u z) (d i) (t- i))
Resposta esperada	((k a z a) (s e s a) (a s i) (a s e) (a s o) (e k o) (a z a) (o s o) (e z d) (a z b) (ts i z m o) (ts i s o) (dz i s o) (dz i k a) (z e r o) (s e r z o) (a s) (o s) (s e s i) (s i s e) (o z a) (i z o) (e z a) (a z m) (o z n) (i z b) (i s) (u s) (dz i) (ts i))
Número de gerações	G = 100
Tamanho da população	M = 1,000

Nesta execução serão descritos resultados com as mesmas funções de domínio empregadas na execução anterior para o caso, porém estas funções farão parte do conjunto de terminais por não operarem sobre argumentos recebidos. Cada uma das funções irá operar diretamente sobre posição atual, anterior ou posterior (tab.6.12). Naturalmente a expectativa é que a produção de boas soluções seja bem mais simples devido ao fato de que as funções de domínio implementam partes das regras sem a dependência de entradas escolhidas no processo evolutivo. O Anexo 2 descreve o código das funções de domínio com e sem argumentos.

O tamanho do conjunto de funções é sensivelmente menor do que na execução anterior ao passo que o conjunto de terminais é maior. A função FEHSS foi mantida no conjunto de funções porque como testa se ambos os parâmetros são a letra "s"; o teste pode ocorrer na letra atual e na próxima ou ainda na anterior e na atual, o que torna a passagem de parâmetros corretos uma tarefa um pouco mais simples. A função foi mantida também para não haver uma redução extrema do conjunto de funções. As funções AND e OR foram retiradas do conjunto porque todos os testes com condições combinadas necessárias para a produção de soluções corretas são implementados pelas funções de domínio.

Igualmente à execução anterior, o fitness básico máximo é de 288 pontos. Todos os parâmetros restantes foram mantidos para permitir uma comparação na evolução das respostas produzidas a partir da alteração das funções de domínio.

O fitness padronizado do melhor indivíduo, encontrado na geração 35, foi 9. A fig. 6.11 demonstra a evolução das soluções.

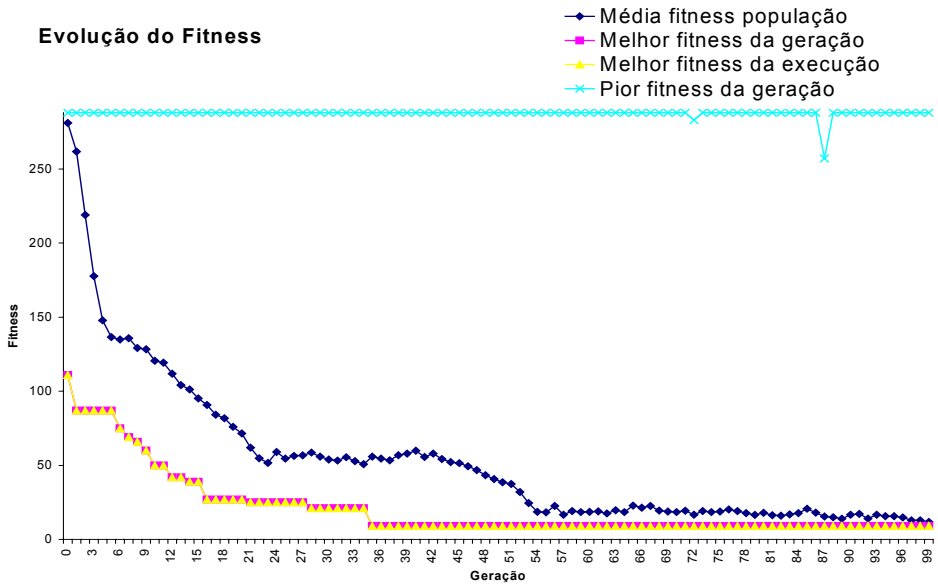


FIGURA 6.11 - Gráfico da evolução do fitness para o caso 4 usando funções de domínio sem argumentos

A hipótese de uma evolução mais rápida se confirma plenamente. Já na geração 35 foi encontrado um indivíduo com um grau de adaptação muito superior ao melhor encontrado na execução com funções que requerem argumentos. Embora tenha sido descoberta uma solução com alto grau de adaptação, no restante do processo a população apenas converge para o mínimo local representado por esta solução sem encontrar um valor ótimo para o fitness. Na maior parte das execuções realizadas com este conjunto de parâmetros, ocorre a convergência da média da população para o fitness da melhor solução. O pior indivíduo entretanto permanece nos piores valores de fitness possíveis, sempre existindo respostas que produzem valores nulos ou qualquer outro que não faça parte do conjunto de fonemas. O código da melhor solução obtida foi:

(IF (FEHSS (IF T13 (CPOSAT) (IF (IF (IF (IF (CPOSPO1) (CPOSAT) (IF (FEHCEI) T11 (CPOSAT))) T12 (IF T11 (IF (IF (FEHVZNIL) (CPOSAT) (FEHTD)) T11 (CPOSAT)) (FEHCEI))) (FEHVSV) (CPOSAT)) (IF (IF (FEHVZNIL) T11 (FEHSS (FRTDI) T13)) T11 (CPOSAT)))) (IF T14 (IF (FEHCEI) T11 T11) (CPOSAN1))) (FEHVZNIL) (IF (FEHCEI) T11 (IF (IF (FEHSCVO) T14 (IF (IF (IF (IF (FEHVZNIL) (FEHVZNIL) (FEHVZNIL)) T14 T14) (FEHVSV) (EQ (FEHVSV) (FEHSCVO))) (IF (FEHCEI) T11 (CPOSAT)))) (IF (FEHVZNIL) (IF T11 (IF (IF (IF (FEHCEI) (FEHVZNIL)) (IF T12 T13 T10)) T11 T11) (CPOSAN1)) (IF (IF (FEHVZNIL) (IF (FEHVZNIL) (FEHVSV) (CPOSAT)) (IF (FEHVZNIL) (CPOSAT) (FEHTD))) (IF T11 (FRTDI) T11) (IF (FEHVSV) T12 (IF T11 (IF (FEHCEI) T11 (CPOSAT)) (FEHCEI))))) (IF T11 T12 (CPOSAT)))))

As regras implementadas são:

Se a letra na posição atual é igual a "s" e a letra na próxima posição é "s"
Retorna valor nulo
Senão
Se a posição atual for igual a "c" e a letra na próxima posição for igual a "e" ou a próxima letra for igual a "i"
Retorna s
Senão
Se (a letra na posição atual é igual a "s" e a letra na próxima posição for igual a constante vozeada) ou (a letra na posição atual é "s" e a anterior e a posterior são vogais)
Retorna z
Senão
Se a letra na posição atual for a letra "z" e a letra na posição anterior é vogal e a próxima é nula
Retorna s
Senão
Se (a letra na posição atual é "t" ou "d") e a letra na próxima posição é "i"
Retorna ts ou dz de acordo com a letra na posição atual
Senão
Retorna a letra na posição atual

É possível perceber que a solução possui uma maior eficiência e melhor aplicação das funções e terminais. Existem poucas combinações de instruções sem efeito, fato verificado na maior parte das execuções anteriores.

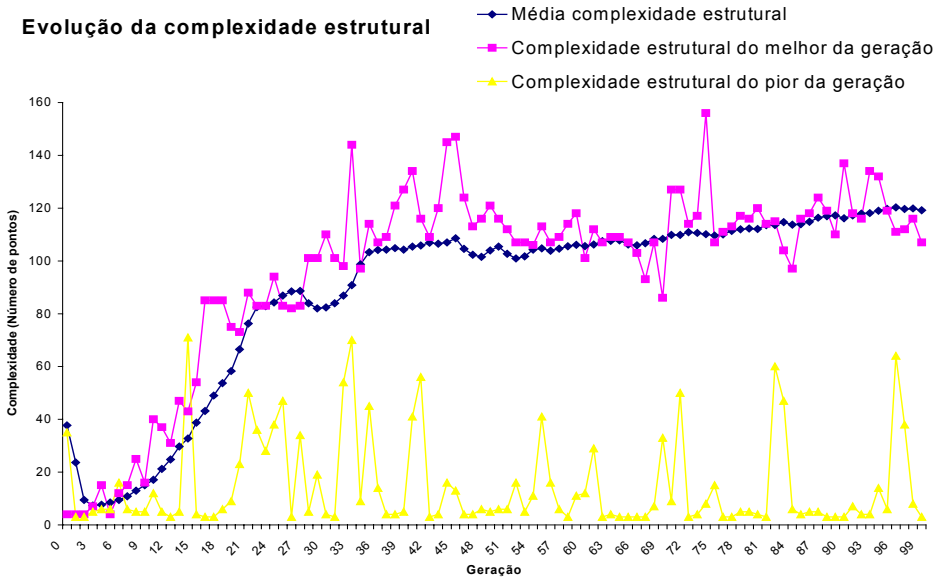


FIGURA 6.12 - Gráfico da complexidade estrutural para o caso 4 usando funções de domínio sem argumentos

A única regra não implementada de forma completa é a regra da letra "c". Quando a letra ocorrer antes de "e" ou "i", será produzido o fonema s, porém nos demais casos será produzida como resposta a própria letra, quando a resposta certa seria o fonema k.

A complexidade estrutural do melhor indivíduo descrito foi de 114 pontos, maior do que o melhor indivíduo apresentado na execução anterior. É necessário salientar que embora a evolução apresente rapidamente complexidades altas, existe uma tendência de estabilização ao longo da evolução. A complexidade do melhor indivíduo e da média da população tendem a manter-se entre 100 e 120 pontos, já a partir da geração 35 (fig.6.12). A utilização de funções sem argumentos tem uma influência direta na complexidade, uma vez que a utilização de funções como argumentos de outras funções é um fator que normalmente produz uma maior variabilidade no tamanho. Tanto na geração da população inicial quanto nas operações de cruzamento, a tendência é a produção de árvores com menor profundidade.

