CO-EVOLUÇÃO AMOSTRA-CLASSIFICADOR INTEGRADA À PROGRAMAÇÃO GENÉTICA GRAMATICAL PARA A CLASSIFICAÇÃO DE DADOS

Douglas Adriano Augusto

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA CIVIL.

Aprovada por:

Prof. Nelson Francisco Favilla Ebecken, D.Sc.

Prof. Helio José Corrêa Barbosa, D.Sc.

Prof. Alexandre Gonçalves Evsukoff, Dr.

Prof. Antonio César Ferreira Guimarães, D.Sc.

RIO DE JANEIRO, RJ - BRASIL
DEZEMBRO DE 2004

AUGUSTO, DOUGLAS ADRIANO

Co-evolução Amostra-Classificador integrada à Programação Genética Gramatical para a Classificação de Dados [Rio de Janeiro] 2004

XIX, 143 pp. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia Civil, 2004)

Tese – Universidade Federal do Rio de Janeiro, COPPE

1 - Programação Genética

2 - Co-evolução Competitiva

3 - Gramática Livre de Contexto

4 - Classificação de Dados

I. COPPE/UFRJ II. Título (série)

Àqueles que em algum grau possam usufruir desta obra.

"Os maiores acontecimentos e pensamentos são os que mais tardiamente são compreendidos."

"Quem chegou, ainda que apenas em certa medida, à liberdade da razão, não pode sentir-se sobre a terra senão como andarilho."

Friedrich Nietzsche

Agradecimentos

À minha família, em especial aos meus pais.

Aos meus orientadores, Helinho e Nelson,
pelo nobre exercício da orientação, sabedoria, paciência e suporte.

À CAPES/MEC,
pela gentileza na concessão da bolsa de estudos.

Aos grandes filósofos e engenheiros do Software Livre, pela disponibilização *livre* e gratuita de ferramentas e ambientes computacionais tecnicamente privilegiados.

A todos autores e seus esforços prévios,
que contribuíram (e contribuem) não somente com a confecção deste trabalho, mas principalmente no desenvolvimento sólido do conhecimento científico.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc)

CO-EVOLUÇÃO AMOSTRA-CLASSIFICADOR INTEGRADA À PROGRAMAÇÃO GENÉTICA GRAMATICAL PARA A CLASSIFICAÇÃO DE DADOS

Douglas Adriano Augusto

Dezembro/2004

Orientadores: Nelson Francisco Favilla Ebecken
Helio José Corrêa Barbosa

Programa: Engenharia Civil

A tarefa de classificação de dados dedica-se a garimpar amostras de dados à procura e extração de conhecimento. Propõe-se a sintetizar a essência das semelhanças que moldam as diferentes classes ou categorias de um certo problema. O êxito desta tarefa depende de algoritmos e técnicas que exerçam com qualidade e exatidão a descoberta do conhecimento implícito em um conjunto de dados.

O presente trabalho trata a tarefa de classificação de dados via técnicas de computação evolucionária, empregando-se essencialmente três abordagens: programação genética, co-evolução competitiva e gramática livre de contexto.

Explora-se a robustez e a qualidade simbólica/interpretativa da programação genética visando a construção, por evolução darwiniana, de árvores classificadoras. A estrutura formal e flexível da gramática livre de contexto substitui a forma representativa tradicional da programação genética, descrevendo uma linguagem capaz de codificar, de forma precisa e estruturalmente semântica, árvores de qualquer complexidade. Finalmente, a co-evolução competitiva integra-se ao sistema promovendo competições entre amostras de dados e árvores classificadoras, no intuito de criar tensões bilaterais positivas que estimulam a aceleração da escalada evolucionária.

Abstract of Thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

COEVOLUTION OF DATA SAMPLES AND CLASSIFIERS INTEGRATED WITH GRAMMATICAL GENETIC PROGRAMMING FOR DATA CLASSIFICATION

Douglas Adriano Augusto

December/2004

Advisors: Nelson Francisco Favilla Ebecken
Helio José Corrêa Barbosa

Department: Civil Engineering

The task of data classification consists in mining data samples in order to find and extract knowledge. It seeks to syntesize the essence of the similarities that define the different classes or categories of a given problem. The success of this task is dependent on algorithms and techniques that perform with efficiency and efficacy the discovery of implict knowledge in a data set.

The present work treats the data classification task by means of evolutionary computation techniques using specially three approaches: genetic programming, competitive coevolution, and context-free grammar.

The robustness and symbolic/interpretative qualities of the genetic programming are employed to construct classification trees via darwinian evolution. The flexible formal structure of the context-free grammar replaces the standard genetic programming representation and describes a language which encodes trees of varying complexity. Finally, competitive coevolution is used to promote competitions between data samples and classification trees in order to create and sustain an evolutionary race for improved solutions.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Capítulo 1

Introdução

A evolução tecnológica dos meios de coleta e armazenamento de dados, juntamente com a valorização da importância da informação e conhecimento, propiciam o crescimento vertiginoso de massas de dados alocadas em dispositivos digitais. Grandes quantidades de dados implicam em potencialmente mais informações, entretanto, o armazenamento em larga escala inviabiliza o processo manual de inspeção e aquisição de conhecimento.

Destarte, aumenta-se a necessidade e o interesse sobre tecnologias capazes de manusear inteligentemente —e automaticamente— grandes repositórios de dados. Esta é a tarefa do processo geral de *Descoberta de Conhecimento em Base de Dados* [1], cujo carro-chefe pertence à *Mineração de Dados*.

A mineração de dados é de fato o esforço de extração de conhecimento implícito em base de dados, cujas tarefas variam em função do domínio de aplicação e categoria. Dentre estas encontra-se a classificação de dados, ocupando provavelmente o posto de tarefa melhor estudada e comumente executada. Atribui-se esta popularidade principalmente à sua intensa presença e utilidade em um amplo leque de aplicações. A tarefa de classificação de dados é o alvo de aplicação dos temas do trabalho desta dissertação, por conseguinte, o capítulo corrente prossegue focando-se nos assuntos correlatos.

[1] KDD — *Knowledge Discovery in Databases*. O KDD é um processo de descoberta de conhecimento em banco de dados que envolve determinadas etapas, como o tratamento e enriquecimento de dados, mineração de dados e a interpretação das saídas.

1.1 O Domínio de Aplicação

1.1.1 Classificação de Dados

A classificação de dados consiste em associar um conjunto de atributos (heterogêneos ou não) a um valor categórico, isto é, a uma classe. As classes são valores discretos e finitos, conceitualmente representando, cada uma, um grupo ou divisão que manifesta características semelhantes, de mesma categoria. O fundamento da tarefa de classificação é decifrar, em uma determinada amostra de dados, a relação (conhecimento implícito) entre o conjunto de atributos (valores preditivos) e a classe. Sendo assim, a classificação de dados pode ser explorada sob dois vieses:

- como ferramenta destinada tão somente à classificação de novas amostras de dados — usa-se apenas o poder de classificação, dispensando a interpretação do conhecimento adquirido pelo classificador. Por exemplo, um classificador poderia ser utilizado para predizer o grau de risco de empréstimo bancário a um cliente, a julgar pelo seu perfil sócio-econômico. Conclui-se imediatamente que a qualidade simbólica (textual) do "conhecimento" do classificador é irrelevante, isto é, a classificação dita "caixa preta" seria tão satisfatória quanto a interpretativa.

- como fonte de conhecimento — explora-se o esforço da tarefa de classificação de dados como um instrumento de inferência de conhecimento. Por exemplo, na base de dados da planta íris [2], pode-se descobrir/entender a relação que mapeia seus atributos numéricos (representando dimensões da planta) nas três diferentes classes da planta. Naturalmente, exige-se que a técnica empregada de classificação seja simbólica, isto é, passível de interpretação humana.

Percebe-se que o princípio do aprendizado "associativo" da classificação de dados é supervisionado. Isto significa que a inferência de conhecimento é patrocinada por conjunto de dados previamente classificados (corretamente), cujo propósito é servir como guia de aprendizado, sendo portanto referidos como conjunto de treinamento. Para este aprendizado, pode-se fazer uso de distintas técnicas, entre elas as redes neurais artificiais, métodos estatísticos e algoritmos da computação evolucionária, em especial a programação genética.

[2] Vide Seção 5.2.

Nesta dissertação, adota-se a técnica de classificação de dados via programação genética, integrada ainda a dois importantes componentes: a programação genética gramatical e a co-evolução amostra-classificador.

Introduz-se no decorrer deste capítulo os conceitos que servem a esta dissertação, juntamente com as motivações da adoção destas abordagens.

1.2 Os Três Pilares

1.2.1 Programação Genética

A natureza tem se mostrado uma rica fonte de inspiração para as mais diversas ferramentas e técnicas computacionais, seja no âmbito biológico, físico ou químico. O complexo biológico da natureza tem sido ultimamente bem explorado por cientistas e pesquisadores, sendo que grande parte da pesquisa incide sobre a área de otimização.

Um importante paradigma concebido às custas da "inteligência" biológica, em especial da evolução natural das espécies, é a computação evolucionária. Neste, o processo de otimização apóia-se no princípio darwiniano de evolução (essencialmente a seleção natural), isto é, uma população de indivíduos artificiais, cada qual representando uma solução candidata para um determinado problema, é posta em evolução simulada. Espera-se assim que, ao longo das gerações, esta população progressivamente contenha indivíduos melhor adaptados à tarefa objetivo, de maneira análoga ao processo de evolução natural.

Dentro da computação evolucionária situam-se técnicas como os algoritmos genéticos e a programação genética [3], apresentando-se como as mais populares e estudadas deste paradigma, mérito este justificado fundamentalmente pela excelência na atuação em variadas áreas do conhecimento.

Distingue-se a programação genética (PG) dos algoritmos genéticos (AG) basicamente pela a estrutura de codificação de seus indivíduos, culminando em propósitos de atuação bem distintos. Enquanto o AG emprega na sua forma canônica um vetor de bits para representar os indivíduos, a PG adota uma estrutura mais complexa, uma estrutura com qualidades para codificar um programa de computador —a árvore é comumente uti-

[3] Os algoritmos genéticos e programação genética foram introduzidos por John Holland [28] e John Koza [34], respectivamente.

lizada. Portanto, a PG visa evoluir programas em uma determinada linguagem, soluções que dependam, por exemplo, de estruturas condicionais, desvios, laços de repetição, operadores relacionais e aritméticos. Percebe-se pelo seu propósito que a aplicabilidade da PG é vasta, virtualmente ilimitada. Dentre inúmeros outros domínios abordados pela PG, estão a regressão simbólica, reconhecimento de padrões, compactação de imagens, classificação de dados e robótica.

Neste trabalho, a programação genética é submetida à tarefa de classificação de dados. A motivação de seu emprego deriva das propriedades intrínsecas à PG, que a tornam adequada às nuanças e exigências da classificação de dados. A PG tem a acrescentar a esta área por ser uma técnica de otimização predominantemente global, relativamente insensível a ruídos, que evolui soluções simbólicas (passíveis de interpretação) e naturalmente afável para com outras técnicas e extensões.

1.2.2 Programação Genética Gramatical

À medida que problemas demandando estruturas representativas mais complexas são abordados pela PG clássica, começam a se evidenciar limitações no que tange à forma tradicional de codificação da PG. Estas fraquezas dão-se pela maneira rudimentar e simplória pela qual a representação típica de uma solução é feita na PG. Dentre estas destacam-se: 1) o fato de que não há imposição quanto às restrições estruturais (semânticas) envolvendo cada operador e operando, em qualquer nível [4]; 2) a restrição de que, para se manter a factibilidade da solução candidata, é necessário que todos operadores e operandos sejam intercambiáveis entre si, isto é, qualquer operador obrigatoriamente deve manipular qualquer operando, mesmo que não haja significado na operação.

Para a tarefa de classificação de dados, onde não é raro a atuação em base de dados contendo atributos heterogêneos (atributos lógicos, ordinais, categóricos e/ou reais), as fragilidades da implementação tradicional tornam não apenas a aplicabilidade deficiente —provavelmente incorrendo em soluções sub-ótimas—, como muitas vezes inviável.

A estrutura formal conhecida como gramática livre de contexto (GLC), gerando qualquer complexidade de linguagem computacional, é uma forte candidata a substituir a forma típica de codificação da PG. A GLC permite que sejam feitas restrições de qual-

[4] Outrossim, a codificação canônica da programação genética não garante nem mesmo restrições de primeiro nível, isto é, restrições quanto à compatibilidade de tipos entre funções e seus argumentos.

quer nível entre qualquer operador e operando. Isto significa o resguardo à semântica de qualquer operação, como por exemplo, garantir que operadores do tipo lógico atuem somente sobre operandos lógicos. Uma conseqüência direta destas restrições é que se limita o espaço de busca, portanto, o processo evolucionário descarta *a priori* regiões estéreis de busca. Ainda, a adoção da GLC faz florescer uma nova direção de otimização, a evolução concorrente da própria gramática [5]. Em outras palavras, torna-se plausível a re-adequação da linguagem que dita a forma do espaço de busca, de acordo com os "palpites" alimentados pela medida de proficiência dos indivíduos.

Diante das qualidades pertinentes à GLC, justifica-se satisfatoriamente o uso desta estrutura em substituição ao modelo tradicional de codificação de indivíduos. Denomina-se, ainda, programação genética gramatical, ou simplesmente PGG, a união entre a programação genética e o conceito de gramática.

1.2.3 Co-evolução Amostra-Classificador

Medir a aptidão de indivíduos, na tarefa de classificação de dados, significa avaliar a solução candidata confrontando-a com todas as amostras do conjunto de treinamento. Para problemas com dimensões modestas, esta definição de avaliação pode trazer uma sobrecarga aceitável. Todavia, à medida que empregam-se base de dados em larga escala, a definição típica da avaliação deixa de ser meramente um fator desconfortável para tornar-se um verdadeiro gargalo no processo evolucionário.

Pode-se pensar em criar algumas variações do modelo padrão, por exemplo, definir uma "janela" contendo uma determinada fração de amostras de dados selecionadas aleatoriamente, então, o indivíduo seria avaliado não pela base inteira, mas sim por esta pequena parcela. É provável que este modelo traga alguma redução no custo computacional para bases de dados mais homogêneas, onde uma fração de dados tende a ter uma representatividade próxima à base completa. Contudo, sua vantagem tende a desaparecer à medida que a "entropia" aumenta, isto é, quando a base de dados torna-se mais diversificada.

Felizmente, uma abordagem mais eficiente e elegante pode ser alcançada através da competição por "recursos", isto é, pela co-evolução competitiva, ou ainda melhor, co-evolução amostra-classificador. Em termos gerais, a co-evolução competitiva é uma variação inteligente do modelo "janela" descrito anteriormente.

[5] A evolução da gramática não foi implementada nesta dissertação.

No modelo co-evolucionário amostra-classificador, empregam-se duas populações de indivíduos, a primeira contendo indivíduos classificadores e a segunda indivíduos encapsulando amostras de dados [6]. São realizadas então avaliações bidirecionais, onde a aptidão do classificador é calculada, naturalmente, por quão bem classifica um conjunto de dados; por outro lado, um indivíduo amostra é avaliado pela sua habilidade em induzir más classificações na população concorrente. A seleção dos adversários durante a avaliação também é direcionada, isto é, favorece indivíduos melhores adaptados.

Toda essa dinâmica cria uma tensão saudável (sob o aspecto evolucionário), trazendo conseqüências positivas, notoriamente:

- focalização na "resolução" de amostras de dados difíceis de serem classificadas, em vez de persistir em casos trivialmente classificáveis.
- estimulação da escalada evolucionária, isto é, à medida que as populações tornam-se cada vez mais adaptadas, exige-se que estas evoluam artifícios capazes de sobrepujarem-se alternadamente.
- inibe a perda da diversidade e conseqüente estagnação em mínimos locais, pois elimina a vantagem evolutiva (e a disseminação de material genético sub-ótimo) de classificadores que não provem, insistentemente [7] , quão bom o são.

1.3 Organização da Dissertação

Esta dissertação está dividida da seguinte forma. No Capítulo 2 é introduzida a noção do conceito darwiniano de evolução, apresenta-se sucintamente os algoritmos genéticos e, então, descreve-se a programação genética. Define-se no Capítulo 3 a teoria formal pertinente à gramática livre de contexto, é realizada a revisão da literatura relacionada, e prossegue-se com os detalhes da integração entre a gramática e a programação genética. É visto no Capítulo 4 a definição de co-evolução competitiva, bem como a discussão de trabalhos prévios e a explanação da co-evolução amostra-classificador. Os conceitos apresentados são então postos em campo no Capítulo 5, onde implementa-se um sistema de

[6] Embora a segunda população não modifique o material genético de seus indivíduos, pois do contrário descaracterizariam os dados a serem preditos.

[7] Nota-se que quanto melhor um indivíduo se apresenta, probabilisticamente são maiores suas chances de seleção para confrontar-se, justamente, com adversários cada vez mais árduos.

classificação de dados e o mesmo é aplicado sobre diversas base de dados, comparando-se ainda os resultados com os de outros algoritmos de classificação.

Finalmente, no Capítulo 6 enunciam-se avaliações sobre as abordagens empregadas neste trabalho juntamente com menção às vantagens e desvantagens de cada método, aponta ainda direções para trabalhos futuros.