TABELA 6.13 - Resultados da validação da melhor solução para o caso 4

Palavras	Resultados esperados	Fonemas produzidos
((c a c a) (c e c e) (c i c i) (c o c a) (a s s a) (i s s o) (e s s a) (a s a) (o s a) (u s e) (a s b) (e s d) (i s m) (t- i t- i) (d i d i) (z i z a) (z e z o) (a z) (u z))	((k a k a) (s e s e) (s i s i) (k o k a) (a s a) (i s o) (e s a) (a z a) (o z a) (u z e) (a z b) (e z d) (i z m) (t i t i) (d i d i) (z i z a) (z e z o) (a s) (u s))	((c a c a) (s e s e) (s i s i) (c o c a) (a s a) (i s o) (e s a) (a z a) (o z a) (u z e) (a z b) (e z d) (i z m) (ts i ts i) (dz i dz i) (z i z a) (z e z o) (a s) (u s))

A validação do indivíduo apresenta um acerto aproximado de 94%. O fitness obtido nesta atividade foi de 12, sendo o fitness total de 189.

6.5 Resultados utilizando definição automática de funções (ADF)

Conforme descrito anteriormente, a definição automática de funções permite estender a Programação Genética, evoluindo além de programas principais, funções que poderão servir a estes programas. Com o objetivo de estabelecer uma comparação dos resultados obtidos com as execuções anteriores, serão descritas duas execuções, uma para o caso 1, de menor complexidade e outra para o caso 4, de maior complexidade. A partir destes dois subconjuntos será possível avaliar o potencial da extensão da técnica aplicada ao problema. Em ambos os casos serão adotadas duas funções automaticamente definidas, que poderão possuir diferentes conjuntos de terminais e funções. Cada uma destas funções chamadas respectivamente de ADF0 e ADF1 receberá três argumentos.

6.5.1 Caso 1 utilizando ADF

TABELA 6.14 - Definições para o caso 1, utilizando ADF:

Parâmetro	Definição para o problema
Conjunto de funções para o ramo de produção de resultados	(EQ IF IF AND OR ADF0 ADF1)
Número de argumentos das funções do ramo de produção de resultados	(2 2 3 2 2 3 3)
Conjunto de funções para as funções ADF0 e ADF1	(EQ IF IF AND OR)
Número de argumentos das funções do conjunto de funções disponíveis para ADF0 e ADF1	(2 2 3 2 2)
Conjunto de terminais	((CPOSAT) (CPOSPO1) t d i ts dz)
Casos de fitness	((p a) (b e) (v i) (t i) (d i) (t a t o) (d a d o) (d i d i) (d i t i))
Resposta esperada	((p a) (b e) (v i) (ts i) (dz i) (t a t o) (d a d o) (dz i dz i) (dz i ts i))
Número de gerações	G = 100
Tamanho da população	M = 1,000

Conforme descrito anteriormente, o caso 1 envolve relação direta entre letra e fonema e o caso das letras “t” e “d” antes de “i”. Parte das especificações para aplicação da programação genética usando definição automática de funções são iguais às definidas na aplicação sem ADF. O conjunto de funções e terminais é o mesmo tanto para o ramo de produção de resultados quanto para cada uma das funções automaticamente definidas, exceção feita ao conjunto de funções do ramo de produção de resultados que além das funções anteriormente disponíveis possui também disponíveis as funções ADF0 e ADF1, que possuem cada uma 3 argumentos.

Não foram utilizadas funções de domínio, tal estratégia visa estabelecer comparações entre a inserção de funções diretamente nas definições para o problema ou a evolução de funções juntamente com um programa principal. A fig. 6.13 demonstra a evolução desta execução, que apresentou resultados semelhantes aos obtidos sem ADF.

O melhor indivíduo foi obtido na geração 89 e apresentou um fitness igual a 5. O código da melhor solução foi:

```
(progn (defun ADF0 (ARG0 ARG1 ARG2)
(values (IF T3_ADF0 (IF ARG2 ARG1 T5_ADF0))))
(defun ADF1 (ARG0 ARG1 ARG2)
(values (IF (IF (IF ARG0 (IF (EQ ARG0 (OR T4_ADF1
T4_ADF1)) T5_ADF1 (CPOSAT))) (EQ (CPOSPO1) (IF
(EQ ARG0 (OR (AND ARG2 ARG1) T4_ADF1)) T5_ADF1
(IF (EQ ARG0 (OR T4_ADF1 T4_ADF1)) T5_ADF1
(CPOSAT))))) (OR T7_ADF1 (OR T7_ADF1 T5_ADF1))
(CPOSAT))))
(values (ADF1 (CPOSAT) T3 (CPOSPO1))))
```

As regras para conversão estão em sua totalidade implementadas na função ADF1. A lógica da solução envolve diversos testes aninhados, porém pode ser resumida de forma relativamente simples:

Se o primeiro argumento(ARG0) é nulo
Retorna a letra na posição atual
Senão
Se (o primeiro argumento é igual a "d" e a letra na próxima posição é igual a "i") ou (a letra na posição atual é igual a letra na próxima posição)
Retorna dz
Senão
Retorna a letra na posição atual

A regra relativa às letras "t" e "d" é implementada apenas parcialmente. O fonema dz é produzido de forma incorreta, caso a mesma letra ocorra de forma seguida; tal ocorrência não é verificada nos casos de treinamento, sendo assim esta implementação incorreta não diminui o valor do fitness no processo evolutivo. A chamada ocorre com o primeiro argumento sendo a letra da posição atual, o segundo o a letra "t" e o terceiro, a letra da próxima posição.

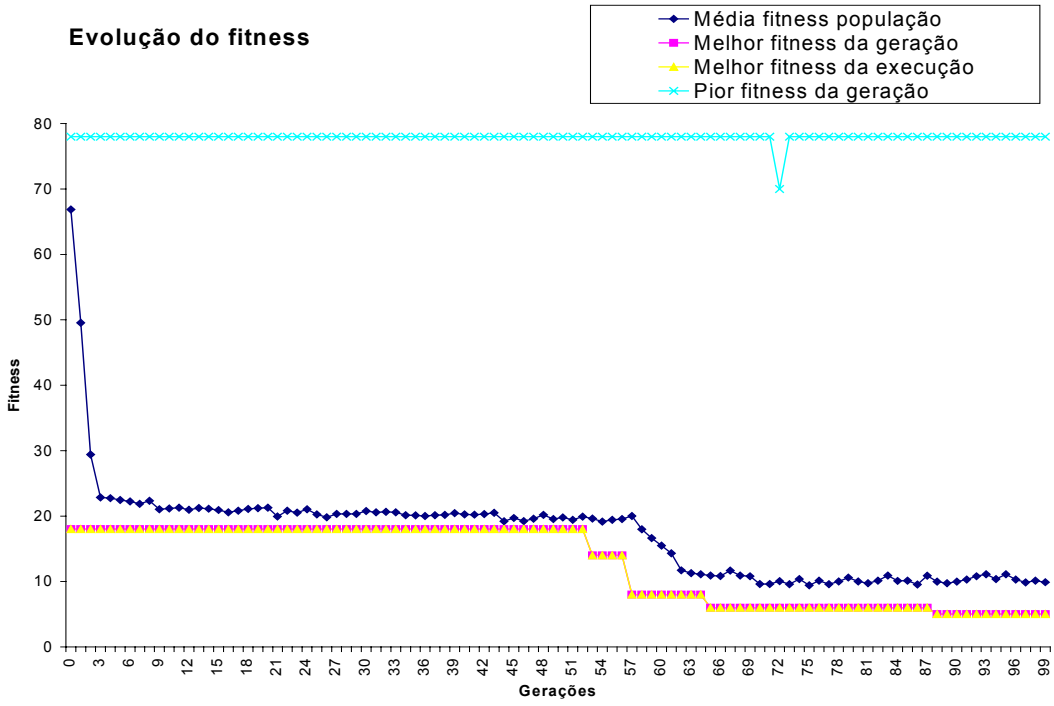


FIGURA 6.13 Gráfico da evolução do fitness para o caso 1, usando ADF.

A complexidade estrutural apresentou valores inferiores aos verificados na execução do caso 1 sem funções de domínio e sem definição automática de funções. O melhor indivíduo obtido nesta execução apresentou complexidade de 46 pontos, inferior ao melhor indivíduo na referida execução, que apresentou 92 pontos. A média da complexidade dos indivíduos nesta execução atinge valores levemente superiores a 100 pontos; mas a partir da geração 50 aproximadamente, os valores passam a diminuir, estabilizando em valores próximos ao número de pontos do melhor indivíduo. A fig. 6.14 mostra a evolução da complexidade estrutural.

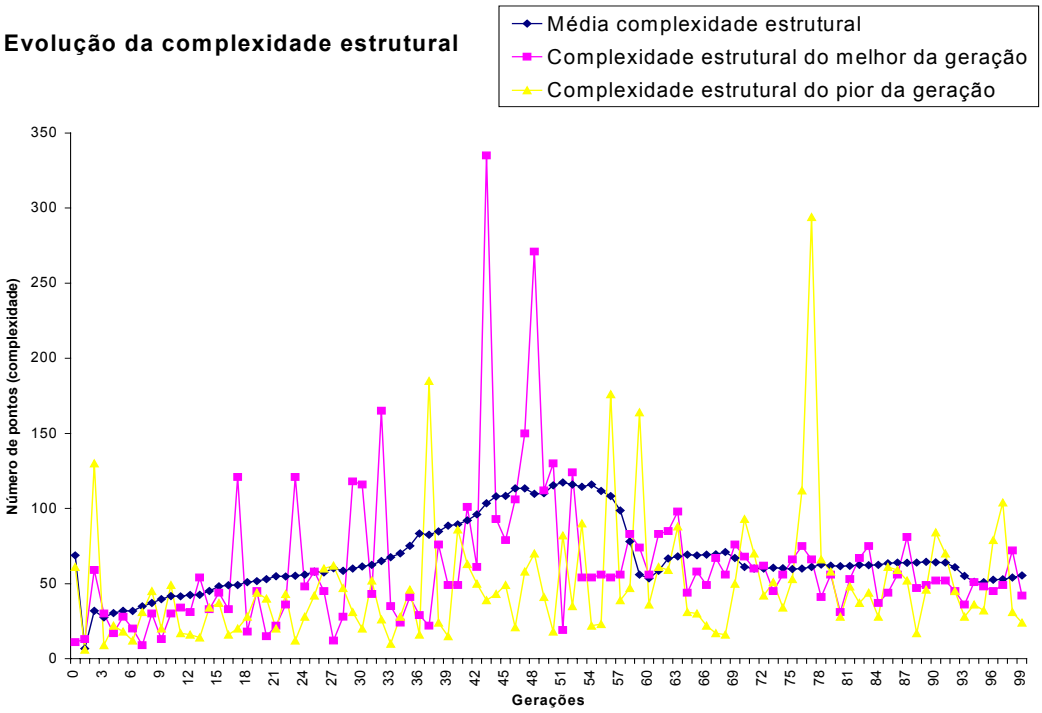


FIGURA 6.14 – Evolução da complexidade estrutural para o caso 1, usando ADF.

A validação para a solução, está descrita abaixo. O fitness padronizado da melhor solução foi 3, de um fitness básico total igual a 126. O percentual de acerto é superior a 97%, semelhante ao verificado na validação da solução do caso 1 sem ADF (tab.6.15).

TABELA 6.15 - Resultados da validação para o caso 1, usando ADF.

Palavras usadas	Resultados esperados	Resultados produzidos
((p a t o) (b e b a) (v i v e) (t i p o) (d i t o) (d e d o) (t u d o) (b a t a t a) (v i d a) (f a d a))	((p a t o) (b e b a) (v i v e) (ts i p o) (dz i t o) (d e d o) (t u d o) (b a t a t a) (v i d a) (f a d a))	((p a t o) (b e b a) (v i v e) (t i p o) (dz i t o) (d e d o) (t u d o) (b a t a t a) (v i d a) (f a d a))

6.5.2 Caso 4 utilizando ADF

O caso 4 apresenta uma maior complexidade do que o caso 1, através dos resultados obtidos neste subconjunto foi possível avaliar mais detalhadamente os benefícios da definição automática de funções. As definições para a execução estão na tab. 6.16.

TABELA 6.16 – Definições para o caso 4, usando ADF

Parâmetro	Definição para o problema
Conjunto de funções para o ramo de produção de resultados	(EQ IF IF AND FEHVOGAL NOT FEHSCVO ADF0 ADF1)
Número de argumentos das funções do ramo de produção de resultados	(2 2 3 2 1 1 2 3 3)
Conjunto de funções para as funções ADF0 e ADF1	(EQ IF IF AND FEHVOGAL NOT FEHSCVO)
Número de argumentos das funções do conjunto de funções disponíveis para ADF0 e ADF1	(2 2 3 2 1 1 2)
Conjunto de terminais	((CPOSAT) (CPOSPO2) (CPOSPO1) (CPOSAN1) (CPOSAN2) c s z k NIL t d ts dz)
Casos de fitness	((c a s a) (c e s s a) (a c i) (a c e) (a s s o) (e c o) (a s a) (o s s o) (e s d) (a s b) (t i s m o) (t i s s o) (d i s s o) (d i c a) (z e r o) (c e r z o) (a z) (o z) (c e c i) (c i c e) (o s a) (i s o) (e s a) (a s m) (o s n) (i s b) (i z) (u z) (d i) (t- i))
Resposta esperada	((k a z a) (s e s a) (a s i) (a s e) (a s o) (e k o) (a z a) (o s o) (e z d) (a z b) (ts i z m o) (ts i s o) (dz i s o) (dz i k a) (z e r o) (s e r z o) (a s) (o s) (s e s i) (s i s e) (o z a) (i z o) (e z a) (a z m) (o z n) (i z b) (i s) (u s) (dz i) (ts i))
Número de gerações	G = 100
Tamanho da população	M = 1,000

Para esta execução, o conjunto de terminais é o mesmo para o ramo de produção de resultados e para as funções automaticamente definidas. O conjunto de funções difere apenas para o ramo de produção de resultados, no qual encontram-se também as funções ADF0 e ADF1. As únicas funções de domínio utilizadas são FEHVOGAL, mantida devido ao alto grau de detalhe na implementação necessária para verificar se uma determinada letra é vogal e FEHSCVO pela complexidade da regra que exige um teste de ocorrência da letra "s" antes de uma consoante vozeada. Sem funções de domínio, a complexidade para implementação das regras tende a ser maior, o que permitirá novamente comparar a alternativa ADF com o uso das funções de domínio.

A variação do fitness demonstrada na fig. 6.15 caracteriza uma evolução mais gradativa e continuada do que as execuções anteriormente descritas. O melhor fitness da geração e execução permanecem o mesmo, todo o tempo e o fitness médio apresenta um decréscimo constante. As variações diminuem sensivelmente a partir da geração 80. A melhor solução alcançou um fitness de 31 e foi obtida na geração 85. Soluções com fitness de 32 pontos já são encontradas a partir da geração 52 e são praticamente equivalentes à mais adaptada.

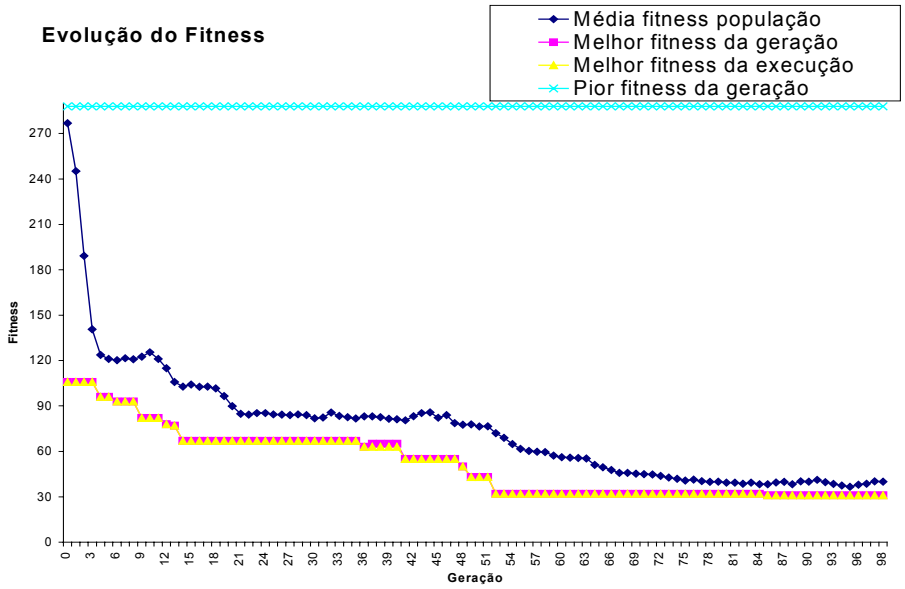


FIGURA 6.15 - Gráfico da evolução do fitness para o caso 4, usando ADF.

O código do melhor indivíduo foi:

```
(progn
(defun ADF0 (ARG0 ARG1 ARG2)
(values (IF (FEHVOGAL (CPOSAN1)) (IF (EQ ARG1
(CPOSAT)) (EQ T7_ADF0 T10_ADF0) (IF (IF (CPOSPO1)
(EQ T6_ADF0 (CPOSAT)) T7_ADF0) T7_ADF0 ARG2)) (IF
(FEHVOGAL (IF (EQ ARG1 T9_ADF0) (EQ T6_ADF0
T10_ADF0) T7_ADF0)) (IF (EQ ARG1 (EQ ARG1 (EQ
ARG1 (FEHVOGAL T14_ADF0)))) (EQ T6_ADF0 (EQ
ARG1 (EQ T6_ADF0 T7_ADF0))) ARG2) (IF (EQ T6_ADF0
(CPOSAT)) (IF (EQ T7_ADF0 (EQ T6_ADF0 ARG1)) (EQ
T6_ADF0 T10_ADF0) T7_ADF0) (IF (EQ T11_ADF0
(CPOSAT)) T14_ADF0 (CPOSAT)))))))
(defun ADF1 (ARG0 ARG1 ARG2)
(values (IF (CPOSAT) T10_ADF1 T6_ADF1)))
(values (ADF0 T8 (CPOSPO1) (ADF0 T8 (CPOSPO2)
T8))))
```

O fitness representa um acerto de aproximadamente 90%. As regras são implementadas totalmente pela função ADF0 e podem ser resumidas nas seguintes atividades:

Se a letra na posição anterior é vogal
Se a letra na posição atual é igual ao segundo argumento(ARG1)
Retorna um valor nulo
Senão
Se a letra na posição posterior é nula ou a letra na posição atual é "c"
Retorna s
Senão
Retorna o terceiro argumento
Senão
Se a letra na posição atual é igual a "c"
Retorna s
Senão
Se a letra na posição atual é igual a "t"
Retorna dz
Senão
Retorna a letra na posição atual

O ramo de produção de resultados faz duas chamadas à função ADF0, (ADF0 T8 (CPOSPO1) e (ADF0 T8 (CPOSPO2) T8). A chamada interna retorna um valor que constitui-se no terceiro parâmetro da chamada que vai efetivamente produzir o resultado e como o segundo argumento passado nesta última referência é a posição posterior a regra de "ss" funciona corretamente. Outra questão importante é que o segundo argumento será "z" nas situações em que ocorre uma letra "s" antes de consoante vozeada e após vogal, sendo assim o retorno da função será exatamente este argumento.

Para melhor compreensão da aplicação das regras, três exemplos de chamadas do ramo de produção de resultados, estão descritos abaixo. O resultado da primeira aplicação de ADF0 é o terceiro argumento da segunda chamada.

(casa) = adf0 z s z = s => adf0 z a s = s
adf0 z a z = a => adf0 z s a = a
adf0 z nil z = z => adf0 z a z = z
adf0 z nil z = a => adf0 z nil a = a

(osso) = adf0 z s z = o => adf0 z s z = o
adf0 z o z = z => adf0 z s z = nil
adf0 z nil z = s => adf0 z o s = s
adf0 z nil z = o => adf0 z nil z = o

(asm) = adf0 z m z = a => adf0 z s a = a
adf0 z nil z = z => adf0 z m z = z
adf0 z nil z = m => adf0 z nil z = m

A função automaticamente definida ADF1 retorna sempre a constante nula, uma vez que a letra na posição atual jamais será um valor nulo. O valor retornado é sempre T10_ADF1 que representa a constante NIL. Esta função é ignorada pelo ramo de produção de resultados e não tem qualquer influência no resultado final. Argumentos e

terminais também são ignorados tanto pelas funções quanto pelo ramo de produção de resultados.