Capítulo 2

Programação Genética

A computação evolucionária é uma área inspirada na evolução biológica que agrega sub-áreas especializadas em diferentes vertentes da evolução natural.

Dois importantes ramos da computação evolucionária são os algoritmos genéticos e a programação genética, ambos baseados no princípio darwiniano de evolução. Estas duas abordagem são temas deste presente capítulo, privilegiando-se entretanto, a programação genética, que é parte integrante do objeto de estudo desta dissertação.

Por compartilhar do mesmo embasamento e origem, o capítulo inicia-se com uma visão geral sobre os algoritmos genéticos, construindo-se assim a base necessária para o ingresso, suave, no universo da programação genética.

No contexto da programação genética, discute-se sua relação com os algoritmos genéticos, bem como os conceitos específicos sobre sua natureza.

2.1 A Teoria de Darwin

Charles Darwin, em 1859, publicou um livro intitulado "Origem das Espécies por meio da Seleção Natural" [15], como resultado de vários anos de pesquisa como naturalista. Darwin dedicou parte de sua vida em uma viagem pelo globo terrestre, estudando a relação das espécies com o seu meio-ambiente. Pôde ele concluir que indivíduos melhores adaptados a um certo ambiente têm mais chance de sobrevivência, conseqüentemente, estes se reproduzirão com mais freqüência, repassando suas características a seus descendentes. Isto é conhecido como Seleção Natural. Ao progresso das gerações, a tendência é

que as características dos melhores indivíduos sejam disseminadas de tal modo que seus descendentes as absorvam, tornando-os, com alta probabilidade, tão ou melhor adaptados em relação a seus progenitores. Assim, ao longo das gerações, é esperado que uma convergência ocorra, ou seja, os indivíduos terão características em comum. A teoria da seleção natural é utilizada por virtualmente todos os algoritmos evolucionários.

2.2 Algoritmo Genético — AG

O algoritmo genético[28] é uma técnica de otimização aplicável em domínios contínuos e/ou discretos, robusta, e que dispensa imposições de continuidade, convexidade e diferenciabilidade da função objetivo. Em sua forma canônica reúne o poder da teoria da evolução e a genética, simulando-os artificialmente no contexto da computação. A idéia consiste em, inicialmente, gerar aleatoriamente uma população de indivíduos. Estes indivíduos são, como é de se esperar, virtuais. Isto é, eles representam computacionalmente um candidato a uma solução para um problema em questão, como por exemplo, valores arbitrários de variáveis numéricas candidatos à raiz de uma equação de algébrica. Então, a cada indivíduo, é atribuída uma *nota*, que corresponde ao seu nível de proficiência —nas equações algébricas a nota seria proporcional a quão bem o "indivíduo" (valor da variável) se aproxima da raiz da equação. Em outras palavras, uma função de avaliação determina a qualidade da solução candidata. Esta nota é comumente referida como *aptidão*. Assim, em conformidade com a teoria da seleção natural, os melhores indivíduos (os que possuem melhor aptidão), terão mais chance de serem selecionados para que possam recombinar seus materiais genéticos (cruzamento). Geralmente são selecionados por iteração pares de indivíduos (pais) para o acasalamento, cada qual concebendo uma nova prole, sendo esta composta de uma mistura dos genes de seus pais. Os descendentes, por sua vez, passam a constituir uma nova geração, que podem ou não conviver com indivíduos das gerações anteriores. O processo é então repetido até que algum critério de parada seja satisfeito, tal como número máximo de gerações (iterações), tempo de processamento, convergência da população, entre outros.

2.2.1 Reprodução

A reprodução é o processo através do qual indivíduos são inseridos nas sucessivas gerações. Pode-se destacar dois modos distintos de reprodução utilizados nos AGs, o *geracional* e o *steady-state*.

A reprodução denominada *geracional* mantém indivíduos pais e filhos em populações distintas, evitando-se assim uma convivência simultânea e conseqüente troca de material genético.

Por outro lado, a reprodução dita *steady-state*, ou em "regime permanente", permite que pais e filhos convivam na mesma população, isto é, pode ser que ocorram cruzamentos entre antepassados e seus descendentes.

Um adicional a estes modos de reprodução, que pode agregá-los, é o chamado reprodução com *elitismo*. O objetivo é manter intacto pelo menos o melhor indivíduo da população, garantindo assim a preservação da melhor solução até então encontrada. Além disso, este é o indivíduo que sempre terá a maior chance de ser selecionado para a criação de novas proles (pois é o mais apto), assim, a tendência é que suas características (que tendem a ser boas) sejam suficientemente disseminadas ao longo da população. O problema deste artefato é a estimulação da convergência prematura, isto é, um indivíduo que seja substancialmente superior aos demais poderá dominar a população, espalhando rapidamente suas características genéticas, provavelmente estagnando o processo evolutivo em um ótimo local.

2.2.2 Representação

No que concerne à representação (codificação) de uma solução candidata, pode-se dizer que o AG, em sua forma original, utiliza um vetor de *bits* (codificação binária) de tamanho fixo. Na verdade, cada indivíduo possui este vetor de *bits*, que pode ser entendido como seu *cromossomo* [1], ou seja, aquele que armazena seu material genético. Para exemplificar esta codificação, suponha que deseja-se procurar as raízes de uma equação do segundo grau. Neste caso, o vetor de *bits* está codificando dois valores de raízes candidatas (números inteiros ou reais) através, é claro, de *zeros* ou *uns*. Cada posição deste

[1] O termo genoma muitas vezes é empregado como sinônimo de cromossomo na área dos algoritmos evolucionários.

vetor (cromossomo) é conhecida como *locus*. Um conjunto de *locus* que se traduz em alguma característica do indivíduo é denominado *gene*. Chama-se de *alelo*, todo valor que um gene pode assumir. Para esclarecer tais conceitos, é apresentado um exemplo simplificado na Figura 2.1 que ilustra as partes que constituem um cromossomo. Nesta, é possível observar que o cromossomo contém seis *locus*. Esse cromossomo está dividido em dois genes. Uma interpretação para isto pode ser dada, imaginando-se que cada gene representa um número inteiro não negativo de *zero* à *sete*[2]. Estes números inteiros (ou reais, por mapeamento) podem, por exemplo, ser os valores candidatos às raízes de uma equação de segundo grau. Na figura, os alelos são todos os valores que um gene pode assumir; então, no exemplo, eles são os números $\{0,1,2,3,4,5,6,7\}$.

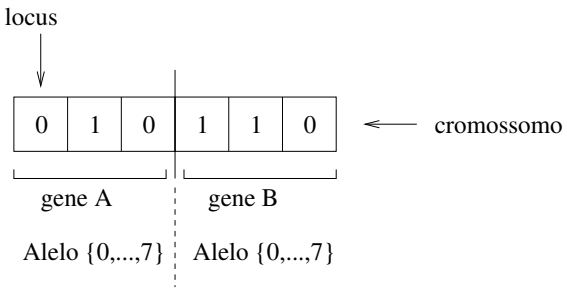

Figura 2.1: As partes de um cromossomo.

Além do padrão binário de codificação, existem vários outros que podem ser utilizados pelo AG, como codificação inteira, real, entre outras. A forma de representação a escolher depende do problema. Não serão descritas neste trabalho outras formas de codificação porque fogem do escopo e objetivo [3].

2.2.3 A Função de Avaliação

A função de avaliação é fundamental para qualquer algoritmo evolucionário, é a responsável por direcionar a busca para locais mais promissores. A função associa a cada indivíduo um valor que corresponde à sua aptidão (qualidade). Deduz-se obviamente, que a função de avaliação é obrigatoriamente dependente do problema que se deseja resolver.

[2]Neste exemplo, codifica-se duas variáveis com três *bits* cada. Assim, o intervalo de valores está compreendido entre 0 ($2^0.0 + 2^1.0 + 2^2.0$) e 7 ($2^0.1 + 2^1.1 + 2^2.1$). Ainda, números reais podem também ser representados (com precisão e amplitude ajustáveis) por conjunto de *bits*, através de um mapeamento adequado.

[3]Contudo, o leitor é convidado a checar as informações constantes em [8], [20], [10], [23], [50] e [25].

Considerando-se o problema de encontrar as raízes de uma equação qualquer, pode-se exemplificar a função de avaliação, visando a minimização, como sendo o inverso da "distância" entre a solução candidata (valor numérico) e o valor procurado, neste caso, zero. Isto é, quanto mais próximo de zero, melhor adaptado é o indivíduo. Um exemplo trivial de uma função de avaliação f_{aval}, para uma equação de primeiro grau, é dado matematicamente por

$$f_{aval}(x) = |ax + b|$$

onde a e b são as constantes numéricas reais da equação e x o valor do fenótipo do indivíduo em avaliação. Qualquer valor para x diferente da raiz da equação implica em um retorno positivo na f_{aval}, que assumindo-se um problema de minimização, promove uma baixa na aptidão do indivíduo —neste caso a aptidão é inversamente proporcional ao valor de f_{aval}.

Enquanto a função de avaliação não necessariamente corresponde de fato a aptidão do indivíduo —como no exemplo anterior, em que os valores eram inversamente proporcionais—, é interessante convencionar, a título de clareza, que a "função de aptidão" limite-se a quantificar tão somente a qualidade do indivíduo, isto é, ser diretamente proporcional à habilidade deste. Nota-se, contudo, que os termos "função de aptidão" e "função de avaliação" são ainda comumente utilizados indiscriminadamente.

2.2.4 Esquemas de Seleção

Quando os indivíduos de uma população estiverem avaliados, são necessários meios que possibilitem aos indivíduos mais aptos uma maior chance de serem selecionados para recombinarem suas características: isto é a seleção natural. São citadas a seguir três formas de seleção.

O primeiro esquema, chamado *seleção por roleta* ou *seleção proporcional à aptidão*, associa ao indivíduo uma probabilidade de ser selecionado proporcional à sua aptidão. O problema deste método, dado seu critério fortemente baseado nas discrepâncias entre aptidões, é facilitar a ocorrência de convergência prematura.

No intuito de amenizar o problema da convergência prematura, é possível utilizar um outro esquema denominado *seleção por ranking*. Esta técnica ordena todos os indivíduos de acordo com sua aptidão, de tal modo que o primeiro no *rank* corresponde ao melhor

indivíduo. Assim, a diferença entre duas soluções candidatas adjacentes será sempre considerada como igual a *um*, ao contrário da seleção proporcional à aptidão.

Um método típico para implementação da seleção *ranking* é atribuir ao melhor indivíduo $1,5$ vezes mais chance de ser selecionado em comparação ao indivíduo mediano [4], mapeando linearmente as probabilidades dos demais membros. Considerando-se tam_{pop} como o tamanho da população, tem-se a função de seleção f_{sel}, retornando o índice do indivíduo selecionado, definida como:

$$f_{sel} = \left\lfloor tam_{pop} \left(1,5 - \sqrt{2,25 - 2,0 \,.\, rand()}\right) \right\rfloor$$

onde a função $rand()$ devolve um número real aleatório x uniformemente distribuído, tal que $0,0 \leq x < 1,0$. Assim, f_{sel} retorna índices (inteiros) entre 0 e $tam_{pop} - 1$.

Finalmente, o último esquema, conhecido como *seleção por torneio*, pode ser empregado nos AGs. Este colhe aleatoriamente um número n de indivíduos da população —comumente dois ou três— para competirem entre si. O vencedor, então, será o escolhido. A seleção por torneio, assim como a seleção por *ranking* abranda o problema da convergência prematura. Todavia, a pressão de seleção pode ser ajustada de acordo com o valor de n, isto é, quanto maior n, mais tendenciosa fica a seleção para os indivíduos mais adaptados.

2.2.5 Operadores Genéticos

Os operadores genéticos, como o próprio nome diz, são os responsáveis por manipular as informações genéticas presentes nos cromossomos. Os principais operadores são o *crossover* e a *mutação*, que são descritos a seguir.

2.2.5.1 *Crossover*

O *crossover* é o principal operador utilizado pelo AG. Usualmente, este operador atua sobre dois indivíduos (pais), produzindo dois novos indivíduos (filhos). O *crossover* é diretamente responsável pela recombinação das características genéticas dos cromossomos dos pais. A forma mais simples do operador *crossover* constitui em emparelhar os

[4] Indivíduo que detém a colocação média do *ranking*.

indivíduos pais e, então, selecionar arbitrariamente um ponto de corte e trocar estas informações, formando a prole. Uma ilustração deste processo pode ser visualizada na Figura 2.2. Os cromossomos progenitores são, na figura, **A** e **B**. Quando o operador de *crossover* é aplicado os códigos genéticos são recombinados, formando então a prole **A'** e **B'**.

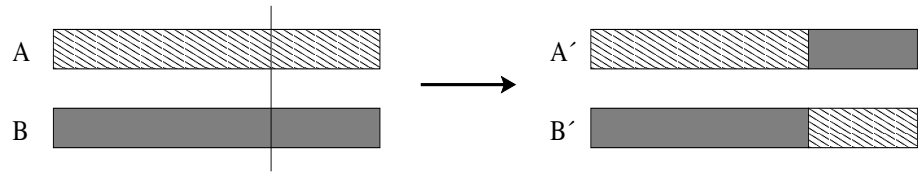

Figura 2.2: O *crossover* de um ponto.

Analogamente, é possível estender o *crossover* de um ponto para quantos pontos desejarmos. Por exemplo, na Figura 2.3, é exibido o *crossover* de dois pontos.

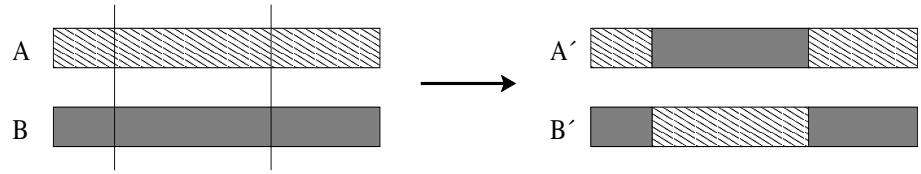

Figura 2.3: O *crossover* de dois pontos.

Um outro tipo de *crossover* é produzido através do uso de uma máscara de *bits*. Uma máscara de *bits*, com o mesmo comprimento dos cromossomos pais, é emparelhada a estes. Então, para cada *locus* contendo o valor 0 (zero), o primeiro e segundo filho recebem o conteúdo do *locus* do primeiro e segundo progenitor, respectivamente. De modo análogo, porém oposto, para o valor igual a 1 (um). A Figura 2.4 mostra claramente este processo.

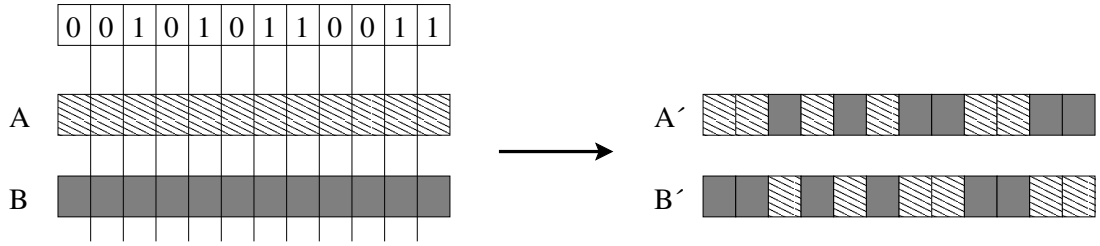

Figura 2.4: O *crossover* com máscara.

Ainda sim, pode-se utilizar um outro *crossover*. Este difere um pouco dos demais, por trabalhar com três progenitores, simultaneamente. Além disso, este gera apenas um filho. Os dois primeiros pais são selecionados por algum método que favoreça os mais aptos, o terceiro é escolhido inversamente, ou seja, favorecendo os indivíduos menos aptos. A idéia consiste em verificar se os materiais genéticos dos pais são os mesmos, se forem, então o filho o receberá. Caso contrário, fica-se em dúvida ao determinar de qual pai a prole receberá o valor do *locus* atual. Entra, então, a função do terceiro pai: como trata-se de um indivíduo ruim, o filho receberá o valor oposto do contido neste *locus*, pois espera-se que o negativo do ruim seja algo bom. Ilustrado na Figura 2.5, tem-se o esquema deste *crossover*, conhecido como *simplex*[5].

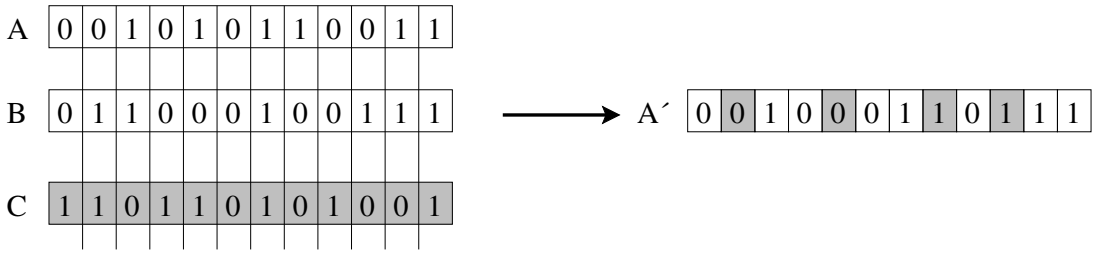

Figura 2.5: O *crossover* simplex.