A solução encontrada é equivalente à descrita na primeira execução para o caso 4 sem uso de ADF. Naquela execução, o melhor fitness encontrado foi de 30 pontos e os resultados obtidos foram semelhantes. A regra que converte a letra "c" para k ou s também não foi implementada de forma completa. A principal diferença entre as soluções está na transferência da solução para uma função definida automaticamente e na complexidade estrutural apresentada por ambas. A fig. 6.16 descreve a evolução desta complexidade durante o processo:

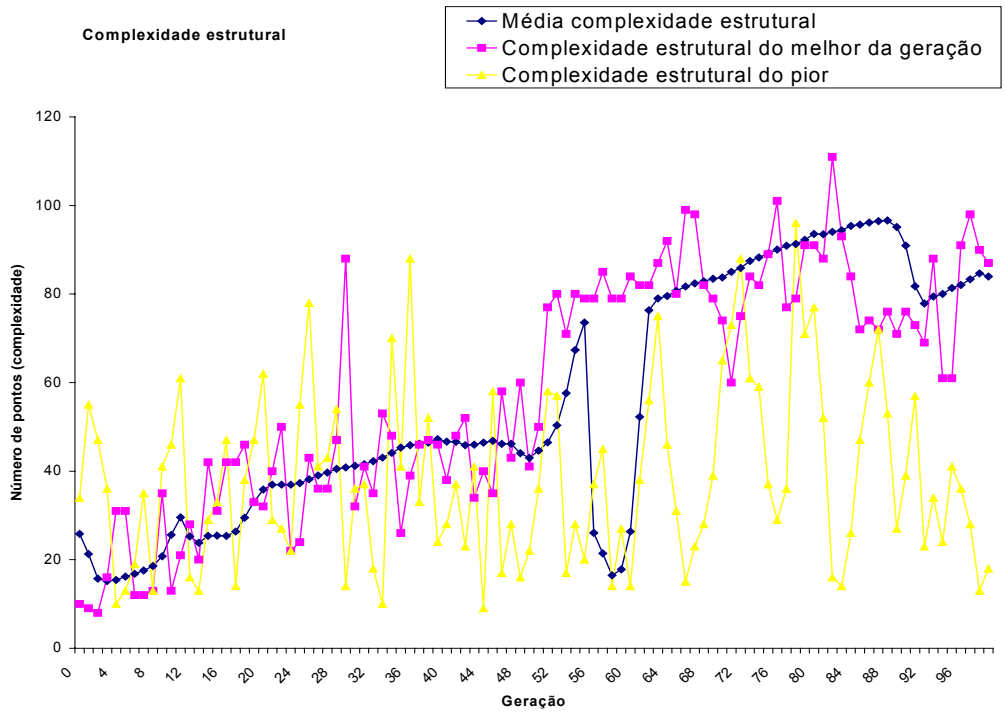


FIGURA 6.16 - Complexidade estrutural para o caso 4, usando ADF

A melhor resposta nesta execução possui 72 pontos, considerando-se somente os pontos funcionais como citado anteriormente. A diferença entre as melhores soluções com e sem ADF, foi de 17 pontos. Observando-se a média da complexidade estrutural é possível perceber que a diferença é ainda mais acentuada. Na execução sem a utilização da definição automática de funções, a complexidade média atingiu em seu ponto mais alto, valores próximos a 300 pontos, enquanto a fig.6.16 mostra que o número médio de pontos atingiu valores máximos próximos a 100.

A validação do melhor indivíduo obteve um fitness de 24; uma vez que o maior valor possível é 189, o percentual de acerto do indivíduo é aproximadamente 88%, um pouco inferior ao obtido no treinamento.

TABELA 6.17 – Validação para o caso 4, usando ADF

Casos de validação	Respostas esperadas	Respostas produzidas
((c a c a) (c e c e) (c i c i) (c o c a) (a s s a) (i s s o) (e s s a) (a s a) (o s a) (u s e) (a s b) (e s d) (i s m) (t i t i) (d i d i) (z i z a) (z e z o) (a z) (u z))	((k a k a) (s e s e) (s i s i) (k o k a) (a s a) (i s o) (e s a) (a z a) (o z a) (u z e) (a z b) (e z d) (i z m) (ts i ts i) (dz i dz i) (z i z a) (z e z o) (a s) (u s))	((s a s a) (s e s e) (s i s i) (s o s a) (a s a) (i s o) (e s a) (a z a) (o z a) (u z e) (a z b) (e z d) (i z m) (t i z i) (d i z i) (z i z a) (z e z o) (a s) (u s))

6.6 Tempos de processamento

Como informação final sobre as diversas execuções e para possibilitar comparações, a tab. 6.18 apresenta um resumo dos tempos aproximados de cada uma das execuções realizadas. Os tempos são expressos em minutos e convertido em horas e permitem avaliar o efeito dos parâmetros, da composição dos indivíduos e das características da modelagem do problema no desempenho da técnica.

TABELA 6.18 - Tempos aproximados das execuções

Execução	Minutos	Horas
Caso 1 - Relação um para um e letras "t" e "d" antes de "i". Resultados utilizando somente funções básicas	71	1,18
Caso 1 - Relação um para um e letras "t" e "d" antes de "i". Inserção de funções de domínio	31	0,52
Caso 2 - Nasalização de vogais.	1610	26,83
Caso3 - Combinação "rr".	192	3,20
Caso 4 - Regras envolvendo as letras "c", "s" e "z". Funções de domínio com argumentos.	960	16,00
Caso 4 - Regras envolvendo as letras "c", "s" e "z". Funções de domínio sem argumentos.	906	15,10
Caso 1 utilizando ADF	252	4,20
Caso 4 utilizando ADF	1435	23,92

Os parâmetros que influenciam diretamente o tempo de processamento são o tamanho da população, o número de gerações e a complexidade estrutural. É possível perceber que o caso 2, que possui 200 gerações e uma alta complexidade estrutural é o que demandou o maior tempo para execução. O caso 4 é o que possui a maior complexidade estrutural e mesmo com a metade das gerações do caso 2, levou mais da metade do tempo deste último, o que confere a ele o maior tempo por geração.

A escolha dos casos de fitness é fundamental para descoberta de boas soluções e afeta de forma significativa o desempenho. Um grande conjunto de casos de fitness ou um maior número de fonemas esperados naturalmente implica em um tempo maior para avaliação de cada indivíduo. Como o processo de avaliação é a atividade mais demorada, estas características tem certamente uma influência decisiva na velocidade da execução. Isto pode ser constatado também nos casos 2 e 4, que possuem maiores valores para o fitness básico, fato decorrente de um maior número de letras e fonemas.

Outra constatação importante é que a manipulação de indivíduos com funções automaticamente definidas é mais lenta se comparada à de indivíduos sem esta estrutura. A complexidade estrutural média para o caso 4 com ADF, por exemplo, é inferior ao mesmo caso sem ADF, porém o tempo de processamento foi maior. Pelo fato de gerar complexidades menores a estratégia com definição automática de funções tende a apresentar melhor desempenho ao longo de um número maior de gerações.

O equipamento utilizado foi uma máquina Pentium 333 Mhz, 128 MB RAM. É preciso salientar que os tempos apresentados dependem diretamente dos recursos do equipamento utilizado e da própria otimização do sistema. No desenvolvimento da aplicação de programação genética para este trabalho não houve uma avaliação dos módulos e rotinas com objetivo de melhorar o desempenho; para aplicações mais complexas ou devido à necessidade de integração com outros sistemas, um trabalho de otimização é importante. A própria natureza aleatória do processo leva a diferentes tempos em diferentes execuções, mesmo utilizando-se parâmetros iguais aos descritos.

7 Conclusões

Este trabalho apresentou a técnica de programação genética como uma forma de descoberta automática de programas e expôs ainda os resultados de sua aplicação na descoberta de regras para conversão de letra para fonema. O estudo baseou-se nas características elementares descritas por John R. Koza [KOZ 92], [KOZ 98].

Conforme já observado por outros autores, no paradigma da Programação Genética, a transformação entre o genótipo, que constitui codificação da solução e o fenótipo, que representa o comportamento do mesmo é uma atividade relativamente óbvia. É possível constatar a partir dos resultados que o comportamento é observável diretamente diretamente nas instruções que compõe o indivíduo, não sendo necessário portanto uma transformação complexa da saída para sua interpretação. Tal característica pode facilitar em futuras aplicações a extração do conhecimento produzido no processo.

Através do estudo, implementação e dos resultados obtidos nas diversas execuções, foi possível constatar que a programação genética pode ser boa alternativa para o descobrimento de regras. Embora utilizando estratégias diferentes das que certamente seriam usadas por um programador humano, a técnica foi capaz de produzir a partir de instruções e terminais, soluções que resolvem senão de forma completa, de maneira aceitável a conversão de letra-fonema.

Os experimentos realizados demonstram que é possível construir sistemas capacitados a descobrir regras através de um aprendizado supervisionado, permitindo assim, a criação de sistemas desenvolvidos que tenham capacidade de adaptação a diferentes idiomas e suas variações regionais. A estratégia escolhida na atividade de modelagem permite o descobrimento de regras que funcionam corretamente para diferentes contextos e entradas. O processo de modelagem descrito demonstra também que é possível utilizar outras abordagens para representação das soluções e que de acordo com objetivos e com necessidades de integração a outros sistemas podem ser alternativas interessantes.

Conforme descrito no item 3.3.1, a indução de programas apresenta como problema clássico a formulação dos casos usados no treinamento. Algumas soluções encontradas apresentam regras que podem não funcionar em qualquer contexto, mas que são verificáveis nos exemplos de palavras ou sílabas usados para o treinamento; tal observação enfatiza a importância e dependência da técnica de uma correta seleção dos casos de fitness. Esta escolha pode parecer em um primeiro momento óbvia, entretanto quanto maior o conjunto de regras a serem descobertas, maior a dificuldade envolvida nesta atividade. Um exemplo clássico desta dificuldade pode ser observado no caso 2, nas regras para nasalização de vogais. A partir dos casos usados para o aprendizado, a nasalização acabou ocorrendo em situações incorretas, devido ao grande número de casos de nasalização usados no treinamento. Entretanto, um conjunto de casos de fitness mais amplo como as palavras utilizadas na validação, tende a apresentar soluções que não implementam a regra de nasalização.