2.2.5.2 Mutação

Caracterizado como um operador secundário nos AGs, mas de grande importância, o operador de *mutação* auxilia na manutenção da diversidade da população, introduzindo novas soluções e permitindo que o AG explore novas regiões no espaço de busca. O operador de mutação atua sobre um único indivíduo, provocando uma perturbação no seu código genético, sendo que esta alteração é normalmente pequena e aleatória, mas não é incomum encontrar mutações direcionadas na literatura. Na Figura 2.6, ilustra-se um exemplo do operador mutação sendo aplicado sobre um cromossomo. Neste exemplo, um ponto é aleatoriamente selecionado no cromossomo e, então, troca-se o valor do bit correspondente. O cromossomo **A** é o original, **A'** é o cromossomo resultante.

[5]O *crossover* simplex foi inicialmente proposto para codificação binária, como mostrado neste capítulo.

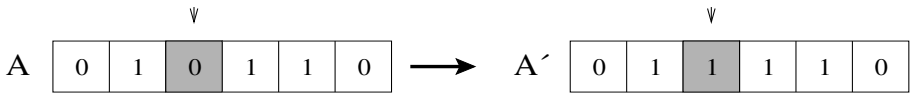

Figura 2.6: Um exemplo de mutação.

2.3 Programação Genética — PG

2.3.1 Introdução

O ser humano, desde as épocas mais remotas, vem tentando automatizar seu trabalho. Em tempos antigos, os homens dispunham apenas de ferramentas manuais, desse modo, as ferramentas necessitavam de uma ação direta e ininterrupta do trabalhador que as manuseava. Pouco a pouco máquinas foram elaboradas, e ao passar dos tempos, elas foram aperfeiçoadas, o que se traduzia em maior rapidez e automação para um trabalho que antes era desenvolvido manualmente. O advento da era tecnológica, especificamente o surgimento da Informática e afins, trouxe consigo uma nova necessidade de mão-de-obra, o programador de sistemas. O programador de sistemas é um trabalhador que desenvolve sistemas computacionais de forma manual ou auxiliado por alguma ferramenta, participando quase integralmente de todo o processo de desenvolvimento. Insatisfeitos, os cientistas, assim como os trabalhadores de épocas passadas, têm almejado formas autônomas de programação, denominadas auto-programação.

A programação genética foi desenvolvida/evoluída com o intuito de abster o programador do processo manual de criação de sistemas computacionais, ou seja, a PG está intimamente relacionada ao conceito de auto-programação —naturalmente limita-se ainda à cobertura de segmentos específicos dentro deste complexo e pretensioso paradigma.

A invenção da meta-heurística PG foi atribuída a *John Koza* [34], mas não é certo omitir que esforços anteriores foram a base para o atual estado da PG [7], incluindo, primeiramente, a colossal descoberta de Darwin e também o advento dos algoritmos genéticos.

A PG é relativamente uma abordagem nova e, como na natureza, está em processo de evolução, sofrendo refinamentos e aperfeiçoamentos. Esta área está sendo alvo de grandes pesquisas e certamente tem muito a acrescentar, não somente à computação, mas estende-se a todas as áreas dependentes destas, bem como ainda instrumento para o me-

lhor entendimento da complexidade biológica da natureza.

2.3.2 Os Conceitos da Programação Genética

A PG pode ser entendida como uma extensão/especialização do AG, onde a codificação binária passa a ser uma codificação capaz de representar, na sua forma geral, um programa de computador. O cerne da PG funciona da mesma forma como no AG, basicamente: inicializar aleatoriamente uma população de indivíduos, avaliá-los, selecionar os mais aptos, cruzá-los, avaliá-los, e assim repete-se, até que algum critério de parada seja alcançado. Dentre as propriedades que podem ser compartilhadas por ambas as meta-heurísticas, pode-se destacar:

- idéia central, oriunda da teoria de evolução e seleção natural;
- criação/inicialização da população de modo aleatório;
- esquema de reprodução geracional e *steady-state*;
- técnicas de seleção, como roleta, *ranking* e torneio.

No AG, um vetor de *bits*[6] é a estrutura capaz de representar uma solução candidata. Entretanto, a programação genética necessita, como dito anteriormente, representar programas de computador. Um vetor de *bits* não seria adequado para esta necessidade [7]. Felizmente, sabe-se que a estrutura árvore é capaz (e ideal) de armazenar um algoritmo computacional e, na sua forma usual, a PG a utiliza[7]. Representações alternativas[7], que não a árvore, mas que também suportam a codificação de programas de computador, podem ser empregadas na PG, como por exemplo, um grafo orientado, um conjunto de seqüências de instruções de máquina, entre outras. O restante do presente capítulo concentra-se exclusivamente na estrutura de codificação de uma solução candidata baseada em árvore.

O fluxograma ilustrado pela Figura 2.7 fornece uma visão da estrutura de um sistema de programação genética.

[6]Considera-se tratar do algoritmo genético que emprega a codificação binária usual.

[7]Estritamente falando, um programa de computador é, em sua última instância, representado por *bits*, na línguagem digital da máquina. Entretanto, *bits* não são próprios para codificarem programas que serão manipulados, no nível semântico, por operadores genéticos.

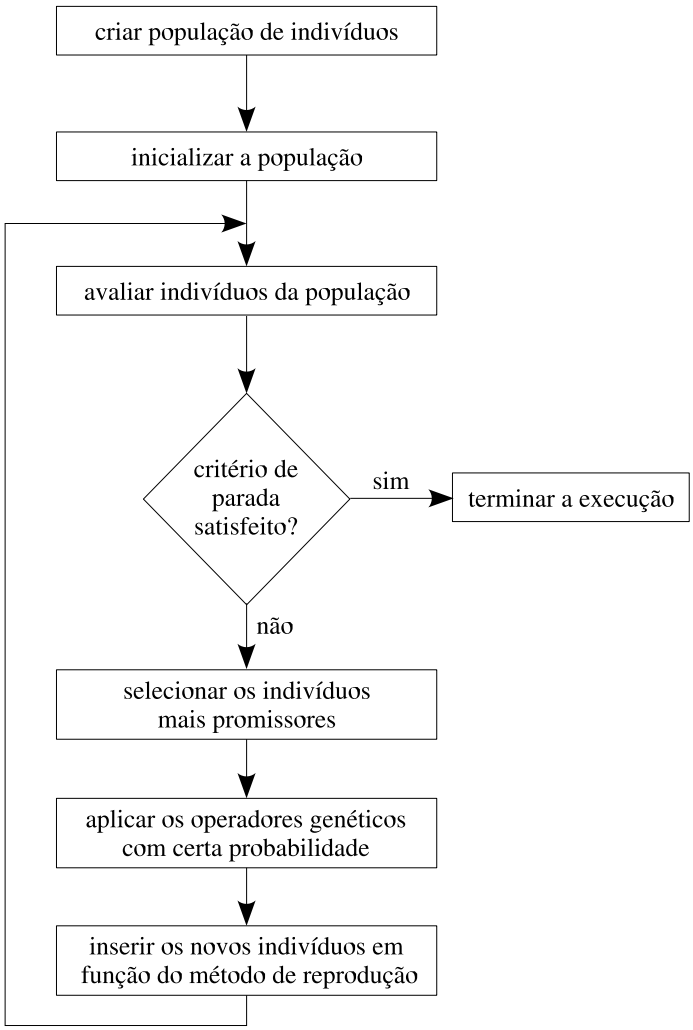

Figura 2.7: A estrutura de um sistema de programação genética.

O fluxo do processo evolutivo inicia-se com a criação seguida da inicialização da população de indivíduos. A população passa então pela avaliação e verifica-se se o critério de parada foi alcançado, isto é, se na população encontra-se indivíduos satisfatoriamente adaptados. Caso negativo, o algoritmo prossegue com a aplicação dos operadores de seleção, favorecendo os mais aptos, igualmente como nos algoritmos genéticos. Então, aplicam-se as recombinações e demais operadores genéticos sobre os indivíduos previamente selecionados. Os descendentes são assim concebidos e inseridos na população (ou em uma nova), de acordo com a política de reprodução. O fluxo entra portanto em um laço iterativo, retornando ao ponto de avaliação (sobre os novos indivíduos).

Os detalhes pertinentes das peculiaridades que dizem respeito à programação genética são descritos e elucidados no decorrer deste capítulo.

2.3.3 Codificação e Representação

Como dito antes, a PG pode suportar várias estruturas para codificar uma solução candidata. Entretanto, a PG, em sua forma de representação original, faz uso da estrutura árvore. A seguir, discute-se a relação da árvore com a PG, bem como a forma em que os operadores genéticos atuam sobre esta estrutura.

Os indivíduos agora têm o cromossomo como sendo uma árvore genérica, de tamanho (ou profundidade) variável. Em outras palavras, cada indivíduo da população, por ser um programa de computador e, por serem distintos, terão, naturalmente, tamanhos diferentes. Por exemplo, na Figura 2.8, tem-se dois indivíduos representando programas de computador (soluções candidatas), sob um dado domínio de aplicação.

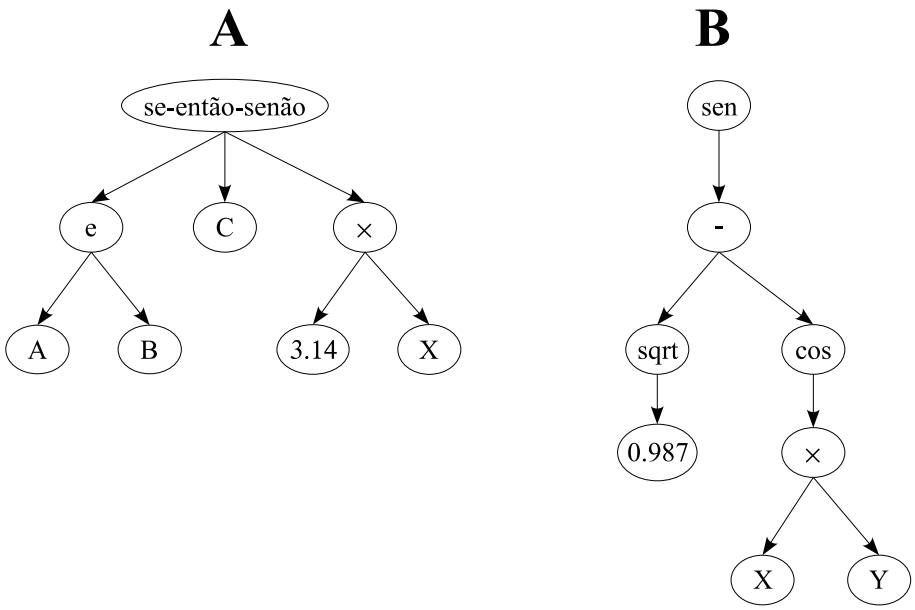

Figura 2.8: Dois exemplos de indivíduos.

2.3.4 Criação da População

Naturalmente, para que seja possível evoluir algum ser/organismo, este deverá antes ser criado, mesmo que seja um organismo simples (não especializado). A programação genética não foge à regra e necessita, obviamente, que uma população seja construída.

Basicamente, esta população inicial é criada de forma aleatória, assim como nos algoritmos genéticos. Entretanto, árvores são estruturas significativamente mais complexas

do que um vetor de *bits*, por exemplo. Quanto melhor for a população inicial, mais rápida e provável será a convergência para uma solução ótima. Pensando nestes fatores, são mostrados aqui alguns métodos[34] para a definição da estrutura do cromossomo (árvore) de cada indivíduo, no momento da criação.

2.3.4.1 Método de Criação Completa – *Full Creation Method*

Na PG é interessante que se imponha limites nos tamanhos dos cromossomos (árvores), evitando assim, que as soluções candidatas cresçam indefinidamente, provocando esgotamento de memória e degradação no desempenho. Um destes limites está relacionado ao momento da criação dos indivíduos. Geralmente é especificado por profundidade máxima [8].

Neste método, todo comprimento entre o vértice raiz e qualquer nó terminal é sempre igual à profundidade máxima especificada. Isto significa que, até se alcançar a profundidade máxima, todos os vértices terão um ou mais filhos (funções com mais de um argumento) e, ao se atingir a profundidade determinada, os vértices não possuirão filhos (funções sem argumentos, variáveis ou constantes).

2.3.4.2 Método de Criação Livre – *Grow Creation Method*

Diferentemente do método completo, este permite que vértices com qualquer número de filhos (argumentos) sejam adicionados (aleatoriamente) no momento de criação, assim, não é necessário que a árvore seja completa. Isto é, se ocorrer em algum ramo um vértice sem filhos (terminal), este ramo será encerrado.

2.3.4.3 Efeito Escada – *Ramping*

No momento da criação, a população é dividida em segmentos. A cada segmento é atribuído uma profundidade máxima, de modo que esta profundidade máxima varie entre a profundidade mínima real e a máxima real. Esta atribuição geralmente é feita em ordem crescente. Por exemplo, considere uma população com 50 indivíduos, e suponha que a profundidade mínima seja 2 e a máxima 6. Então, a população seria dividida em cinco

[8] Profundidade é o maior comprimento entre o vértice raiz e os demais vértices de uma árvore, sendo comum a especificação de algum limite mínimo.

segmentos, cada qual com dez indivíduos. Assim, o primeiro segmento receberia uma profundidade máxima de 2, o segundo 3, e assim, até que o último receba a profundidade 6.

2.3.4.4 Método de Criação Misto – *Ramped Half-and-Half*

Segundo *Koza*[34], este é o método mais recomendado. É capaz de gerar uma ampla diversidade de indivíduos, produzindo várias formas e tamanhos distintos de árvores. A idéia consiste em fazer com que cada metade de um segmento produzido por "*ramping*" seja criada ou com o método completo (*full*) ou com o método livre (*grow*). Isto é, metade de um segmento será criado pelo método completo e a outra metade com o método livre.

2.3.5 Conjunto de Funções e Terminais

A programação genética trabalha em diversos domínios de aplicações, como por exemplo, regressão simbólica, classificação de dados e imagens, robótica, desenvolvimento de circuitos elétricos, entre outros. Cada área de atuação requer que funções/procedimentos sejam escolhidos adequadamente. Portanto, caso esteja-se trabalhando sobre um domínio de aplicações que é puramente lógico, é desejável que se tenha operadores tais como **e**, **ou**, **neg** e **xou**[9], e operandos como **verdadeiro** e **falso**. Para isto, a PG possui dois conjuntos distintos, denominados *conjunto de terminais* e *conjunto de funções* [10]. As funções e terminais são as primitivas a partir das quais um programa é criado em PG.

O conjunto de terminais, denotado por T, é responsável por armazenar *constantes*, *variáveis* e *funções que não requerem argumentos*. Um exemplo poderia ser $T = \{x, y, 1.0, 2.0, \pi\}$, ou talvez $T = \{verdadeiro,\ falso\}$, e ainda $T = \{$*vire-à-esquerda*, *vire-à-direita*, *siga-em-frente*$\}$, entre outros.

O conjunto de funções, F, contém *sentenças*, *condicionais* e *funções que possuam argumentos*. Como exemplo, poderia-se supor que o conjunto fosse $F = \{+, -, *, /,$ *sen*, *cos*, *tan*, *exp*$\}$, ou $F = \{$*se-então-senão*, *menor*, *caso-igual*, *faça-enquanto*$\}$.

[9] O operador **xou** é o operador lógico "*ou exclusivo*".

[10] A fim de evitar possíveis confusões, é conveniente lembrar que programação genética gramatical (Capítulo 3) trata diferentemente o significado dos conjuntos funções e terminais.

2.3.5.1 Suficiência e Consistência

Koza [33] definiu duas propriedades de relevada importância para a viabilidade de um sistema baseado em PG: suficiência e consistência.

A suficiência dita que a solução do problema deverá estar no domínio do espaço criado por toda combinação possível entre os conjuntos funções e terminais. Em outras palavras, as combinações factíveis entre funções e terminais devem ser capazes de representar a solução.

Consistência é a condição de integridade, requer que cada função do conjunto de funções seja flexível o suficiente para aceitar como argumento qualquer valor produzido dentro das possibilidades combinatórias dos conjuntos funções e terminais, sendo assim, qualquer função ou terminal poderia ser argumento de qualquer outra função. Em outras palavras, os conjuntos terminais e funções têm que ser tais, que qualquer combinação significativa[11] entre F, T, e o próprio F, seja válida. Esta propriedade também é conhecida como *fecho*.

A consistência é uma proposta extremamente relaxada das restrições da linguagem das soluções candidadas. Ao mesmo tempo que provém uniformidade, traz consigo certos empecilhos [12]:

- há um aumento improdutivo no espaço de busca, pois são criadas alternativas redundantes que não seguem a estrutura do problema;
- dificuldade de satisfazer a condição de consistência quando estão envolvidos diferentes tipos e estrutura de dados, tornando-a inviável ou na melhor das hipóteses forçando uma adaptação crassa.

2.3.6 Inicialização da População

A criação e a inicialização podem, à primeira vista, parecer sinônimos. Entretanto, no contexto da PG, a criação corresponde à formação estrutural de um indivíduo (cromossomo). Diferentemente, a inicialização está relacionada à definição do conteúdo de um

[11] Combinação significativa quer dizer uma combinação que possua uma semântica. Por exemplo, não haveria sentido uma combinação entre dois terminais.