Embora cada um dos subconjuntos descritos tenha obtido no mínimo uma solução aceitável, não é possível afirmar que todas as execuções chegarão a uma boa solução. Uma das características dos resultados descritos é que eles representam amostras dos melhores resultados para cada caso, sendo assim, a execução da técnica com os mesmos parâmetros e definições, não garante a obtenção de resultados iguais aos documentados no trabalho.

Uma das dificuldades verificadas na análise dos resultados é a descrição das atividades implementadas por cada solução. A dificuldade decorre diretamente do aumento da complexidade, do grande número de instruções que compõe estes indivíduos. Outro problema está relacionado com a própria linguagem LISP, utilizada neste trabalho, que na maioria dos casos apresenta um código difícil de ser entendido e por conseqüência de difícil manutenção.

O tamanho e excesso de material genético inútil presente nas boas soluções é um problema observado na grande maioria das soluções produzidas, e ocorre principalmente pelo alto grau de destruição de bons blocos de instruções, decorrente das operações de cruzamento e mutação. Variações nestas operações podem permitir uma maior preservação de bons blocos e consequentemente um tamanho médio menor das soluções. Entretanto, é importante salientar que altas taxas de destruição também colaboram para uma maior diversidade do material genético, fato que pode levar a construção de melhores blocos do que aqueles verificados nas gerações iniciais. A partir destas constatações fica evidente a dificuldade para solucionar o problema e a necessidade de um aprofundamento desta análise em trabalhos futuros.

Os resultados descritos apresentam algumas variações que merecem análise detalhada. A utilização de funções de domínio tem uma influência direta para geração de melhores soluções, o que é compreensível porque parte do trabalho esperado na execução da técnica é feito manualmente e inserido como uma função pronta. Funções de domínio sem argumentos aumentam ainda mais a capacidade de solucionar o problema, entretanto pode ser inviável em diferentes estratégias de solução ou ainda em diferentes problemas. Funções sem argumentos podem ser consideradas como um excesso de influência de programadores humanos, sendo que esta alternativa pode direcionar em demasia o processo.

A avaliação de fitness é uma das atividades mais importantes em qualquer técnica evolutiva. A medição do grau de adaptação neste trabalho é simples e constitui na comparação direta do resultado esperado, com aquele produzido por cada indivíduo. Diversos indivíduos possuem conjuntos de instruções bons, que entretanto não são executados ou estão mal situados na árvore representativa do indivíduo, o que faz com que a resposta final seja diferente daquela esperada. Sugere-se para futuros trabalhos, a busca de funções de avaliação que possam aproveitar estes blocos, pois a partir de operações de cruzamento e mutação que envolvam estas soluções, possivelmente serão geradas combinações com alto grau de adaptação.

A extensão da técnica, que permite evoluir funções, além de um ramo principal se mostrou uma boa alternativa, tendo sido possível verificar a produção de resultados iguais ou melhores e principalmente com diminuição do número de pontos das soluções. Esta diminuição produz soluções mais eficientes e diminui sensivelmente o consumo de recursos computacionais. O problema ao qual foi aplicada a técnica não apresenta uma regularidade ou aplicação sucessiva dos mesmos conjuntos de regras. Devido a isto a reutilização foi pouco explorada. A utilização de diferentes estratégias, como por exemplo o uso de repetição na solução poderia levar a uma maior exploração da reutilização das funções automaticamente definidas.

Variações em conjuntos de funções e terminais, funções de domínios, funções automaticamente definidas são demonstrações claras das alternativas possíveis a serem exploradas para busca de melhores soluções. A riqueza da representação da

programação genética proporciona estas e outras alternativas, que também merecem um aprofundamento em trabalhos futuros.

Os recursos computacionais e o tempo necessário para obtenção de soluções também é um fator importante. A execução descrita nesta dissertação foi realizada em um conjunto reduzido de recursos computacionais e apesar disso boas soluções foram produzidas. Respostas melhores podem ser obtidas usando-se sistemas computacionais de alto desempenho. Os tempos de processamento descritos, embora aceitáveis, tendem a crescer muito com o aumento do número de casos de fitness e da complexidade do problema, que requer maiores conjuntos de funções e terminais.

Cabe enfatizar a importância do sistema desenvolvido de forma a facilitar variações nos conjuntos de parâmetros do processo. Através deste sistema foi possível variar facilmente os diversos parâmetros e características do problema. Trabalhos futuros poderão envolver a aplicação da Programação Genética a outras áreas do conhecimento humano, utilizando o sistema desenvolvido.

Finalizando, é possível concluir que a técnica tem um bom potencial para descoberta de soluções para o domínio apresentado e ainda outros problemas relacionados à síntese de voz. A facilidade oferecida pela representação da técnica, que utiliza instruções comuns em uma linguagem de programação, oferece uma flexibilidade e facilidade para abordagem de diferentes problemas.

Anexo 1 Representação de fonemas no "IPA - International Phonetic Alphabet" e no sistema Festival

Símbolos IPA	Símbolos Festival(Spoltech)	Exemplos
A	a	Pá, gato
E	e	Medo, seda
ɛ	E	Mel, pé
I	i	Vir, bico
O	o	Soma, morro
ɔ	>	Avó, cola
u	u	Tu, bambu
ã,ẽ(nasalização vogais)	ax~, e~, i~, o~, u~	Cama, tenda, monte.
m	m	Mato
n	n	Nada, cano
ɳ	n~	Ninho, vinho
b	b	Bate, boi
p	p	Pai, pato
d	d	Dar, andar
t	t	Tu, taco
g	g	Gola, gato
k	k	Casa, quero
f	f	Faca, afiar
v	v	Uva, vovô
s	s	Saber, posso
z	z	Bazar, casa
ʃ	S	Mecha, xarope
ʒ	Z	Já, gela
ʎ	L	Filho, calha
l	l	Lata, veludo
l	w	Fuzil, alto
r	r	Cara, mara
R	x	Carroça, carro
t,	ts	Tia, atira
d,	dz	Dia, adiar