[12] O Capítulo 3 traz uma abordagem elegante e requintada, capaz de suprir estas deficiências, ao mesmo tempo que agrega outras vantagens.

cromossomo. Na Figura 2.9, tem-se uma árvore *criada* (**A**), com seu tamanho e forma definidas e, em **B**, esta mesma árvore, agora *inicializada*.

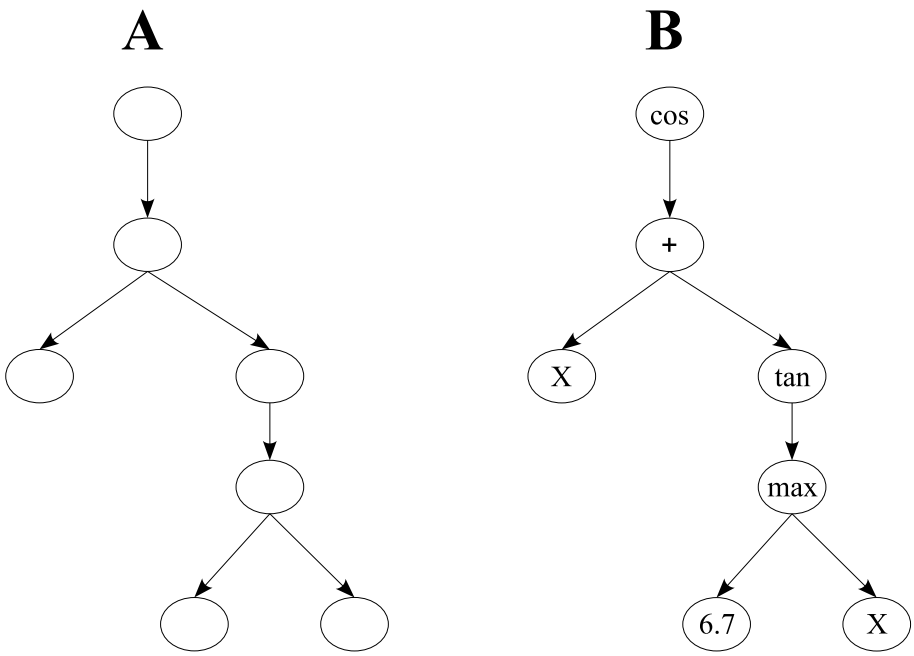

Figura 2.9: Criação × inicialização.

Quando inicializa-se um cromossomo, depara-se com a estrutura já definida. Assim, sabe-se *a priori* o número de argumentos que cada função requer. Se o número de argumentos for igual a zero (terminal), então, escolhe-se um elemento, aleatoriamente, dentre o conjunto de terminais T e o atribui ao vértice. Caso contrário, de maneira análoga, escolhe-se um elemento qualquer sobre o conjunto de funções F e o atribui ao vértice.

2.3.7 Funções de Avaliação

As funções de avaliação são fortemente dependentes da área de aplicação. Como a programação genética atua em incontáveis domínios, seria impossível descrevê-las, mesmo que resumidamente. Porém, um ingrediente essencial às funções de avaliação é o denominado *conjunto de treinamento*. Estes formam os dados que serão confrontados com os resultados obtidos pela solução candidata, durante o processo de avaliação. Por exemplo, na regressão simbólica (vide Seção 2.4), este conjunto de treinamento é composto por pontos, distribuídos em um intervalo. Assim, a aptidão é calculada em função da comparação entre os valores obtidos e os esperados.

Geralmente, o conjunto de treinamento é limitado, ou seja, ele não é suficiente para cobrir toda a faixa de resultados que uma solução candidata pode gerar. Além disso, o custo computacional é proporcional à cardinalidade do conjunto de treinamento. Portanto, uma boa/desejada função de avaliação também agrega recursos capazes de medir o potencial de uma solução candidata, além do conjunto de treinamento. Isto é, uma função de avaliação que somente enxerga o conjunto de treinamento é dita míope e seu poder de generalização[13] é fraco.

2.3.8 Operadores Genéticos

Assim como os AGs, a PG necessita de métodos que possam recombinar/modificar os materiais genéticos da população. Porém, como manuseia-se indivíduos que codificam árvores, é necessário ter técnicas específicas para tal, como as descritas a seguir.

2.3.8.1 *Crossover*

O operador de cruzamento (*crossover*) pode ser realizado trocando-se sub-árvores dos indivíduos pais. Para cada cromossomo, escolhe-se, aleatoriamente, um ponto. Então, as sub-árvores referentes aos respectivos pontos são comutadas, gerando dois novos indivíduos. Graficamente, é possível ilustrar o processo de *crossover* como mostrado na Figura 2.10. Os indivíduos demarcados por **A** e **B** são os que recombinarão suas características genéticas, ou seja, os pais. Um "ponto" (sub-árvore) arbitrário é escolhido para ambos progenitores. O processo de *crossover* é então aplicado, trocando-se as sub-árvores, formando os indivíduos **A'** e **B'**.

2.3.8.2 Mutação

Um outro operador genético é o chamado mutação. A idéia é a mesma do operador sob o contexto dos AGs, ou seja, provocar uma perturbação na informação genética. Este operador é o responsável pela introdução de novos materiais genéticos, produzindo uma maior diversidade, o que significa que novas regiões do espaço de busca poderão ser

[13]Entende-se como poder de generalização, a capacidade de uma solução agir eficientemente em domínios não submetidos a testes pela função de avaliação. Por exemplo, em um problema de classificação/ordenação de números, se o conjunto de treinamento apenas engloba números compreendidos entre 0 e 10, espera-se que o algoritmo encontrado seja capaz de ordenar números além destes, como por exemplo de 0 a 100.

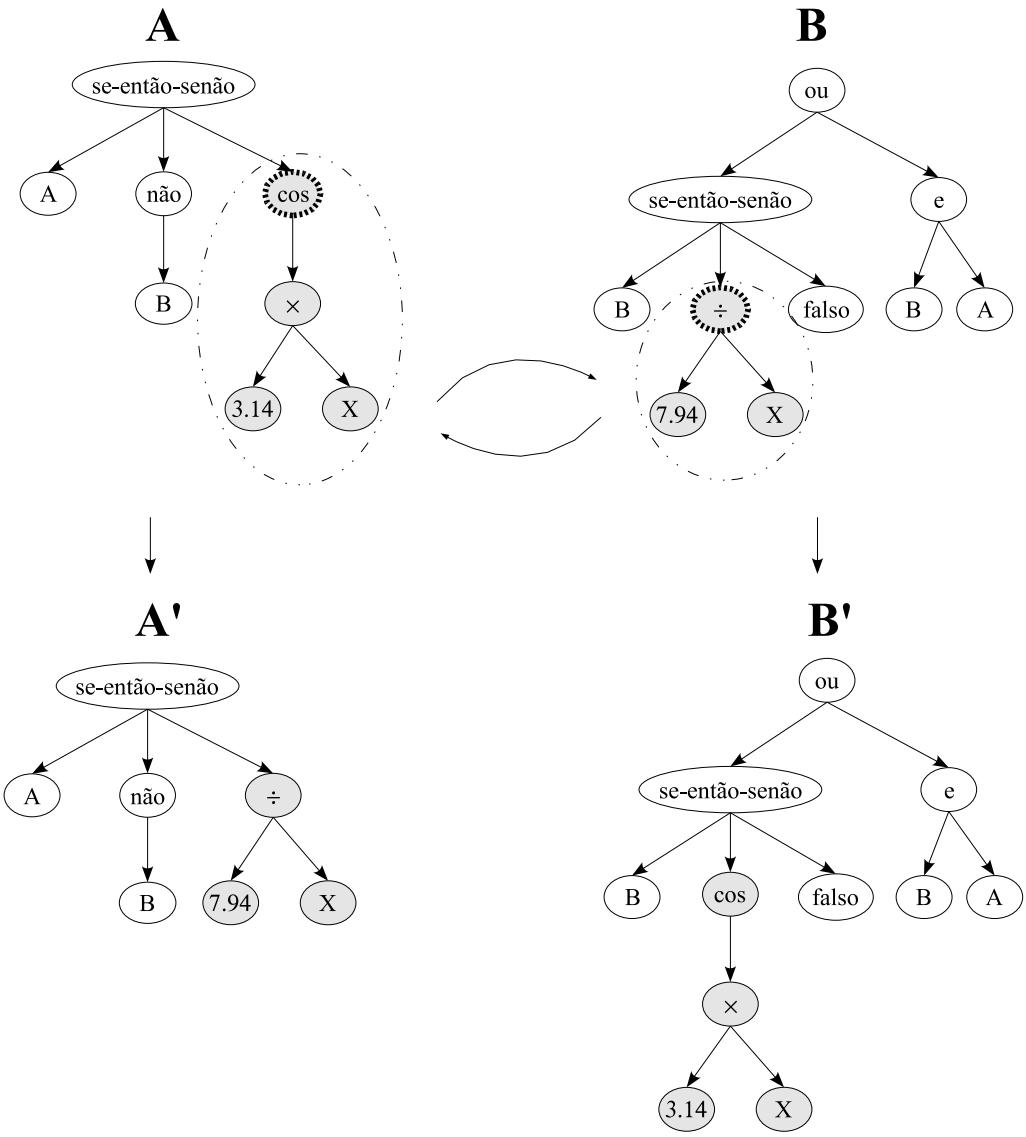

Figura 2.10: O processo de *crossover* padrão.

exploradas. Na PG, a mutação pode ser implementada da seguinte forma. Selecione, aleatoriamente, uma sub-árvore. Exclua-a e, então, gere uma nova sub-árvore e a introduza no ponto onde foi retirada a outra. Na Figura 2.11 tem-se um exemplo da aplicação do operador de mutação. O cromossomo **A**, após o processo, é transformado em **A'**.

Não obstante, pode-se ainda empregar duas variações da mutação usual, denominadas *mutação alelo* e *mutação encolhimento* [14]. A mutação alelo, como o nome indica, provoca uma alteração sobre algum alelo. Assim, após selecionado um elemento, arbitrariamente, este é substituído por algum outro, de tal forma que o novo alelo possua o mesmo número

[14] *Shrink.*

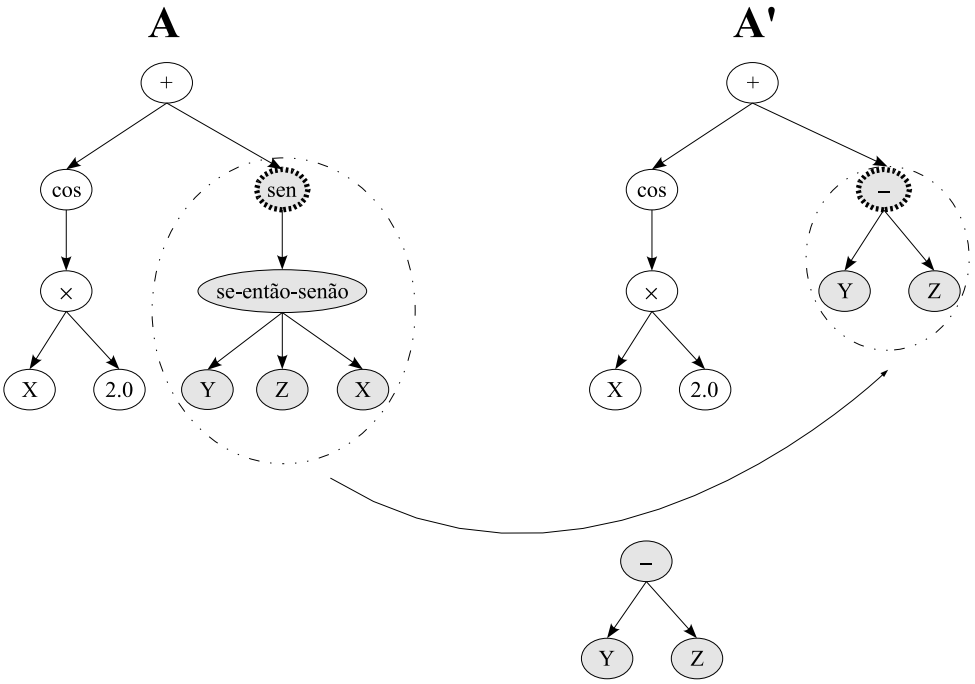

Figura 2.11: O processo de mutação padrão.

de argumentos que o original. A Figura 2.12 ilustra graficamente este processo.

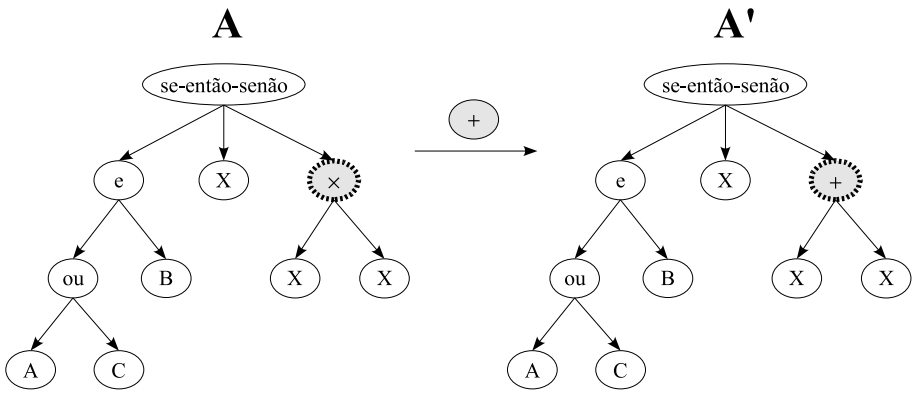

Figura 2.12: O processo de mutação do alelo.

A mutação encolhimento, como o nome diz, objetiva reduzir o tamanho do cromossomo (árvore), inspirada no fato de que soluções menores (menos complexas) tendem a ser mais compreensíveis, e portanto desejadas. Seu procedimento consiste em, após escolhido um elemento aleatoriamente, selecionar um filho qualquer e, então, substituí-lo na posição do elemento escolhido (vértice pai), excluindo-se todos os outros filhos, juntamente com suas ramificações. Isto "compacta" a sub-árvore em questão, e portanto reduz o cromossomo como um todo. A Figura 2.13 ilustra visualmente a aplicação da mutação

encolhimento.

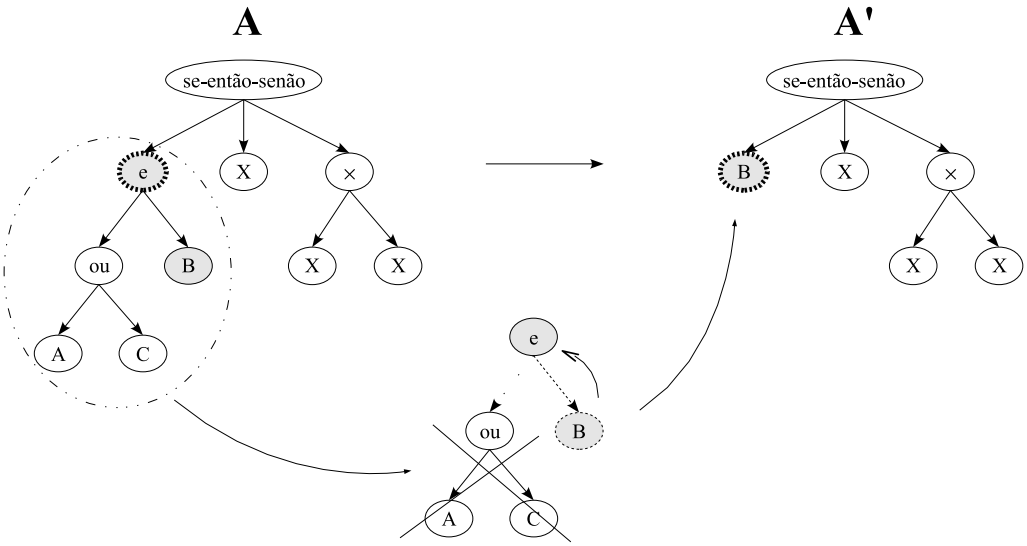

Figura 2.13: O processo de mutação encolhimento.

2.3.8.3 Permutação

A permutação é um operador genético que provoca uma alteração (permuta) na ordem dos argumentos de uma função qualquer. Logicamente, este operador só se aplica a funções com no mínimo dois argumentos. Um exemplo pode ser visto na Figura 2.14.

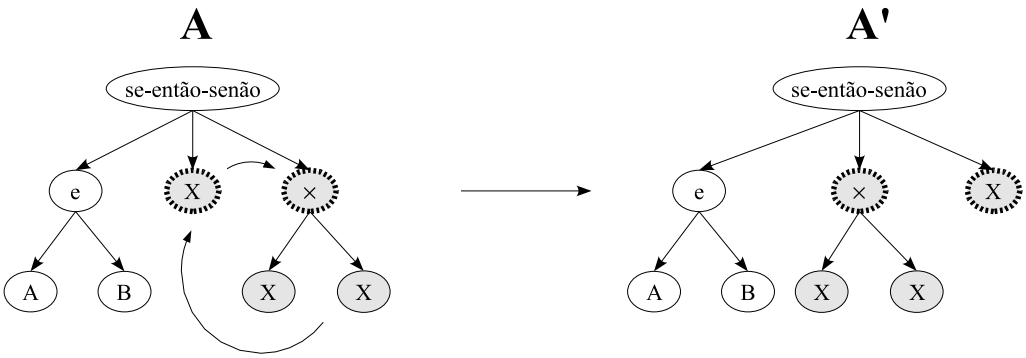

Figura 2.14: O operador de permutação.