Anexo 2 Regras para conversão letra para fonema no sistema Festival

```
Letter to sound rules for Brazilian Portuguese
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(lts.ruleset
      ;  Name of rule set
 port_br
      ;  Sets used in the rules
 (
  (ALPHA a e i o u á é í ó ú â ê ô ü ã õ à b c ç d f g j k l m n p q r
s t v w x y z)
  (PRE_RL b c d f g p t v k)
  (STV á é í ó ú â ê ô)
  (TV ã õ)
  (TREMA ü)
  (LNS l n s )
  (AV a e i o u á é í ó ú â ê ô ü ã õ à)
  (AV_EI a o u á ó ú â ô ü ã õ à)
  (EI e i é í ê)
  (V a e i o u )
  (AO a o)
  (AEOU a e o u á é ó ú)
  (AOU a o u)

  (A a á â ã à)
  (E e é ê)
  (I i í)
  (O o ó ô õ)
  (U u ú)

  (BDGLMN b d g l m n )
  (MN m n )
  (RL r l)
  (C_MN  b c ç d f g h j k l p q r s t v w x y z #)
  (C b c ç d f g h j k l m n p q r s t v w x y z)
  (C_H b c ç d f g j k l m n p q r s t v w x y z)
  (C_QG b c ç d f h j k l m n p r s t v w x y z)
  ;; consoantes com exceção de Q e G
  (C_RL b c ç d f g h j k m n p q s t v w x y z)
  ;; consoantes com exceção do r e l
  (CVEL g k)
  ;; consoantes velares(incompleto)
  (QG q g)
  (CO l m n r z u)
  (TILDE "~"))

 (
  ( [ a i ] x = a j )
  ( [ a i ] s # = a j )
  ( [ a m ] # = ax~ )
  ( [ a n ] # = ax~ )
  ( [ á m ] # = ax~1 )
  ( [ á n ] # = ax~1 )
  ( [ â m ] # = ax~1 )
  ( [ â n ] # = ax~1 )
  ( [ a ] C_MN = a )
  ( [ a ] AV = a )
  ( [ a ] MN C = ax~)
```

```
( [ a ] = a )

( [ á ] C_MN = a1 )
( [ á ] MN C = ax~1 )
( [ á ] AV = a1 )
( [ á ] = a1 )

( [ â ] C_MN = a1 )
( [ â ] MN C = ax~1 )
( [ ã ] = ax~ )

( [ e i ] a = e j )
( [ e i ] x = e j )
( [ e i ] s # = e j )
( [ e i ] # = e j )
( [ e i ] j = e j )
( [ e i ] r V = e j )
( [ e i ] MN = e j )
( [ e u ] = e w )
( STV C + [ e o ] # = j u2 )
( [ é u ] = E1 w )
( [ é i ] AV = E j )
( [ é i ] C_MN = E1 j )
( [ e n ] # = e~ )
( [ e m ] # = e~ )
( [ é n ] # = e~1 )
( [ é m ] # = e~1 )
( [ ê n ] # = e~1 )
( [ ê m ] # = e~1 )
( C [ e ] # = i2 )
( C [ e ] s # = i2 )

( [ e ] I = e )
( [ e ] I s = e )
( [ e ] MN C = e~ )
( [ e ] C_MN = e )
( [ e ] AV = e )
( V [ e ] # = j )
( STV C [ e ] # = j )
( V [ e ] s # = j )
( STV C [ e ] s # = j )
( [ e ] # = i2 )
( [ e ] = e )

( [ é ] u  = E1 )
( [ é ] i  = E1 )
( [ é ] C_MN = E1 )
( [ é ] V = E1 )
( [ é ] MN C = e~1 )
( [ é ] = E1 )

( [ ê ] C_MN = e1 )
( [ ê ] MN C = e~1 )
( [ ê ] = e1 )

( [ e ] C a = e~ )
( # [ e ] s = e )
( # [ e ] x = e )

( # C [ e ] m = e~ )
( # C [ e ] n = e~ )
```

```
( # [ e ] m = e~ )
( # [ e ] n = e~ )

( STV C * [ i o ] # = j u )
( [ i o ] # = i w )
( STV C * [ i a ] # = j a )
( STV C * [ i l ] # = j u )

( [ i l ] # = i w )
( [ i u ] # = i w )
( [ i m ] # = i~ )
( [ i n ] # = i~ )
( [ í m ] # = i~1 )
( [ í n ] # = i~1 )
( [ i ] r # = i )
( STV C [ i ] V = j )
( [ i ] MN C = i~ )
( QG u [ i ] = i )
( V [ i ] = j
( [ i ] = i )
( [ í ] C_MN = i1 )
( [ í ] MN C = i~1 )
( [ í ] V = i1 )
( [ í ] = i1 )

( C [ o ] # = u2 )
( C [ o ] s # = u2 )

( [ o u ] = o w )
( [ o l ] # = > w )
( [ o i ] = o j )
( [ ó i ] = >1 j )
( [ o n ] C_H = o~ )
( [ o m ] # = o~ )
( [ o n ] # = o~ )
( [ ó m ] # = >1 )
( [ ó n ] # = >1 )
( [ ô m ] # = o~1 )
( [ ô n ] # = o~1 )

( CVEL [ o ] a = w )
( CVEL [ o ] e = w )
( STV C [ o ] V # = w )
( V [ o ] s # = w )
( STV C [ o ] V s # = w )
( [ o ] C_MN = o )
( [ o ] MN C = o~ )
( [ o ] AV = o )
( AV [ o ] # = w )
( [ o ] # = u )
( [ o ] = o )

( [ ó ] C_MN = >1 )
( [ ó ] MN C = >1 )
( [ ó ] V = >1 )
( [ ó ] = >1 )

( [ ô ] C_MN = o1 )
( [ ô ] MN C = o~1 )
( [ ô ] V = o1 )
( [ ô ] = o1 )
```

```
( [ õ ] = o~ )

( [ u o ] = w > )
( [ u e ] C_MN # = w e )
( STV C + [ u e ] # = w i )
( [ u é ] # = w E )
( [ u m ] # = u~ )
( [ u n ] # = u~ )
( [ ú m ] # = u~1 )
( [ ú n ] # = u~1 )

( # [ u ] V r = u )
( # [ u ] V m = u )
( # [ u ] V s = u )

( C_QG [ u ] AV r = u )
( C_QG [ u ] AV m = u )
( C_QG [ u ] AV s = u )
( QG [ u ] AO = w )
( QG [ u ] EI = )

( V [ u ] CO = u )
( AV [ u ] = w )
( STV C + [ u ] AV = w )

( [ u ] C_MN = u )
( [ u ] MN C = u~ )
( [ u ] AV = u )
( [ u ] = u )

( QG [ ü ] EI = w )
( [ ú ] C_MN = u1 )
( [ ú ] MN C = u~1 )
( [ ú ] AV = u1 )
( [ ú ] = u1 )

( [ b ] AV = b )
( [ b ] RL AV = b )
( [ b ] C = b )
( [ b ] = b i )

( [ c h ] AV = S )

( [ c ] C_RL = k i )
( [ c ] EI = s )
( [ c ] AV_EI = k )
( [ c ] RL AV = k )
( [ c ] = s i )

( [ ç ] AV = s )
( [ ç ] = s i )

( [ d ] I = dZ )
( [ d ] AV = d )
( [ d ] RL AV = d )
( [ d ] = dZ )

( [ f ] AV = f )
( [ f ] RL AV = f )
( [ f ] = f i )
```

```
( [ g ú ] EI = g u1 )
( [ g ü ] EI = g w )
( [ g u ] EI = g )
( [ g u ] AV_EI = g w )
( [ g ] RL AV = g )
( [ g ] EI = Z )
( [ g ] AV_EI = g)
( [ g ] = g i )

( [ j ] AV = Z )
( [ j ] = Z i )

( [ k ] AV = k )
( [ k ] RL AV = k )
( [ k ] = k i )

( [ l h ] V = L )
( STV [ l i ] V = L )
( STV [ l e ] V = L )
( U [ l ] # = )
( U [ l ] C = )
( AV [ l ] # = w )
( AV [ l ] C = w )
( [ l ] AV = l )
( [ l ] = l u~ )

( [ m ] AV = m )
( AV [ m ] C = )
( AV [ m ] # = )
( [ m ] = m ax~ )

( [ n h ] AV = n~ )

( [ n ] AV = n )
( AV [ n ] C_H = )
( AV [ n ] # = )
( [ n ] = n ax~ )

( [ p ] AV = p )
( [ p ] RL AV = p )
( [ p ]  = p e )
( [ q u ] EI = k )
( [ q u ] AV_EI = k w )
( [ q ü ] = k w )
( [ q ú ] = k u1 )
( [ q ] = k w )

( AV [ r r ] AV = x )
( AV [ r ] AV = r )
( AV [ r ] C = r )
( PRE_RL [ r ] AV = r )
( # [ r ] AV = x )
( AV [ r ] # = r )
( [ r ] = r i )

( [ s s ] AV = s )
( [ s ç ] AV = s )
( [ s c ] EI = s )
( # [ s ] AV = s )
( C [ s ] AV = s )
```

```
  ( AV [ s ] C = s )
  ( AV [ s ] AV = z )
  ( AV [ s ] # = s )
  ( C [ s ] # = s )
  ( [ s ] = i s )

  ( [ t e ] # = tS i2 )
  ( [ t ] I = tS )
  ( [ t ] AV = t )
  ( [ t ] RL AV = t )
  ( [ t ] = t )


  ( [ v ] AV = v )
  ( [ v ] RL AV = v )
  ( [ v ] = v i )
  ( AV [ x c ] EI = s )
  ( AV [ x s ] AV = s )
  ( # EI [ x ] AV = z )
  ( C [ x ] AV = S )
  ( # [ x ] AV = S )
  ( AV [ x ] C AV_EI = s )
  ( AV [ x ] C RL AV = s )
  ( AV [ x ] C = s )
  ( AV [ x ] # = k s )
  ( AV [ x ] AV = S )

  ( [ w ] AV = v )
  ( [ w ] RL AV = v )
  ( [ w ] = v i )

  ( [ y ] = i )

  ( [ z ] AV = z )
  ( AV [ z ] C = s )
  ( AV [ z ] # = s )
  ( [ z ] = z i )


  ( ALPHA [ h ] ALPHA = )
  ( # [ h ] = ) ( [ h ] # = )


(define (port_br_lts word features)
  "(port_br_lts WORD FEATURES)
(let (phones dword)
    ;;    (if (lts.in.alphabet word 'port_downcase)
    (set! word (port_cleanword word))
    (if (string-equal word "")
      nil
      ;; else
      (begin
        (set! dword (port_downcase word))
        (print dword)
        ;;  (set! dword (port_downcase "equis")))
        (set! phones (lts.apply dword 'port_br))
        (print phones)
        (list word
            nil
            (port.syllabify.phstress phones))))))
```

```
(define (port_cleanword word)
;(set! LETTERS (a e i o u á é í ó ú â ê ô ü ã õ à b c ç d f g j k l m
n p q r s t v w x y z A E I O U Á É Í Ó Ú Â Ê Ô Ü Ã Õ À B C Ç D F G J
K L M N P Q R S T V W X Y Z))
  (if word

    (if (string-matches word "[a-zA-Záéíóúâêôüãõàç ÁÉÍÓÚÂÊÔÜÃÕÀÇ]*")
      word
      ;; else
      (begin
      (set! letter (substring word 0 1))
      (set! word (substring word 1 (string-length word)))
      (if (string-matches letter "[a-zA-Záéíóúâêôüãõàç ÁÉÍÓÚÂÊÔÜÃÕÀÇ]")
                              ;(member (car word) LETTERS)
        (set! result (string-append letter (port_cleanword word)))
        (set! result (port_cleanword word)))
      result))
   nil))

(define (port_downcase word)
  "(port_downcase WORD)
Downs case word by letter to sound rules becuase or accented form
this can't use the builtin downcase function."
  (lts.apply word 'port_downcase))
 (provide 'port_br_lts)
```

Anexo 3 Código das funções de domínio utilizadas no problema

```
;; Implementação das funções externas necessárias para o problema
;; Estas funções são usadas na resolução como conjunto de funções


; Modifica a lista que contém o resultado (*lvresult*)
; Usada em testes com estratégias iniciais
(defun fresult (pfone)
   (setf lvtemp '())
   (if (not (eq pfone nil)) (setf lvtemp (cons pfone lvtemp)))
   (setf vcontdo (- (length *lvresult*) 1))
   (loop until (< vcontdo 0) do
      (setf lvtemp (cons (nth vcontdo *lvresult*) lvtemp))
      (setf vcontdo (- vcontdo 1)))
   (setf *lvresult* lvtemp)
   (values pfone))

; Função que testa se o parâmetro ppos for maior que zero e menor ou
igual ao tamanho da lista
; retorna o caractere da posição atual
(defun cposat ()
   (let (vcpos)
   (setf vcpos NIL)
   (if (numberp *vposletra*)
      (setf vcpos
         (if (and (> *vposletra* 0)
             (<= *vposletra* (length *lvcasofit*)))
             (nth (- *vposletra* 1) *lvcasofit*) NIL)))
   (values vcpos)))

; Função que testa se o parâmetro ppos for maior que um e menor ou
igual ao tamanho da lista, retorna o caractere da posição
imediatamente anterior
(defun cposan1 ()
   (let (vcpos)
   (setf vcpos NIL)
   ; Teste se é número para evitar erro de tipo
   (if (numberp *vposletra*)
       (setf vcpos (if (and (> *vposletra* 1)
                       (<= *vposletra* (length *lvcasofit*)) )
             (nth (- *vposletra* 2) *lvcasofit*) NIL)))
   (values vcpos)))

; Função que testa se o parâmetro ppos for maior que dois e menor ou
igual ao tamanho da lista, retorna o caractere duas posições atrás da
atual
(defun cposan2 ()
   (let (vcpos)
   (setf vcpos NIL)
(if (numberp *vposletra*)
   (setf vcpos
         (if (and
             (> *vposletra* 2) (<= *vposletra* (length *lvcasofit*)) )
             (nth (- *vposletra* 3) *lvcasofit*) NIL)))
   (values vcpos)))
```

```
; Função que testa se o parâmetro ppos for maior que zero e menor que
o tamanho da lista, retorna o caractere da posição imediatamente
posterior
(defun cpospo1 ()
   (let (vcpos)
   (setf vcpos NIL)
   ; Teste se eh número para evitar erro de tipo
   (if (numberp *vposletra*)
   (setf vcpos
         (if (and
             (> *vposletra* 0) (< *vposletra* (length *lvcasofit*)) )
             (nth *vposletra* *lvcasofit*) NIL)))
   (values vcpos)))

; Função que testa se o parâmetro ppos for maior que zero e no minimo
dois a menos que o tamanho da lista
; retorna a o caractere duas posições a frente da atual
(defun cpospo2 ()
   (let (vcpos)
   (setf vcpos NIL)
   ; Teste se eh número para evitar erro de tipo
   (if (numberp *vposletra*)
   (setf vcpos
         (if (and
             (> *vposletra* 0) (< *vposletra* (- (length *lvcasofit*)
1)))
             (nth (+ *vposletra* 1) *lvcasofit*) NIL)))
   (values vcpos)))

; Macro que implementa uma repetição controlada
; Executa o segundo argumento até que o predicado avaliado no primeiro
seja satisfeito
; Tem um dispositivo de proteção para que nunca ocorra loop infinito.
; Usada em algumas estratégias

(defmacro dolfon (corpo)
   (let (vcontdo)
   (setf vcontdo 1)
   (loop until (> vcontdo (length *lvcasofit*))
     do (eval corpo)
        (setf vcontdo (+ vcontdo 1))
   )))

; Função de incremento protegido
; Usada na avaliação de fitness para incremento da posição atual
(defun incpos ()
   (setf *vposletra* (+ *vposletra* 1))
   (values *vposletra*))

;; Função de incremento protegido, que aumenta
(defun incpos2 ()
   (setf *vposletra* (+ *vposletra* 2))
   (values *vposletra*))

;Verifica se a letra recebida como parametro eh uma vogal
(defun fehvogal (pletra)
   (values (if (or (eq pletra 'a)
           (eq pletra 'e)
           (eq pletra 'i)
           (eq pletra 'o)
           (eq pletra 'u))
```

```
      t
      nil)))

;Retorna a nasalizacao da vogal recebida como parametro
;Caso nao seja vogal, retorna valor nulo

(defun fnasal (pvogal)
   (values (if (eq pvogal 'a) 'ax~
               (if (eq pvogal 'e) 'e~
                  (if (eq pvogal 'i) 'i~
                     (if (eq pvogal 'o) 'o~
                        (if (eq pvogal 'u) 'u~ nil)))))
    )
   )

;Verifica se a letra recebida como parametro eh letra m ou n
(defun fehmn (pletra)
   (values (if (or (eq pletra 'm) (eq pletra 'n))
               t
               nil))
   )

; Verifica se o primeiro argumento é a letra t ou d
; Verifica também se o segundo argumento é a letra i
(defun fehtd (pletra pletra2)
   (values (if (or (eq pletra 't-) (eq pletra 'd))
                (if (eq pletra2 'i) t nil)
            nil))
   )

; Retorna fonemas das letras t e d quando elas forem antes de i
(defun frtdi (pletra)
   (values (if (eq pletra 't) 'ts
               (if (eq pletra 'd) 'dz))
   )
)

;Verifica se o primeiro argumento é a letra c
; Verifica também se o segundo argumento é e ou i
(defun fehcei (pletra pletra2)
   (values (if (and
                  (eq pletra 'c)
                  (or (eq pletra2 'e) (eq pletra2 'i)))
               t
               nil))
   )

; Verifica se o primeiro argumento é a letra s
; Verifica se o segundo argumento é uma das letras (d,b,m,n ou g)
(defun fehscvo (pletra pletra2)
   (values (if (and (eq pletra 's)
                (or (eq pletra2 'd)
                   (eq pletra2 'b)
                   (eq pletra2 'm)
                   (eq pletra2 'n)
                   (eq pletra2 'g)))
               t
               nil))
   )
```

```
; Verifica se ambos os argumentos são s
(defun fehss (pletra pletra2)
   (values (if (and
                    (eq pletra 's)
                    (eq pletra2 's))
               t
               nil)
     )
   )

; Verifica se o primeiro argumento é vogal
; Verifica também se o segundo s
; Por fim teste se o terceiro é vogal
(defun fehvsv (pletra pletra2 pletra3)
    (values (if (and
                  (fehvogal pletra)
                  (eq pletra2 's)
                  (fehvogal pletra3))
                t
                nil)))

; Verifica se o primeiro argumento é vogal
; Verifica se o segundo é z
; Por fim se o terceiro tem valor nulo
(defun fehvznil (pletra pletra2 pletra3)
   (values (if (and
                (fehvogal pletra)
                (eq pletra2 'z)
                (eq pletra3 nil))
               t
               nil)))
;; A partir daqui algumas funções estão redefinidas
;; As funções não possuem argumentos, a aplicação é diretamente sobre
caracteres em posições específicas.

; Função fehtd aplicada sobre a posição atual e a próxima
(defun fehtd ()
   (values (if (or (eq (cposat) 't) (eq (cposat) 'd))
                (if (eq (cpospo1) 'i) t nil)
               nil))
   )

; Função frtdi aplicada diretamente a posição atual
(defun frtdi ()
   (values (if (eq (cposat) 't) 'ts
               (if (eq (cposat) 'd) 'dz))
   )
)

; Função fechcei aplicada diretamente a posição atual e posterior
(defun fehcei ()
   (values (if (and
                  (eq (cposat) 'c)
                  (or (eq (cpospo1) 'e) (eq (cpospo1) 'i)))
               t
               nil))
   )

; Função fehscvo aplicada diretamente a posição atual e próxima
(defun fehscvo ()
   (values (if (and (eq (cposat) 's)
```

```
                (or (eq (cpospo1) 'd)
                   (eq (cpospo1) 'b)
                   (eq (cpospo1) 'm)
                   (eq (cpospo1) 'n)
                   (eq (cpospo1) 'g)))
                t
                nil))
   )


; Verifica se a posicao anterior eh vogal, a atual s e a proxima vogal
(defun fehvsv ()
    (values (if (and
                 (fehvogal (cposan1))
                 (eq (cposat) 's)
                 (fehvogal (cpospo1)))
                t
                nil)))

; Verifica se a primeira eh vogal, a segunda z e a terceira nula
; Implementação sem parâmetros
(defun fehvznil ()
   (values (if (and
                (fehvogal (cposan1))
                (eq (cposat) 'z)
                (eq (cpospo1) nil))
               t
               nil)))
```

Anexo 4 Funções de definição de casos e avaliação de fitness

```
; lvresult eh a lista de fonemas que um programa produz como resultado
(defvar *lvresult*)
; lvcasofit eh a sequencia de letras que constitui um caso de fitness.
(defvar *lvcasofit*)
; Variável que totaliza o fitness basico para calculo depois do
fitness padronizado
(defvar *vtotfitbas*)
; Variável que é usada para controlar a posição atual
(defvar *vposletra*)

; Estrutura de cada caso de fitness. Tipo do vetor de casos
(defstruct casos-fitness-letra-fonema
  palavra
  resfonemas
  )

; Definição dos casos de fitness, gera como resultado um vetor
retornado ao programa da PG
; Os casos compõe-se de um conjunto de letras e conjunto de fonemas
correspondente
(defun define-casos-fitness-letra-fonema ()

(let (pcasosfit vcasosfit presultcasos vresultcasos caso-fitness
casos-fitness indice numero-casos caso-atual vcontfon pfitmin)
    ; Obtém os casos dos componentes da interface
   (setf pcasosfit (widget :txtcasosfit :principal))
   (setf vcasosfit (dialog-item-value pcasosfit))
   (setf presultcasos (widget :txtresultcasos :principal))
   (setf vresultcasos (dialog-item-value presultcasos))
   (setf *vtotfitbas* 0)
   (setf casos-fitness (make-array *number-of-fitness-cases*))
   (setf numero-casos (- (length vcasosfit) 1))
   (setf indice 0)
   (loop until (> indice numero-casos) do
     (progn
      (setf caso-fitness (make-casos-fitness-letra-fonema))
      (setf (casos-fitness-letra-fonema-palavra caso-fitness)
            (nth indice vcasosfit))
      (setf (casos-fitness-letra-fonema-resfonemas caso-fitness)
            (nth indice vresultcasos))
      (setf (aref casos-fitness indice) caso-fitness)
      (setf caso-atual (nth indice vresultcasos))
      (setf vcontfon 1)
      (loop until (> vcontfon (length caso-atual)) do
        (progn
        (setf *vtotfitbas* (+ *vtotfitbas* 3))
        (incf vcontfon)))
        (incf indice)))
      (setf pfitmin (widget :txtfitnessmin :principal))
      (set-dialog-item-value pfitmin *vtotfitbas*)
      (print *vtotfitbas*)
  (values casos-fitness)))


; Avaliação dos casos de fitness, processo que retorna o fitness
padronizado e hits. Recebe como argumento o programa a ser avaliado e
```

os casos de fitness. O numero de hits é uma contagem para cada palavra completa acertada

```
(defun avaliacao-fitness-letra-fonema (programa conjunto-casos-
fitness)
   (let (vtotfitbasind vhitsind vfitpadind vindice caso-fitness
vresprog vfonpos vfonposr vindfones vfitbascaso as-letras os-fonemas
tamresach tamresre vindaval vteste)
      (setf vtotfitbasind 0)
      (setf vhitsind 0)
      (dotimes (vindice *number-of-fitness-cases*)
         (setf caso-fitness (aref conjunto-casos-fitness vindice))
         (setf as-letras (casos-fitness-letra-fonema-palavra
               caso-fitness))
         (setf os-fonemas (casos-fitness-letra-fonema-resfonemas
                           caso-fitness))
         (setf *lvcasofit* as-letras)
         (setf *lvresult* '())
         (setf *vposletra* 1)
         (dotimes (vindaval (length *lvcasofit*))
           (fresult (eval programa))
           (incf *vposletra* 1))
         (setf vfitbascaso 0)
         (setf tamresach (length *lvresult*))
         (setf tamresre (length os-fonemas))
         (if (< tamresach tamresre) (setf vtamrep tamresach)
                                    (setf vtamrep tamresre))
         (if (and (> (length *lvresult*) 0) (not (eq *lvresult* nil)))
            ; Repete e pega cada fonema do resultado
            (dotimes (vindfones vtamrep)
                (setf vfonpos (nth vindfones *lvresult*))
                (setf vfonposr (nth vindfones os-fonemas))
                (if (eq vfonpos vfonposr)
                    (progn
                     (setf vtotfitbasind (+ vtotfitbasind 3))
                     (setf vfitbascaso (+ vfitbascaso 3)))
                    (if (not (eq (member vfonpos os-fonemas) nil))
                        (progn
                         (setf vtotfitbasind (+ vtotfitbasind 1))
                         (setf vfitbascaso (+ vfitbascaso 1)))))
             )
(if (= (* (length os-fonemas) 3) vfitbascaso) (incf vhitsind))))
   (setf vfitpadind (- *vtotfitbas* vtotfitbasind))
   (if (< vfitpadind 0) (setf vfitpadind 0))
; Retorna os valores
   (values vfitpadind vhitsind)))
```

Bibliografia

[ALB 2000] ALBUQUERQUE, Paul; CHOPARD, Bastien; MAZZA, Christian; TOMASSINI, Marco. On Impact of the Representation of Fitness Landscapes. In: EUROPEAN WORKSHOP ON GENETIC PROGRAMMING, EUROGAP, 2000. **Genetic Programming:** European Conference: proceedings. Berlim: Springer 2000. P.1-15. (Lecture Notes in Computer Science, v.1802).