2.3.8.4 Edição

A *edição* é, na verdade, uma simplificação de um cromossomo. Sua função é retirar redundâncias contidas em um indivíduo (árvore). Sua forma básica elimina os elementos

de uma árvore que não executam tarefa, por exemplo, na expressão $\sqrt{1}$, o operador raiz quadrada está redundante e poderia ser eliminado. A *edição* pode também ser aplicada em casos nos quais é possível a substituição de uma forma por outra mais simples. Por exemplo, seria interessante trocar $sen^2x + cos^2x$ por 1.

O problema do operador *edição* é que ele geralmente requer um custo computacional muito grande, sendo assim, raramente é empregado.

2.3.9 Áreas de Aplicação

O campo de atuação da PG é vasto e em expansão, qualquer tentativa de enumerá-lo seria um fracasso. Porém, como caráter didático, pode-se citar algumas áreas [15]

- **regressão simbólica –** consiste em determinar funções que melhor se ajustam aos pontos fornecidos como comparação. A Seção 2.4 fornece um exemplo comentado acerca deste campo de atuação.
- **classificação –** o objetivo é possibilitar a "discretização" de domínios "contínuos", agrupando-os em classes distitnas, sob um certo critério. Pode ser aplicado ao reconhecimento de padrões, de formas, entre outros. Como exemplo, suponha que se queira realizar reconhecimento de caracteres em uma imagem gráfica. O algoritmo solução seria encarregado de classificar os *pixels* em texto alfa-numérico. Uma importante área da classificação é a classificação de dados, onde se visa evoluir um programa classificador capaz de ler um conjunto de amostras (contendo inúmeros atributos) de um banco de dados e então classificá-las corretamente de acordo com o conteúdo de seus atributos [16].
- **ordenação –** sua função é desenvolver algoritmos que sejam capazes de ordenar, por alguma chave, conjunto de registros. O treinamento pode ser realizado apropriando-se de um conjunto de registros ordenados, e verificando-se quão bem as soluções candidatas ordenam estes registros. Resultados mais eficazes e elegantes surgem quando agrega-se à PG técnicas como a co-evolução, como demonstrado por Hillis [27].

[15] Koza, em *Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems* [32], fez um trabalho bastante abrangente de enumeração dos domínios de aplicação da programação genética.

[16] No Capítulo 5 são realizados experimentos neste contexto, porém integrando à PG outras técnicas, a serem vistas nos Capítulos 3 e 4.

- **robótica –** objetiva desenvolver robôs que sejam capazes de executar uma determinada tarefa, evitando-se, dentre outras coisas, a colisão contra obstáculos. Robôs podem, por exemplo, guiar-se pela coordenada da região do ambiente, tomando decisões baseadas no resultado de uma função evoluída para um propósito particular. Ou ainda, estender as variáveis de entrada e incluir, por exemplo, dados sobre a distância até o obstáculo, etc.

- **compactação de imagens –** teoricamente, as imagens, constituídas de *pixels* e cores, são passíveis de descrição simbólica, por meio de funções. De uma maneira básica, em uma imagem bi-dimensional, cada ponto (*pixel*) descrito pela coordenada (x, y) pode ser mapeado por uma função evoluída via PG, que retorna uma cor (código) de acordo com a coordenada deste ponto. Com isto, varrendo-se a região da imagem, pode-se reconstruí-la, apenas com a utilização da função evoluída.

2.4 Regressão Simbólica

No intuito didático, e visando introduzir os conceitos da PG em uma aplicação prática, esta seção explica o que é regredir simbolicamente uma expressão, como os métodos tradicionais implementam a "regressão" e qual a vantagem da utilização da programação genética sobre estes métodos. Ao final, são citados exemplos de algumas classes de aplicação onde a regressão simbólica seria de fundamental importância.

2.4.1 Regressão Simbólica de uma Expressão

Regredir simbolicamente uma expressão é o ato de inferir simbolicamente uma expressão algébrica, dispondo apenas de um conjunto finito de amostras[17] numéricas. Basicamente, procura-se uma curva que melhor se ajuste aos dados fornecidos. O objetivo é encontrar uma função que minimize uma medida dos erros (ou discrepâncias) entre os valores previstos pela função e aqueles de fato observados. Dessa forma, matematicamente tem-se:

Seja F o conjunto de todas as funções $f : \Re^n \mapsto \Re$ admissíveis para o problema, $\{p_1, p_2, \ldots, p_k\}$ o conjunto de pontos conhecido como amostragem e $Y \in \Re^k$ o vetor cuja componente Y_i é o valor observado em $p_i \in \Re^n$, onde k é o número de pontos fornecidos.

[17] Amostras são também conhecidas como conjunto de treinamento ou dados experimentais.

Então, regredir simbolicamente é buscar por uma função f^* tal que:

$$f^* = \arg\min_{f \in F} d(f, Y)$$

onde $d(f,Y) \geq 0$ mede a discrepância entre os valores previstos por f e aqueles observados.

A função f^* é dita a mais adaptada (ou melhor ajustada) aos pontos fornecidos.

Uma possibilidade é fazer

$$f^* = \arg\min_{f \in F} \left[\sum_{i=1}^{k} |f(p_i) - Y_i|^q \right]^{\frac{1}{q}} \qquad q \in \Re \mid q \geq 1$$

O caso $q = 2$ corresponde ao popular ajuste por mínimos quadrados. Outra possibilidade é

$$d(f,Y) = \max_{1 \leq i \leq k} |f(p_i) - Y_i|$$

e ainda outra é

$$d(f,Y) = med\{|f(p_1) - Y_1|, |f(p_2) - Y_2|, \ldots, |f(p_k) - Y_k|\}$$

onde $med\{\}$ denota a mediana do conjunto.

Exemplo de regressão simbólica

Como exemplo, suponha que deseja-se descobrir qual a melhor função que se ajusta aos pontos (x_i, y_i) mostrados na Tabela 2.1.

x_i	y_i
-1.0	2.0
-0.5	0.5
0.0	0.0
0.5	0.5
1.0	2.0

Tabela 2.1: Pontos discretos da função exemplo

O ajuste exato para as amostras da Tabela 2.1 corresponde à função $f(x) = 2x^2$.

Graficamente os pontos da Tabela 2.1 de amostras podem ser visualizados na Figura 2.15. A Figura 2.16 ilustra o ajuste dos pontos da Tabela 2.1 através da função $f(x) = 2x^2$.

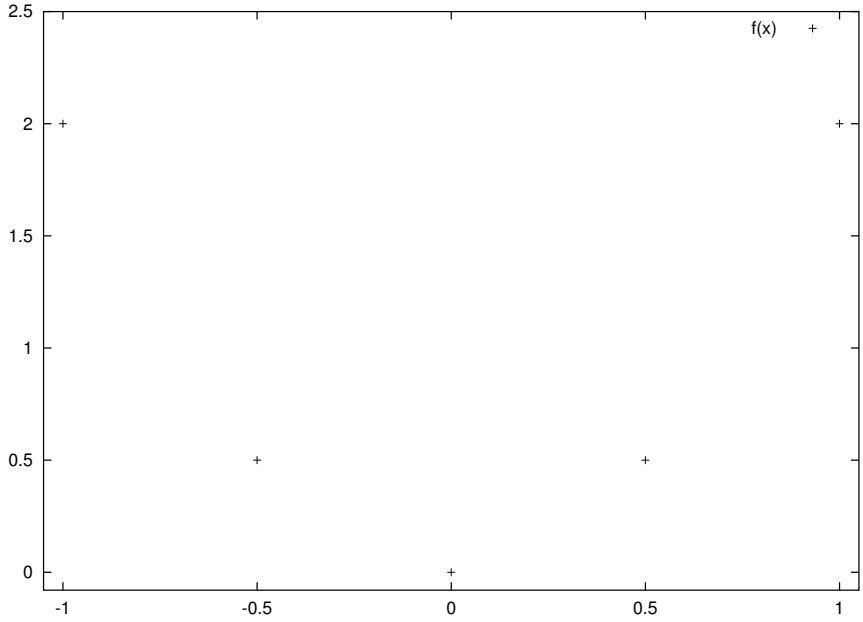

Figura 2.15: Gráfico baseado nos pontos discretos da Tabela 2.1.

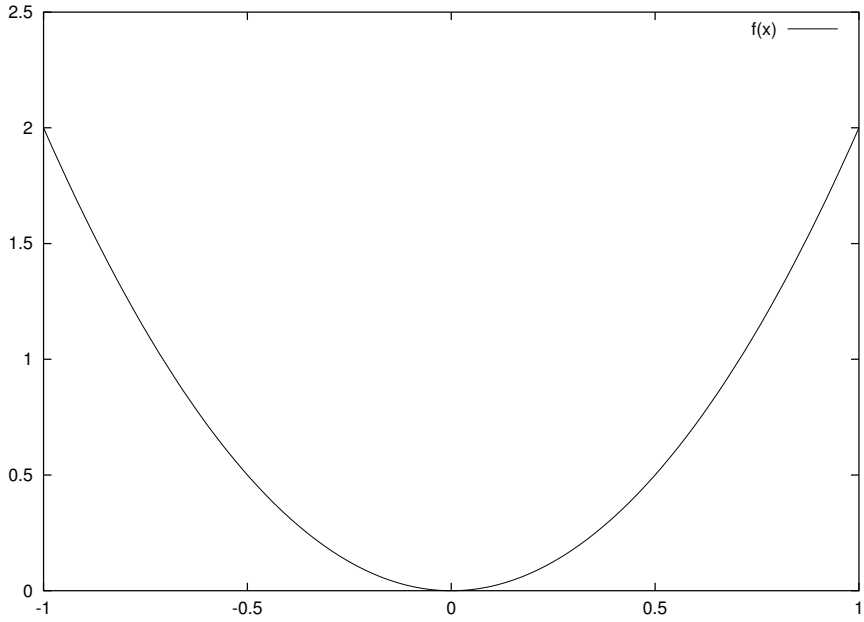

Figura 2.16: Gráfico contínuo baseado na forma exata da função $f(x) = 2x^2$.

2.4.1.1 Métodos Clássicos × Programação Genética

Muitos matemáticos e cientistas da computação têm desenvolvido técnicas de ajuste de funções[18]. Porém, estas requerem um modelo de expressão definido, sendo fortemente dependentes da classe das funções, e por isso são específicos e pouco flexíveis. Isto é, considerando o exemplo da Seção 2.4.1, seria fornecido ao método a forma pré-definida $f(x) = cx^2$ e este ajustaria apenas o coeficiente real c. Ou ainda, em um ajuste mais complexo, seria provido o modelo pré-estabelecido $f(x) = c_1 x^{c_2}$ e então o método, além do ajuste do coeficiente c_1, também seria responsável por ajustar o expoente c_2.

Não é de interesse apenas a regressão linear, quadrática ou mesmo polinomial. Busca-se regredir, também, funções não lineares e sem qualquer forma ou complexidade pré-definida, em outras palavras, deseja-se identificar simultaneamente a estrutura e os coeficientes da função. Isto significa que se estaria trabalhando sobre um espaço vastíssimo de busca. Portanto, são necessários métodos de busca que viabilizem o manejamento de tal espaço. A programação genética, pela sua robustez e flexibilidade, é capaz de lidar com espaços de busca vastos e complexos.

A programação genética aplicada à regressão simbólica não requer uma estrutura de expressão definida, ajustando-se perfeitamente ao problema, pois ela trabalha sem nenhum conhecimento sobre a função almejada, o que torna essa meta-heurística muito robusta. Voltando ao exemplo da Seção 2.4.1, utilizando-se a programação genética, não é preciso informar nenhum dado sobre o modelo da função procurada e muito menos sobre seus coeficientes, estas "deduções" ficam a cargo da evolução.

2.5 Conclusão

Este capítulo introduziu o conceito geral da evolução por seleção natural, que se apresenta como a inspiração e o princípio do funcionamento das técnicas computacionais de evolução denominadas algoritmos genéticos e programação genética.

Descreveu-se sucintamente os componentes do AG, como a reprodução, representação, função de avaliação, esquemas de seleção e os operadores genéticos. Prosseguiu-se com a explanação sobre a PG, evidenciando-se a diferença crucial em relação aos AGs,

[18] Mais especificamente eles buscam ajustar os coeficientes, como na regressão linear, quadrática e polinomial.

isto é, a codificação e evolução de programas de computador, geralmente representados pela estrutura árvore. Os conceitos pertinentes à PG foram elucidados e, por fim, foi exemplificada uma aplicação da PG no domínio da regressão simbólica, mostrando-se ainda como a PG pode ser vantajosa em relação aos métodos clássicos.

Em suma, foi visto que a PG é robusta, qualidade herdada da teoria da seleção natural; evolui soluções simbólicas/interpretativas; é flexível e maleável, no sentido de receptividade e integração com outras técnicas.

O próximo capítulo apresenta um elegante mecanismo formal de representação de soluções, em substituição ao método tradicional —e limitado— da PG.

Capítulo 3

Programação Genética Gramatical

Este capítulo aborda os aspectos relacionados com a representação e codificação de indivíduos, por meio gramatical, na programação genética.

Inicia-se com a definição genérica de uma gramática, apresenta-se os aspectos formais, notações, a hierarquia e os tipos de gramáticas. Uma maior atenção é direcionada às gramáticas livre de contexto, não poupando exemplos didáticos como uma maneira conveniente de elucidar o seu conceito.

São revistas diferentes propostas de codificação alternativas e, finalmente, é feita a introdução e detalhamento da integração da gramática livre de contexto no âmbito da programação genética, adotada no presente trabalho.

3.1 Gramática

Uma linguagem é basicamente uma seqüência de símbolos dispostos segundo regras estabelecidas pela sintaxe. Por sua vez, a sintaxe de uma linguagem é expressa na forma de uma gramática.

A gramática essencialmente consiste do tratado dos fatos da linguagem e das leis que a regulam e define as regras e princípios que regem o funcionamento de uma língua [1].

[1] Uma introdução às gramáticas e linguagens formais pode ser encontrada em [9] e [2].

3.1.1 Definição Formal

Formalmente, as gramáticas atuam como dispositivos de geração de sentenças das linguagens, sendo representadas por uma quádrupla $G = (N, \Sigma, S, P)$ onde

- N é um alfabeto (conjunto finito de símbolos) de **não-terminais** [2];
- Σ é um alfabeto de **terminais** [3], sendo que $N \cap \Sigma = \emptyset$ e $N \cup \Sigma = V$, onde V é o conjunto de símbolos da linguagem;
- S é o símbolo **inicial** ou de partida, $S \in N$;
- P é um conjunto de regras ou **produções**, da forma $\alpha \rightarrow \beta$, onde $\alpha \in V^+$ e $\beta \in V^*$. V^* denota o conjunto de todas as seqüências compostas por elementos de V, incluindo o conjunto vazio. Analogamente, porém excluindo o conjunto vazio, V^+ contém as seqüências formadas por elementos de V. Produções do tipo $\{\alpha \rightarrow \beta, \alpha \rightarrow \gamma\}$ podem ser elegantemente resumidas, através do símbolo de disjunção $|$, para $\{\alpha \rightarrow \beta|\gamma\}$.

3.1.1.1 Notação Gramatical

No tocante à notação gramatical, convencionou-se a adoção de escrita dos símbolos em caracteres minúsculos. Para a distinção dos símbolos não-terminais, em relação aos terminais, optou-se pela inclusão do sinal menor e maior, no início e fim da descrição, respectivamente. Isto significa que símbolos não-terminais são escritos na forma **<símbolo>**. Por sua vez, são denotadas por letras gregas as formas que potencialmente são tanto símbolos terminais como não-terminais.

Como exemplo, considere uma linguagem L representando sentenças do tipo $ab, aabb, \ldots, a^n b^n$. Formalmente tem-se:

$$L = \{a^n b^n | n \geq 1\}.$$

[2] Símbolos não-terminais são aqueles que são substituídos por outros símbolos (sejam terminais ou não-terminais) no momento da geração da sentença da linguagem.

[3] O significado da palavra terminal adotado no âmbito das gramáticas difere do conceito empregado na programação genética, como visto no Capítulo 2. Na PG os terminais significam qualquer entidade que não requer argumentos, isto é, um nó terminal (folha) da árvore. Exemplos são as constantes, variáveis ou funções que não requerem argumentos. No mundo gramatical, os símbolos terminais representam o último nível de substituição, sendo, pois, fixados quando gerados. São os símbolos que comporão a sentença gerada da linguagem. Em outras palavras, os terminais são os símbolos que não podem ser substituídos.

L pode ser expressa pela gramática $G_L = (\{\text{<s>}\}, \{a, b\}, S, P)$, onde

$$S = \text{<s>}$$

e

$$P = \{\text{<s>} \rightarrow ab | a\text{<s>}b\}$$

Uma produção pode ter um símbolo não-terminal cujo lado direito é uma cadeia vazia. O objetivo é expressar uma substituição nula (representada pelo símbolo especial épsilon – ε), "retirando" o não-terminal da palavra sendo gerada. Por exemplo, em uma gramática que define parênteses balanceados (número de parênteses abertos é igual ao número de fechados)

$$G = (\{\text{<p>}\}, \{(,), \varepsilon\}, \text{<p>}, \{\text{<p>} \rightarrow (\text{<p>}) | \text{<p>}\ \text{<p>} | \varepsilon\})$$

requer o artifício épsilon para que a sentença seja finita. As sentenças geradas por esta gramática, incluindo a sentença nula, são do tipo $()$, $(())$, $()()$, $(()())$, ...