[AND 94] ANDRE, David. Learning and upgrading Rules for na OCR System using Genetic Programming. In:IEEE WORLD CONGRESS ON COMPUTACIONAL INTELIGENCE, WCCI, 1994, Orlando. **Proceedings**... Rscata-way: IEEE, 1994. Disponível em: <http://www.geneticprogramming.com>. Acesso em: 01 nov. 2000.

[BAN 95] BANZHAF, Wolfgang; NORDIN, Peter. **Real time Control of a Kephera Robot using Genetic Programming**. Dortmund, Germany:[s.n.], 1995. Disponível em: <http://www.geneticprogramming.com>. Acesso em: 04 dez. 2000.

[BAN 98] BANZHAF, Wolfgang; NORDIN, Peter; KELLER Robert; FRANCONE, F.D. **Genetic Programming An Introduction. On the Automatic Evolution of Computer Programs and Its Applications**. San Francisco:Morgan Kaufmann, 1998. 470 p.

[BAN 98a] BANZHAF, Wolfgang; NORDIN, Peter; CONRAD, Markus. **Speech Sound Discrimination With Genetic Programming**. 1998. Disponível em: <http://www.geneticprogramming.com>. Acesso em: 04 dez. 2000.

[BAR 2000] BARONE, Dante; SCHRAMM, Maurício; FREITAS, Luis Felipe R. A Brazilian portuguese language corpus development. In: INTERNATIONAL CONFERENCE ON SPOKEN LANGUAGE PROCESSING 6., 2000, Beijing. **Proceedings...** [S.l.:s.n.], 2000. v. 2, p. 579-582.

[BIS 99] BISOL, Leda. **Introdução a estudos de fonologia do português brasileiro**. 2.ed. Porto Alegre. EDIPUCRS, 1999. 254 p.

[BRA 92] BRANKO SOUCEK; IRIS GROUP. **Dynamic, Genetic and Chaotic Programming**. New York: John Wiley & Sons, 1992.

[CAL 90] CALLOU D.; LEITE Y. **Iniciação à Fonética e à Fonologia**. Rio de Janeiro: J. Zahar, 1990.

[CAM 80] CAMPOS G.L**. Síntese de Voz para o Idioma Português**. 1980. Tese (Doutorado) - Escola Politécnica, Universidade de São Paulo, São Paulo.

[CAR 98] CARDON, André. **Um Sistema de Síntese de Fala Através da Concatenação de Sílabas**. 1998. Dissertação (Mestrado em

Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

[CHB 94] CHBANE, Dimas T. **Desenvolvimento de sistema para Conversão de textos em fonemas no idioma português**. 1994. Dissertação (Mestrado em Engenharia) - Escola Politécnica, Universidade de São Paulo, São Paulo.

[COR 2000] CORDENONSI, André. **Um Ambiente de Evolução de Comportamentos para Sistemas Multiagentes Reativos**. 2000. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

[DAV 91] DAVIS, Lawrence D. **Genetic Algorithms Handbook**. New York: Van Nostrand Reinhold, 1991. 383 p.il.

[DOB 77] DOBZHANSKY, T.; AYALA, F.J.; STEBBINS, G.L.; VALENTINE, J.W. **Evolution**. San Francisco:W.H.Freeman, 1977.

[DOS 98] DOSVOX. **Síntese de fala no sistema dosvox**. 1998. Disponível em: <http://caec.nce.ufrj.br/~dosvox/textos.html> Acesso em: 10 maio 2002.

[DUT 96] DUTOIT Thierry. **An Introduction to Text-To-Speech Synthesis**. Dordrecht: Kluwer Academic, 1996. 280 p.

[FOG 2000] FOGEL, David B. **Toward a New Philosophy of Machine Intelligence**. New York: IEEE Press, 2000. 270 p.

[FRA 94] FRASER, Adam P. **Genetic Programming in C++.** [S.l.]: University Of Salford: Cybernetics Research Institute, 1994. (Technical Report, 040)

[HAU 99] HAUSSER, Roland. **Fundations of Computacional Linguistics. Man-Machine Comunication in Natural Languag**e. Berlin: Springer-Verlag, 1999. 534 p.

[HOL 75] HOLLAND, John H. **Adaptation in Natural and Artificial Systems**. Ann Arbor: University of Michigan Press, 1975.

[KLA 90] KLATT D.H.; KLATT L.C. Analysis, Synthesis and Perception of Voice Quality Variations Among Female and Male Talkers. **Journal of Acoustical Society of America**. [S.l.], v.87, n.2, p.820-56, Feb. 1990.

[KOZ 90a] KOZA, John R. **A Genetic Approach to Econometric Modeling**. 1990. Disponível em: <http://www.geneticprogramming.com>. Acesso em: 04 dez. 2000.

[KOZ 90b] KOZA, John R.; HALL, Margaret J. **Genetic evolution and co-evolution of Computer Programs**. 1990. Disponível em: <http://www.geneticprogramming.com>. Acesso em: 04 dez. 2000.

[KOZ 92] KOZA, John R. **Genetic Programming**: On the Programming of Computers by Means of Natural Selection. Cambridge: The MIT Press, 1992. 819 p.

[KOZ 98] KOZA, John R. **Genetic Programming II**: Automatic Discovery of Reusable Programs. Cambridge: The MIT Press, 1998. 746 p.

[KOZ 2000] KOZA, John R.; KEANE, Martin A.; BENNETT F. H.; ANDRE, Davis. **Genetic Programming**: Turing's Third Way to Achieve Machine Intelligence. Disponível em: <http://www.geneticprogramming.com> Acesso em: 04 dez. 2000.

[LEM 2000] LEMMETTY, Sami. **Review of speech synthesis technology**. Disponível em: <http://www.acoustics.hut.fi/~slemmett/dippa/chap1.htm>. Acesso em 20 dez. 2000.

[LAW 2000] LAWRENCE, Steve. Natural Language Gramatical Inference with Recurrent Neural Networks. **IEEE Transactions on Knowledge and Data Engineering**, New York, v.12, n.1, p.126-140, 2000.

[MIT 92] MITCHELL, Melanie. **An Introduction to Genetic Algorithms**. Cambridge: The MIT Press, 1992. 205 p.

[MBR 2000] MBROLA**. Frequently Asked Questions with answers**. 2000. Disponível em: <http://tcts.fpms.ac.be/synthesis/>. Acesso em 10 maio 2002.

[O'MA 90] O'MALLEY, M.H.; Text-To-Speech Conversion Technology. **Computer**, Los Alamitos, v.23, n.8, p.17-23, Aug. 1990.

[O'NE 2000] O'NEIL, Michael; RYAN, Conor. Crossover in Gramatical Evolution: A Smooth Operator?**.** In: EUROPEAN WORKSHOP ON GENETIC PROGRAMMING, EUROGAP, 2000**. Genetic Programming**: European Conference: proceedings. Berlin: Springer, 2000. p.149-162. (Lecture Notes in Computer Science, v.1802).

[O'RE 95] O'REILLY, Una-May et al**. An Analysis of Genetic Programming**. Ottawa: Ottawa-Carleton Institute for Computer Science, Carleton University, 1995.

[PET 2000] PETRY, Adriano. Bhattacharyya distance applied to speaker identification**.** In: INTERNATIONAL CONFERENCE ON SIGNAL PROCESSING APPLICATIONS AND TECHNOLOGY, 2000 , Dallas. **Conference proceedings**. [S.l. : DSP World], 2000.

[ROS 96] ROSCA, Justinian P.; BALLARD, Dana H. Evolution-based Discovery of Hierarchical Behaviors. In: INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE, 1996, Cambridge. **Proceedings**...[S.l.]: The MIT Press, 1996.

[ROS 96a] ROSCA, Justinian P. Generality versus Size in Genetic Programming. In: GENETIC PROGRAMMING CONFERENCE, GP, 1996. **Genetic Programming Conference**: proceedings. Cambridge: The MIT Press. 1996.

[SIL 93] SILVA, Miriam B. **Ensaios. Leitura, Ortografia e Fonologia**. São Paulo: Ática, 1993. 110 p.

[SIL 99] SILVA, T.C. **Fonética e Fonologia do Português, roteiro de estudos e guia de exercícios**. São Paulo: Contexto, 1999. 254 p.

[SLA 98] SLADE, Stephen. **Object-Oriented Common Lisp**. Upper Saddle River: Prentice Hall PTR, 1998. 774 p.

[SPO 01] SPOLTECH. Advancing Human Language Technology in Brazil and the United States Through Collaborative Research on Portuguese Spoken Language Systems. In: PROTEM-CC, 4., 2001, Rio de Janeiro. **Projects Evaluation Workshop**: international cooperation: procedings. Brasília: CNPw, 2001. p. 118-142.

[WES2001] WESTERDALE, Thomas. Local Reinforcement and Recombination in Classifier Systems. **Evolutionary Computing**, Cambridge, v.3, n.9, p.259-281. 2001.