3.1.1.2 Passos de Derivação

Um passo de derivação é o processo de obtenção de um conjunto de símbolos — conhecidos como palavra ou cadeia de caracteres [4]— através da aplicação de uma produção sobre um símbolo não-terminal da cadeia de caracteres a ser derivada.

A derivação inicia-se pela aplicação de uma produção compatível [5] sobre o símbolo inicial S. Se esta palavra gerada contiver um ou mais símbolos não-terminais, o processo de derivação continua e segue com a aplicação das produções, até encontrar um estado onde não é possível qualquer derivação, isto é, a cadeia de caracteres resultante não contém símbolos não-terminais, culminando na forma final, a sentença.

Uma situação de indeterminação pode ocorrer quando há mais de um símbolo não-terminal em uma determinada palavra a ser derivada. Dependendo da ordem de escolha, uma derivação pode suceder-se diferentemente, produzindo sentenças distintas. Como forma de padronizar as ordens das substituições, as derivações podem ser realizadas ou

[4] Uma palavra ou cadeia de caracteres sobre um alfabeto é uma seqüência finita de símbolos (do alfabeto) justapostos.

[5] Diz-se que uma produção é compatível quando o símbolo que encabeça a produção (lado esquerdo) é o mesmo do símbolo a ser derivado.

pela esquerda ou pela direita. Isto significa que caso a derivação seja feita à esquerda, o não-terminal que terá prioridade na substituição será sempre aquele mais à esquerda. Raciocínio análogo para derivações à direita. Os exemplos didáticos presentes neste capítulo usam derivação à esquerda, salvo menção contrária.

A derivação é representada pelo símbolo $\Rightarrow$, sendo que:

- $\alpha \underset{G}{\overset{p}{\Rightarrow}} \beta$ simboliza a derivação de α em β pela produção p, usando-se a gramática G. A gramática pode ser omitida caso esteja implícita, podendo reduzir a representação para $\alpha \overset{p}{\Rightarrow} \beta$.
- $\alpha \overset{*}{\Rightarrow} \beta$ representa a transformação de uma palavra por zero ou mais passos de derivação. Ou ainda, explicitando-se a gramática, $\alpha \underset{G}{\overset{*}{\Rightarrow}} \beta$.
- $\alpha \overset{+}{\Rightarrow} \beta$ analogamente, indica a derivação por um ou mais passos.

Dando continuidade ao exemplo anterior da linguagem L, pode-se descrever a obtenção de uma certa sentença, por exemplo *aaabbb*, pelos seguintes passos de derivação:

1. uma produção compatível p é aplicada sobre o símbolo inicial S,

   $$\text{<s>} \overset{\text{<s>}\rightarrow a\text{<s>}b}{\Rightarrow} a\text{<s>}b$$

2. a mesma produção p anterior é aplicada,

   $$a\text{<s>}b \overset{\text{<s>}\rightarrow a\text{<s>}b}{\Rightarrow} aa\text{<s>}bb$$

3. finalmente, obtendo uma cadeia de caracteres sem símbolos não-terminais (sentença), é aplicada a produção <s> $\rightarrow ab$,

   $$aa\text{<s>}bb \overset{\text{<s>}\rightarrow ab}{\Rightarrow} aaabbb$$

É possível dizer também que

$$\text{<s>} \overset{+}{\Rightarrow} aaabbb$$

Uma série de derivações também pode ser exposta graficamente, na forma de árvore sintática, também conhecida como árvore de derivação [6], como mostrado na Figura 3.1.

[6] A árvore de derivação (sintática) tem como propriedades: 1) Os nós internos são símbolos não-terminais e, complementando, os nós folhas são símbolos terminais, 2) Um nó e seus filhos imediatos devem respectivamente formar a cabeça (lado esquerdo) e o corpo (lado direito) da produção.

Esta visualização tem um caráter especial, pois as operações genéticas são vistas mais naturalmente pela manipulação sobre árvores.

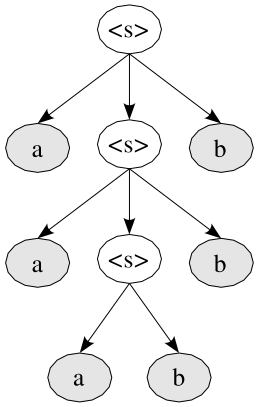

Figura 3.1: Árvore sintática dos passos de derivação da sentença *aaabbb*.

Em uma mesma gramática, uma palavra pode ter árvores ou séries de derivações distintas. Estas gramáticas são ditas ambíguas. Por exemplo, a gramática

$$G = (\{\text{<s>}\}, \{a, b, ab\}, \text{<s>}, \{\text{<a>} \rightarrow \text{<a>}a\text{<a>}|\text{<a>}b\text{<a>}|ab\})$$

pode gerar a cadeia

abaabbab

que, por sua vez, pode ser gerada por duas ou mais árvores de derivação, como mostrado na Figura 3.2. Por conseguinte, *G* é ambígua.

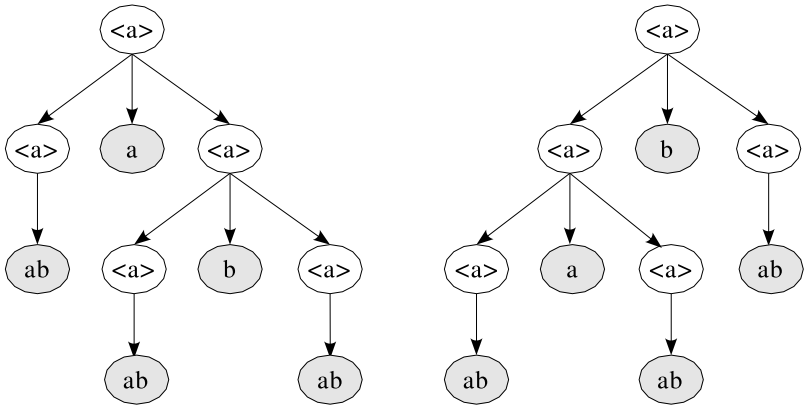

Figura 3.2: Árvores sintáticas da sentença *abaabbab*.

3.1.2 Hierarquia de Chomsky

O lingüista Noam Chomsky [14], com o propósito de construir potenciais modelos para linguagens naturais, agrupou as gramáticas em quatro classes, criando uma relação hierárquica entre elas. Esta hierarquia é conhecida como *Hierarquia de Chomsky*, ilustrada na Figura 3.3.

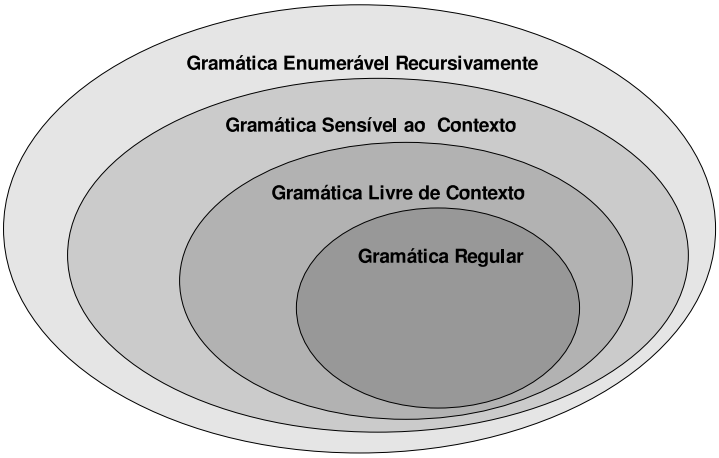

Figura 3.3: Hierarquia de Chomsky.

Caminhando pelos níveis da Hierarquia, cada classe gramatical torna-se mais expressiva e engloba a classe inferior. Da mais específica para a mais genérica, tem-se a **gramática regular** (tipo 3), **gramática livre de contexto** (tipo 2), **gramática sensível ao contexto** (tipo 1) e a **gramática enumerável recursivamente** (tipo 0).

Cada hierarquia expressa um nível de restrição em relação ao tipo de produção aceitável, sendo a gramática tipo 0 totalmente relaxada e a tipo 3 a mais restrita. Lembrando-se da definições $\alpha \in V^+$ ($|\alpha| \geq 1$), $\beta, \gamma, \delta \in V^*$ ($|\beta|, |\gamma|, |\delta| \geq 0$), <*a*>,<*b*>,<*c*>,... representando símbolos não-terminais, e $a, b, c, \ldots$ os símbolos terminais, tem-se na Tabela 3.1 a relação das gramáticas e suas produções.

Cada uma destas gramáticas atende a um propósito específico. Problemas ou aplicações relacionados ou que interagem com gramáticas são melhor atendidos quando trabalha-se com uma gramática que seja suficiente para expressar a complexidade requerida e, ao mesmo tempo, seja a menos genérica.

O envolvimento do presente trabalho com as estruturas gramaticais é de uma complexidade tal, que uma gramática regular seria insuficiente para representar as necessidades,

Gramática	Produções	
Tipo 0	$\alpha \rightarrow \beta$	Gramática irrestrita, produções podem ser de qualquer forma.
Tipo 1	β<*a*>$\gamma \rightarrow \beta\delta\gamma$	As produções são condicionadas a terem pelo menos um símbolo não-terminal no lado esquerdo. O adjetivo "sensível ao contexto" vem do fato de que a substituição do símbolo <*a*> em δ pode ser contextualizada por β e γ, ficando então restrita a este contexto.
Tipo 2	<*a*> $\rightarrow \beta$	A cabeça da produção é obrigatoriamente um e apenas um símbolo não-terminal. Diferentemente da gramática Tipo 1, na gramática livre de contexto não há um contexto para "moldar" as substituições, justificando portanto o adjetivo "livre de contexto".
Tipo 3	<*a*> $\rightarrow a$<*b*>$\vert b$ ou <*a*> $\rightarrow$ <*b*>$a \vert b$	Além da restrição de apenas um símbolo não-terminal no lado esquerdo da produção, a gramática regular ainda adiciona a condição da existência, no lado direito, de ou um terminal ou um não-terminal e terminal.

Tabela 3.1: Forma das produções das gramáticas na hierarquia de Chomsky.

enquanto uma gramática sensível ao contexto seria demasiadamente genérica. No entanto, a gramática livre de contexto mostra-se perfeitamente adequada para suprir toda a demanda necessária.

Por conseguinte, o desenvolvimento deste capítulo é focado na gramática livre de contexto, primeiramente através de um maior detalhamento desta e, adiante, abordando-se os aspectos de sua integração à programação genética.

3.1.3 Gramática Livre de Contexto — GLC

A GLC é uma gramática que exibe grande poder de expressão juntamente com a simplicidade, tanto conceitual como de implementação. Estas características fazem com que a gramática livre de contexto seja, por exemplo, a preferida para a definição das maioria das linguagens modernas de programação [45].

3.1.3.1 Representação

Recordando-se a classe e restrições das produções da GLC, tem-se

$$<a> \rightarrow \beta$$

onde <*a*> é um e apenas um não-terminal e β pode ser uma cadeia de símbolos terminais e/ou não-terminais.

Uma representação alternativa, conhecida como Forma Normal de Chomsky, é equivalente [7]e coloca as produções nas formas

$$<a> \rightarrow <b><c>$$

$$<a> \rightarrow a$$

Implicações práticas e teóricas surgem pela forma especialmente simples das regras de produções na Forma Normal de Chomsky. Por exemplo, dada uma GLC, pode-se construir um algoritmo em tempo polinomial [35] para decidir se uma dada cadeia de caracteres pertence ou não à linguagem representada pela gramática.

3.1.3.2 Exemplos Ilustrativos

Suponha um exemplo gramatical que ilustre a construção de sentenças de tal forma que uma determinada palavra fique em algum lugar dentre uma seqüência de caracteres, maiúsculos ou minúsculos.

Esta gramática pode ser construída como

$$G_{pal} = (\{<s>,<l>\}, \{\text{palavra}, A, B, \ldots, Z, a, b, \ldots, z, \varepsilon\}, S, P)$$

sendo

$$S = <s>$$

e

$$P = \{$$

[7]Equivalente significa que as duas formas de gramática geram a mesma linguagem.

<s> $\rightarrow$ <l>palavra<l>,

<l> $\rightarrow$ <l><l>|ε,

<l> $\rightarrow A|B|\ldots|Z|a|b|\ldots|z$

}

A gramática G_{pal} produz sentenças do tipo "Esta**palavra**aqui", cujos passos de derivação são

<s> $\overset{\text{<s>}\rightarrow\text{<l>palavra<l>}}{\Rightarrow}$ <l>palavra<l> $\overset{\text{<l>}\rightarrow\text{<l><l>}}{\Rightarrow}$ <l><l>palavra<l>

$\overset{\text{<l>}\rightarrow\text{<l><l>}}{\Rightarrow}$ <l><l><l>palavra<l> $\overset{\text{<l>}\rightarrow\text{<l><l>}}{\Rightarrow}$ <l><l><l><l>palavra<l>

$\overset{\text{<l>}\rightarrow E}{\Rightarrow} E$<l><l><l>palavra<l> $\overset{\text{<l>}\rightarrow s}{\Rightarrow} Es$<l><l>palavra<l> $\overset{\text{<l>}\rightarrow t}{\Rightarrow}$

Est<l>palavra<l> $\overset{\text{<l>}\rightarrow a}{\Rightarrow} Esta$palavra<l> $\overset{\text{<l>}\rightarrow\text{<l><l>}}{\Rightarrow} Esta$palavra<l><l>

$\overset{\text{<l>}\rightarrow\text{<l><l>}}{\Rightarrow} Esta$palavra<l><l><l> $\overset{\text{<l>}\rightarrow\text{<l><l>}}{\Rightarrow} Esta$palavra<l><l><l><l>

$\overset{\text{<l>}\rightarrow a}{\Rightarrow} Esta$palavra$a$<l><l><l> $\overset{\text{<l>}\rightarrow q}{\Rightarrow} Esta$palavra$aq$<l><l> $\overset{\text{<l>}\rightarrow u}{\Rightarrow}$

$Esta$palavraaqu<l> $\overset{\text{<l>}\rightarrow i}{\Rightarrow} Esta$palavra$aqui$

A estrutura gráfica das seqüências de derivações está exposta pela árvore sintática da Figura 3.4.

Considere agora o seguinte exemplo envolvendo a definição de uma gramática G_{pre} para geração de expressões aritméticas na forma prefixa sobre os números reais:

$$G_{pre} = (\{\text{<exp>}\}, \{+, -, \times, \div, \Re\}, S, P)$$

onde

$$S = \text{<exp>}$$

e ainda

$$P = \{\text{<exp>} \rightarrow \Re | + \text{<exp><exp>} | - \text{<exp><exp>} | \times \text{<exp><exp>} | \div \text{<exp><exp>}\}$$

Que produz, por exemplo, sentenças tais como

$$+ \div 3.14\ 2 - \times 7\ 3.2 - 604\ 50$$

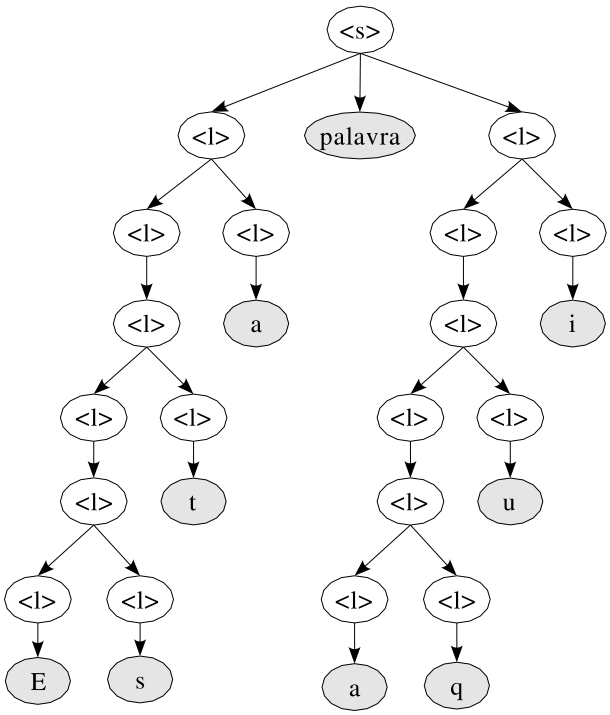

Figura 3.4: Árvore sintática dos passos de derivação sobre a gramática G_{pal}.

equivalente à forma infixa

$$(3.14 \div 2) + \big((7 \times 3.2) - (604 - 50)\big)$$

com a seguinte série de derivações

$$\begin{aligned}
&\texttt{<exp>} \overset{\texttt{<exp>}\to+\texttt{<exp><exp>}}{\Rightarrow} + \texttt{<exp><exp>} \overset{\texttt{<exp>}\to\div\texttt{<exp><exp>}}{\Rightarrow} \\
&+ \div \texttt{<exp><exp><exp>} \overset{\texttt{<exp>}\to 3.14}{\Rightarrow} + \div 3.14\ \texttt{<exp><exp>} \overset{\texttt{<exp>}\to 2}{\Rightarrow} \\
&+ \div 3.14\ 2\ \texttt{<exp>} \overset{\texttt{<exp>}\to-\texttt{<exp><exp>}}{\Rightarrow} + \div 3.14\ 2\ - \texttt{<exp><exp>} \overset{\texttt{<exp>}\to\times\texttt{<exp><exp>}}{\Rightarrow} \\
&+ \div 3.14\ 2\ - \times \texttt{<exp><exp><exp>} \overset{\texttt{<exp>}\to 7}{\Rightarrow} + \div 3.14\ 2\ - \times 7\ \texttt{<exp><exp>} \overset{\texttt{<exp>}\to 3.2}{\Rightarrow} \\
&+ \div 3.14\ 2\ - \times 7\ 3.2\ \texttt{<exp>} \overset{\texttt{<exp>}\to-\texttt{<exp><exp>}}{\Rightarrow} + \div 3.14\ 2\ - \times 7\ 3.2\ - \texttt{<exp><exp>} \\
&\overset{\texttt{<exp>}\to 604}{\Rightarrow} + \div 3.14\ 2\ - \times 7\ 3.2\ -\ 604\ \texttt{<exp>} \overset{\texttt{<exp>}\to 50}{\Rightarrow} + \div 3.14\ 2\ - \times 7\ 3.2\ -\ 604\ 50
\end{aligned}$$

Estes passos de derivação também podem ser vistos graficamente, na forma de uma árvore de derivação, Figura 3.5.

Por curiosidade, a gramática infixa das expressões aritméticas seria

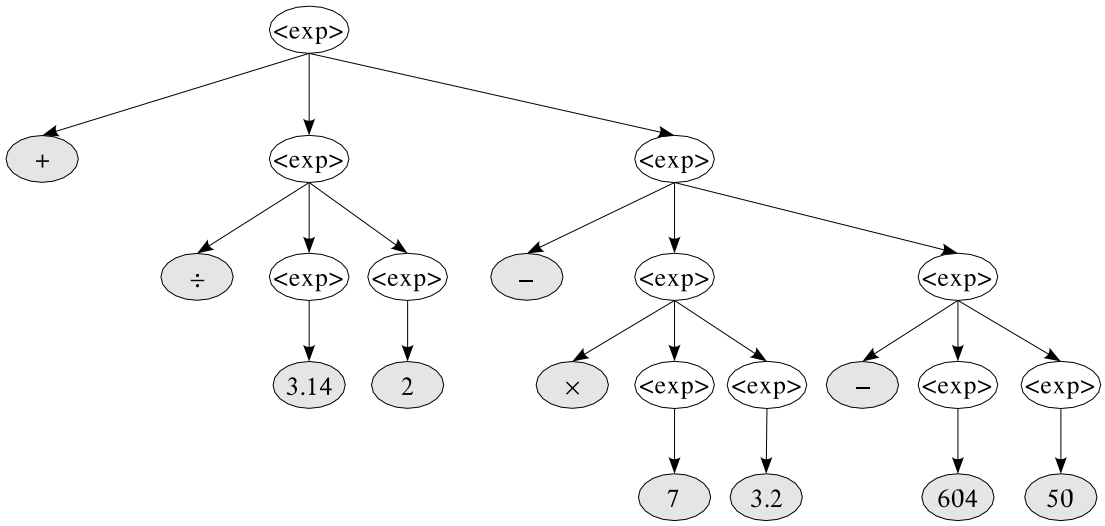

Figura 3.5: Árvore sintática dos passos de derivação da expressão prefixa.

$$G_{inf} = (\{\text{<exp>}\}, \{+, -, \times, \div, \Re\}, S, P)$$

onde

$$S = \text{<exp>}$$

e ainda

$$P = \{\text{<exp>} \rightarrow \Re | (\text{<exp>}) |$$

$$\text{<exp>} + \text{<exp>} | \text{<exp>} - \text{<exp>} | \text{<exp>} \times \text{<exp>} | \text{<exp>} \div \text{<exp>}\}$$

Na gramática G_{inf} nota-se, além das modificações dos operadores na posição infixa, a introdução da produção <exp> $\rightarrow$ (<exp>) que permite, através da utilização de parênteses, impor precedências outras na ordem das operações.

Uma variável, por exemplo x, conhecida no linguajar gramatical também como um terminal, pode ser trivialmente inserida nas definições das gramáticas. No caso da gramática G_{inf}, bastaria a adição da variável x no conjunto dos terminais Σ e, claro, sua definição como parte das produções, resultando em

$$G_{inf} = (\{\text{<exp>}\}, \{+, -, \times, \div, \Re, x\}, S, P')$$

e a nova produção

$$P' = P \cup \{\text{<exp>} \rightarrow x\}$$

que poderia gerar expressões como

$$(3.14 \div 2) + \big((x \times 3.2) - (604 - x)\big)$$

tornando plausível a aplicabilidade das GLCs em incontáveis problemas.

3.2 Linguagem e Representação na Programação Genética

Da forma canônica às propostas mais requintadas, soluções interessantes têm sido criadas para representar programas dentro da programação genética. Algumas são de caráter mais geral, enquanto outras são desenvolvidas especificamente para abordar um determinado problema.

A revisão bibliográfica posta nesta seção visa levantar variadas alternativas genéricas empregadas como forma de codificação, apontando suas vantagens e desvantagens. No entanto, não se pretende avaliar exaustivamente e muito menos exaurir toda e qualquer proposta que se encaixa neste perfil.

São mencionados e comentados os trabalhos importantes para então introduzir e discutir a proposta do presente capítulo: a programação genética gramatical.

3.2.1 Estruturas Sintáticas Restritas — ESR

Na tentativa de abordar as fraquezas da PG no tocante à condição de consistência (vide Capítulo 2), Koza [33] elaborou, em caráter *ad hoc*, uma restrição de sintaxe denominada Estruturas Sintáticas Restritas – ESR (*Constrained Syntactic Structures*). Basicamente, foram definidas regras (dependentes do problema) especificando quais funções e terminais eram permitidos como argumento de um determinado nó da árvore (função).

Por exemplo [33], no problema Séries de Fourier, Koza especificou as seguintes regras de sintaxe:

- A raiz da árvore deve ser uma função especial, &.
- As únicas funções admitidas como argumento de & são as funções trigonométricas *xsen*, *xcos* e a própria &.

- Os argumentos das funções trigonométricas são somente funções aritméticas ($+,-,\times,\div$) ou uma constante real aleatória.

Os operadores genéticos, como inicialização, cruzamento e mutação são realizados de maneira que não violem as restrições de sintaxe:

- **inicialização –** o nó raiz da árvore deve retornar o tipo requerido pelo problema e, sucessivamente, os nós filhos (argumentos) devem retornar o tipo de dado requerido pelo nó pai.
- **cruzamento –** a troca de sub-árvore[8] deve respeitar o rótulo atribuído ao nó escolhido como ponto de corte. Isto é, o cruzamento é restrito aos nós que possuem afinidade em relação ao tipo.
- **mutação –** análoga ao processo de inicialização, considerando-se a mutação canônica empregada na PG.

Embora esta abordagem contorne os pontos negativos da proposta inicial, ela precisa ser modificada para cada problema. Além disso, exige que a declaração seja feita de forma textual (linguagem humana).

3.2.2 Programação Genética com Tipos Fortes — PGTF

Montana [37] propôs uma generalização para o método ESR, desenvolvido por Koza. Sua ênfase foi direcionada ao fortalecimento do conceito "tipo", isto é, cada função especifica exatamente o tipo do argumento requerido e o tipo do valor retornado.

Na Programação Genética com Tipos Fortes (*Strongly Typed Genetic Programming*), ou simplesmente PGTF, cada variável ou constante tem um tipo associado, por exemplo, a constante π é do tipo real, enquanto que a variável x é inteira. De modo semelhante, às funções são atribuídos os tipos dos argumentos bem como o valor retornado.

No entanto, em relação ao ESR, a maior contribuição feita por Montana, em seu trabalho, foi a extensão da noção dos tipos, incluindo os conceitos de **funções genéricas** e **tipos genéricos de dados**.

[8] Troca de sub-árvore é operação de *crossover* comumente praticada na programação genética.

Funções genéricas são funções que aceitam argumentos de qualquer tipo. Por exemplo, basta apenas uma função *se-então-senão*, com os argumentos do tipo *lógico*, *t* e *t*, retornando *t*, onde *t* é um tipo de dado qualquer, podendo ser *inteiro*, *real*, entre outros. Sem a generalização, seria preciso definir a função *se-então-senão-inteiro* com argumentos *lógico*, *inteiro* e *inteiro*, retornando *inteiro*; e *se-então-senão* com argumentos *lógico*, *real* e *real*, retornando *real*.

Em termos de implementação, a diferença entre uma função comum para a genérica é que a última precisa ser instanciada no momento de geração de árvores (inicialização). Neste processo, a função genérica é "amarrada" pelo tipo de argumento exigido pela função pai —se for a raiz, o argumento é do tipo requerido pelo problema. Respeitando esta restrição de retorno, a escolha da função específica procede livremente.

A função genérica traz uniformidade e clareza para funções que realizam o mesmo tipo de operação em tipos de dados distintos.

Os tipos genéricos de dados, por sua vez, permitem que os tipos de dados possam ser definidos genericamente. O principal benefício da utilização destes tipos é que as funções evoluídas são funções genéricas. Por exemplo, considere que um sistema de programação genética necessite evoluir a função **n-ésimo**, que dada uma lista de um tipo qualquer *t* e um inteiro *n*, retorna o *n*-ésimo elemento desta lista. Se *t* não pudesse ser declarado como genérico, a evolução ocorreria apenas para um tipo definido, como *real* ou *inteiro*, de cada vez.

Basicamente, no que diz respeito aos operadores genéticos, os procedimentos são semelhantes ao descritos para o ESR, com ressalva para as funções genéricas, que necessitam ser instanciadas no momento da construção da árvore.

Alguns estudos indicam que o uso de regras de sintaxe que restringem o espaço de busca tem se mostrado promissor. Haynes *et al.* [26], por exemplo, demonstrou superior desempenho sobre a PG convencional, utilizando a PGTF no domínio de evolução de estratégias predador-presa.

No que concerne à forma estrutural de restrição, a PGTF não é capaz de representá-la globalmente, limitando-se a manipulá-la localmente. Isto é, uma função ou argumento é restrito apenas em relação a outra função ou argumento. Por conseguinte, esta característica fortemente orientada ao tipo inviabiliza a criação de relações estruturais mais

complexas. Por exemplo, existem situações onde apesar da compatibilidade quanto ao tipo, funções ou terminais não deveriam se intercambiar. Suponha a existência de variáveis e constantes de mesmo tipo. Pode ser desejável ou requerido pelo problema que variáveis nunca co-existam em operações binárias relacionais, exigindo argumento constante e variável ou vice-versa. Este tipo de restrição fica prejudicada [9], pois a sintaxe é construída exclusivamente sobre o tipo.

3.2.3 Evolução Gramatical — EG

Ryan, O'Neill *et al.* [41] propuseram um sistema chamado Evolução Gramatical (*Grammatical Evolution*), baseado na gramática livre de contexto, que teoricamente evolui programas em uma linguagem arbitrária, desde que esta possa ser descrita pela GLC.

A proposta foi baseada nos algoritmos genéticos com cromossomo de comprimento variável, codificando em seus genes números inteiros que representam a seqüência de escolhas das produções em uma determinada gramática (do tipo GLC). As produções que possuíam um mesmo símbolo não-terminal como cabeça eram agrupadas e rotuladas. A interpretação do genótipo [10] inicia-se pelo símbolo inicial, que restringe o domínio de produções que podem ser selecionadas. Em outras palavras, as produções que estão concorrendo obrigatoriamente casam com o símbolo inicial. Seguindo, o primeiro gene é lido e a produção rotulada com este número é escolhida. Em seguida, a próxima posição do cromossomo é lida, porém, a restrição de seleção agora é dada pelo primeiro símbolo não-terminal mais a esquerda da cadeia previamente gerada, em vez do símbolo inicial. O processo continua até que todos os símbolos não-terminais sejam substituídos.

Por exemplo, seja a gramática da Tabela 3.2, que define uma linguagem de expressões envolvendo seno, cosseno, soma e subtração sobre a variável x. As produções de mesma cabeça estão agrupadas e especificadas com números romanos. Cada produção foi devidamente numerada, de acordo com o grupo a qual pertence. Por exemplo, no grupo **I**, representado pelo não-terminal <*exp*>, existem três escolhas, com produções numeradas de 0 a 2.

[9] De fato existe a possibilidade de se "burlar" esta limitação. Poderiam ser criados "pseudos-tipos" como *var-t*, denotando uma variável do tipo t e *const-t*, representando uma constante do tipo t. Entretanto, este artifício quebra a estrutura lógica da sintaxe e requer a adaptação em cascata das funções a fim de acomodar tipos de dados diferentes mas logicamente equivalentes.

[10] Um dos aspectos da evolução gramatical em relação à programação genética convencional é que a EG preserva a analogia biológica da morfogênese, ou seja, há uma distinção clara entre genótipo e fenótipo.

N =	{<exp>, <op>, <preop>, <var>}				
Σ =	$\{x$, sin, cos, $+$, $-$, $(,)\}$				
S =	<exp>				
P =	**I**	<exp>	→	<exp> <op> <exp>	**(0)**
				| <preop> (<exp>)	**(1)**
				| <var>	**(2)**
	II	<op>	→	$+$	**(0)**
				$\mid -$	**(1)**
	III	<preop>	→	sin	**(0)**
				| cos	**(1)**
	IV	<var>	→	x	**(0)**

Tabela 3.2: Gramática das operações sin, cos, $+$ e $-$ sobre a variável x.

Considere o genótipo da Figura 3.6, de comprimento 10 e valores inteiros entre 0 a 255 (8 bits [11]). Como o valor que cada gene pode representar é maior do que o número de produções indexadas, usa-se a função módulo para obter valores dentre os limites.

204	143	56	223	15	7	76	191	233	1

Figura 3.6: Cromossomo com codificação numérica inteira em 8 bits.

A "derivação" começa com a leitura do primeiro gene, 204. Existem três produções apontadas pelo símbolo de partida (<exp>), portanto, a selecionada é o resultado de 204 mod 3, que resulta em 0, escolhendo a produção <exp> → <exp> <op> <exp>. O próximo passo é ler o próximo gene, 143, escolhendo-se assim a produção substituta do não-terminal <exp> (esquerda). O módulo 143 mod 3 é igual a 2, por conseguinte, a produção eleita é <var> → x, que aponta para a variável x. Nota-se que como a "cardinalidade" do grupo **IV** é um, só existe uma produção, não é necessário promover uma seleção (e nova interpretação de gene), portanto, a substituição é direta, produzindo até o momento a expressão x<op> <exp>. As próximas séries são $x+$ <exp>, $x+$ <preop> (<exp>), $x+\cos($<exp>$)$, $x+\cos($<preop> (<exp>)$)$, $x+\cos(\sin($<exp>$))$, e finaliza com a interpretação do oitavo gene do cromossomo, produzindo $x+\cos(\sin(x))$ como a expressão final.

A evolução gramatical possui uma grande vantagem, ela lida com cromossomos lineares, sendo assim a EG pode empregar uma vasta gama de operadores genéticos usados nos

[11] A quantidade de bits necessária para a codificação está intimamente relacionada com o número de produções da gramática.

algoritmos genéticos. Naturalmente, existem também implicações práticas, por exemplo a linearidade do cromossomo facilita o trabalho de implementação.

Um fato curioso ocorrido no exemplo ilustrativo foi que o processo chegou ao fim sem a leitura de alguns genes. Isto ocorre porque não é possível determinar o momento onde a sentença (expressão completa) já estaria totalmente formada. Esta informação só seria possível mediante a operação de morfogênese, ou seja, somente quando ocorrer a interpretação do genótipo.

Apesar de haver alta incidência de genes não usados na natureza (também conhecidos como íntrons), supostamente atuando na longevidade de certas combinações entre genes [48], bem como na manutenção da diversidade, o paralelo com os algoritmos genéticos pode ser não vantajoso [3, 48], causando crescimento indesejável e atrasando a evolução como um todo.

Como tentativa de minimizar os contratempos causados pelos íntrons, os autores da EG introduziram o operador de poda, para a retirada periódica dos genes em excesso. Infelizmente, dada a natureza representativa dentro da EG, os íntrons sempre aparecerão durante o processo evolutivo.

Outro problema, ainda mais grave, é a possibilidade de não haver genes suficientes para gerar a forma final da expressão. Isto é, o comprimento do genótipo pode ser insuficiente. Se no exemplo da Figura 3.6, o oitavo gene de valor 191 fosse mudado para qualquer valor cujo módulo resultasse em zero, por exemplo, 192 ($192 \bmod 3 = 0$), a produção selecionada seria <exp> $\rightarrow$ <exp> <op> <exp>, e não haveria genes suficientes para substituir todos os não-terminais da produção escolhida. Neste cenário, pelo menos três alternativas podem ser pensadas: 1) eliminar os indivíduos com genótipo corrompido, 2) inserir novos genes arbitrariamente e 3) rotacionar a leitura dos genes, isto é, redirecionar ao primeiro gene quando o processo alcançar a última posição do cromossomo. A primeira e a segunda opção retardariam o processo evolucionário, visto que ou indivíduos estariam sendo sacrificados ou novos materiais genéticos estariam sendo indiscriminadamente inseridos, atuando como uma espécie de mutação agressiva. Ainda, a segunda abordagem poderia provocar um crescimento incontrolado, dependendo de quão "azarada" fosse a seleção das produções. A terceira alternativa, escolhida pelos autores da EG, parece ser a mais sensata, pois absolveria os indivíduos e ao mesmo tempo evitaria a introdução massiva de novos genes. Porém, há um sério problema: a EG não garante

a geração de genótipos factíveis. Seja a Figura 3.7, onde o resultado da aplicação do módulo nos três genes é igual a um ($x \bmod 3 = 1$).

202	145	55

Figura 3.7: Genótipo infactível sob rotação dos genes.

A produção <exp> → <preop> (<exp>) seria selecionada recursivamente por infinitas vezes. A rotação neste genótipo causaria, portanto, um crescimento ilimitado, invalidando o indivíduo.

3.3 Programação Genética Gramatical — PGG

Na programação genética os indivíduos passam por vários estágios de transformação ou interpretação, compreendendo basicamente a criação, avaliação, recombinação e aplicação de operadores genéticos. Em todos estes estágios, independentemente do número de ocorrências, é crucial que seja mantida a factibilidade do indivíduo. Infelizmente esta exigência tende a tornar-se cada vez mais desafiante à medida que a quantidade de tipos de dados aumenta.

Na sua concepção original, a estrutura representativa da PG trata de forma simplória e até mesmo deficiente a questão da representação (ou codificação) de programas. A maior limitação é a exigência da propriedade de consistência (Capítulo 2), obrigando que toda combinação possível entre funções e/ou terminais seja viável, para que assim os estágios de transformação possam ser processados. Em problemas envolvendo um único tipo de dado, como a regressão simbólica, esta restrição pode ser satisfeita naturalmente. Porém, quando o problema exige tipos de dados distintos, como a construção de um programa classificador de dados mistos, esta limitação torna-se um estorvo ou até mesmo proíbe o uso da codificação canônica.

Outro ponto frágil envolvendo a representação inicial está relacionado ao fato de que não há meios de impor uma verdadeira estrutura na codificação. Uma estrutura não meramente baseada na restrição de compatibilidade entre tipos de dados, mas sim no significado da relação entre os componentes (funções, constantes, variáveis), sob a ótica lógica do problema. Deficiência esta que abrange também as propostas ESR e PGTF, discutidas anteriormente.

O uso da representação definida por gramática cobre concomitantemente estes dois pontos chaves. Não obstante, as regras bem estabelecidas da gramática podam construções improfícuas, reduzindo-se assim o espaço de busca e, conseqüentemente, aceleram o processo evolutivo. Finalmente, a programação genética, evoluindo programas em uma determinada linguagem, é naturalmente uma forte candidata a ser submetida ao formalismo de uma gramática, em especial a GLC.

As duas alternativas discutidas neste trabalho que lidam com a representação gramatical e, por conseguinte, são isentas dos problemas mencionados, são a evolução gramatical e a programação genética gramatical. Por sua vez, a EG, apesar de sua simplicidade, apresenta alguns graves pontos colaterais. Em contrapartida a PGG, apesar de ser simples conceitualmente mas nem tanto em termos de implementação, mostra ser uma excelente forma de codificação para problemas no campo da programação genética, em especial na tarefa de classificação de dados, a qual é tema deste trabalho.

Este capítulo, e em particular esta seção, aborda os aspectos envolvidos na integração do sistema gramatical com a programação genética.

3.3.1 Definição

A aplicação de uma gramática no campo da PG incide primeiramente sobre a estruturação dos programas em processo de evolução. Define basicamente a gama de componentes ofertados (por exemplo funções, variáveis e constantes) e as regras de combinações entre estes, assegurando-se a consistência.

Whigham [56, 57], desenvolveu uma forma de representação baseada em gramática livre de contexto, conhecida como Programação Genética Gramatical [12], ou apenas PGG. Na PGG, o domínio do espaço de busca é moldado pela gramática, notoriamente pelas regras de produção. Isto significa que o objetivo do processo de evolução é encontrar, dentre todos os programas contidos neste espaço de busca, aquele que melhor se adapta ao "ambiente" do problema, o programa mais proficiente segundo a função de aptidão definida para a tarefa em questão.

Não satisfeito, Whigham ainda propôs [57], como conseqüência da inserção gramatical na PG, um novo conjunto de ferramentas a dispor do processo evolutivo, dentre elas

[12] *Grammatically-based Genetic Programming*, Programação Genética baseada em Gramática ou simplesmente Programação Genética Gramatical.

destacam-se duas:

- possibilidade de evoluir a própria gramática, ou seja, modificar símbolos não-terminais, terminais e produções, no decorrer da evolução;

- mutações direcionadas contendo produções (passíveis de evolução) que especificam a forma da sub-árvore a ser substituída.

Por questões envolvendo delimitação de escopo, e de fato as necessidades práticas acerca da gramática, estas questões adicionais não são investigadas no trabalho corrente. Certamente bons temas para trabalhos futuros.

3.3.2 PGG: GLC adaptada à Programação Genética Tradicional

O Capítulo 2 descreve a forma tradicional da PG, como concebida inicialmente por Koza. A introdução da representação por gramática requer modificações, sintetizadas por:

1. definição da gramática (GLC) adequada ao problema;

2. geração de indivíduos conforme a gramática definida;

3. adaptação dos operadores genéticos, como a recombinação e mutação.

As estruturas comparativas entre a PGG e a PG tradicional, visualizadas na forma de fluxograma, são mostradas na Figura 3.8. Nota-se grande semelhança entre as estruturas, sendo que a PGG difere apenas em dois pontos, a necessidade de uma gramática e a exclusão do processo de inicialização encontrado na PG tradicional (vide Seção 2.3.6). O primeiro ponto é trivial e já foi comentado. O segundo ponto é explicado pelo fato de que o processo de criação e inicialização são realizados juntamente, sendo considerados, na PGG, como sinônimos. Esta questão é abordada adiante.

O ciclo de processamento usando-se a PGG pode ser descrito sucintamente pelo pseudo-algoritmo traçado pelo Algoritmo 3.1. As etapas do ciclo são abordadas nos seus pormenores nas seções seguintes.

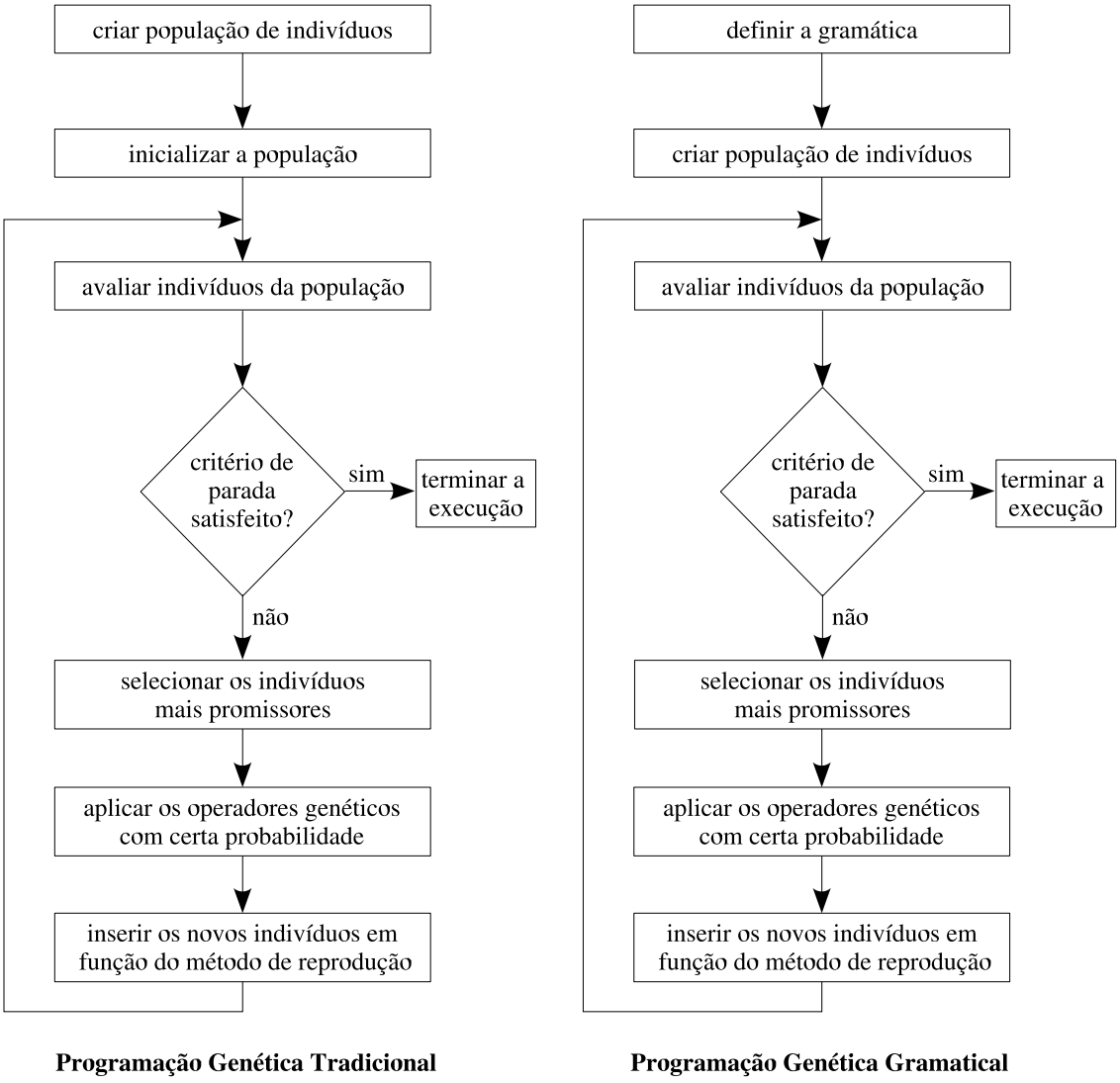

Figura 3.8: A estrutura do fluxo da PG e PGG.

3.3.2.1 Gramática Livre de Contexto para o Problema

A definição de uma gramática (livre de contexto) é a primeira tarefa a ser desenvolvido para o problema. A GLC estipula o conjunto de funções, constantes e variáveis, bem como as relações e restrições destes. Esta gramática é exatamente nos moldes [13] das GLC discutidas na Seção 3.1.3.

Como exemplo referência, considere uma gramática simplificada (G_{iris}) para o problema de classificação ternária para a planta íris [14], exibida na Tabela 3.3. Em resumo,

[13] Naturalmente, alguns detalhes podem diferir-se para que possam ser adaptados à implementação em uma linguagem computacional.

[14] Uma descrição mais minuciosa encontra-se na Seção 5.2 do capítulo 5.

Algoritmo 3.1 Pseudo-algoritmo do funcionamento da PGG.

Requer: gramática definida para o problema

```
crie os indivíduos aleatoriamente;
laço
  avalie os indivíduos;
  se critério de parada satisfeito então
    termine a execução;
  fim se
  selecione os indivíduos mais promissores;
  aplique os operadores genéticos;
  insira os novos indivíduos de acordo com o método de reprodução;
fim laço
```

trata-se da tarefa de descobrir uma função capaz de melhor classificar em classes distintas as três variedades (*setosa*, *versicolour* e *virginica*) de planta íris, sendo para isto fornecidas quatro variáveis numéricas contínuas.

N =	{<se-então-senão>, <lógico>, <numérico>, <relacional>, <atrib-numérico>, <classe>, <const-numérica> }		
Σ =	{se-então-senão, e, ou, não, igual, maior, menor, atr_1, atr_2, atr_3, atr_4, *setosa*, *versicolour*, *virginica*, $const_n$ }		
S =	<se-então-senão>		
P =	<se-então-senão>	→	<classe> | se-então-senão <lógico> <se-então-senão> <se-então-senão>;
	<lógico>	→	e <lógico> <lógico> | ou <lógico> <lógico> | não <lógico> | <relacional>;
	<relacional>	→	maior <numérico> <numérico> | menor <numérico> <numérico> | igual <numérico> <numérico>;
	<numérico>	→	<atrib-numérico> | <const-numérica>;
	<atrib-numérico>	→	atr_1 | atr_2 | atr_3 | atr_4;
	<const-numérica>	→	$const_1$ | $const_2$ | ... | $const_n$;
	<classe>	→	*setosa* | *versicolour* | *virginica*;

Tabela 3.3: Gramática G_{iris} simplificada.

A GLC G_{iris} fornece a habilidade de se criar indivíduos (árvores de classificação) que envolvam decisões condicionais (**se-então-senão**), comparações relacionais (**maior**, **menor** ou **igual**) entre os atributos numéricos da base de dados íris e/ou constantes numéricas, bem como operações lógicas (**e**, **ou** ou **negação**) sobre qualquer operação relacional.

A gramática é uma geradora de sentenças, e a linguagem de uma determinada gramática é o conjunto (possivelmente infinito) de todas as sentenças que podem ser geradas

a partir desta. A característica recursiva da G_{iris} define uma linguagem própria digna de construir programas de complexidade arbitrária, respeitando-se as relações definidas pela gramática.

3.3.2.2 Criação dos Indivíduos da População Inicial

Uma sentença é gerada por uma série de derivações, e um indivíduo na PGG é exatamente isso, uma árvore contendo uma determinada seqüência (completa) de derivações, enraizada pelo símbolo inicial definido na GLC.

No exemplo da gramática G_{iris}, todos os programas deverão obrigatoriamente iniciar por

<se-então-senão> → <classe>

ou

<se-então-senão> → se-então-senão <lógico> <se-então-senão> <se-então-senão>

já que o símbolo inicial é o não-terminal <se-então-senão>, encabeçando duas produções distintas.

Graficamente, como árvore sintática, as formas iniciais possíveis são visualizadas na Figura 3.9.

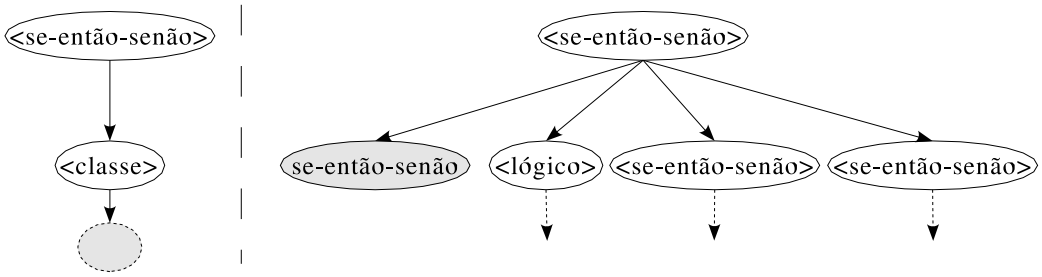

Figura 3.9: As possíveis raízes das árvores de derivação segundo a G_{iris}.

Na Figura 3.9, a primeira estrutura, culminando diretamente em uma classe, tem possibilidades remotas de ser uma solução útil, não local. O motivo é que o símbolo não-terminal <classe> deriva obrigatoriamente em um símbolo terminal, que representa a classe da planta íris, sendo *setosa*, *versicolour* ou *virginica*. Em outras palavras, a árvore classificadora fecha precocemente, classificando cegamente qualquer entrada como

uma e somente uma das três possíveis classes. Considerando-se eqüidade probabilística na seleção de ambas as produções, 50% dos indivíduos nasceriam virtualmente inválidos, enquanto a outra metade seria de fato a esperança no processo evolutivo. Felizmente, a pressão de seleção tenderia a eliminá-los o quanto antes, entretanto a presença de disparidade entre a ocorrência de amostras para cada uma das classes no conjunto de treinamento pode forçar uma convergência prematura —pode-se, entretanto, pré-processar a base de dados a fim de torná-la homogênea neste quesito. De qualquer forma, existem diversas maneiras de se contornar este tipo de construção, se necessário for. Destacam-se:

- a restrição gramatical por meio da adição de produções;
- a proibição no momento da criação dos indivíduos;
- a atribuição de uma baixa probabilidade para escolha desta produção;

A primeira proposta sobre a modificação gramatical poderia ser alcançada alterando-se as produções de G_{iris}, como por exemplo:

<se-então-senão>	→	se-então-senão <lógico> <a> <a>;
<a>	→	<se-então-senão> | <classe>;

Que seria o suficiente para restringir a raiz dos programas a aceitar unicamente a forma condicional. Entretanto, em raros casos onde a solução é trivialmente dada diretamente por uma classe, esta abordagem certamente prejudicará o processo. A alternativa para a proibição no momento da criação dos indivíduos também junta-se a esta observação. Caso decida-se por um tratamento especial acerca da formação precoce, talvez o melhor procedimento seja conseguido atribuindo-se uma pequena taxa para a seleção da produção trivial.

Criação não-determinística

Geralmente, na computação evolucionária, é adotado o método aleatório de criação dos indivíduos, assim também funciona com a programação genética gramatical. Na PGG o indivíduo é formado por uma seqüência não determinística de derivações, que resulta em uma sentença, isto é, até a resolução completa de todos os símbolos não-terminais. O processo de criação de um indivíduo inicia-se, como descrito, pela fixação do símbolo

de partida (não-terminal) como a raiz da árvore. Neste ponto sorteia-se uma produção dentre todas produções encabeçadas pelo símbolo inicial e adiciona-se esta como filha do símbolo de partida. Seguindo, para cada símbolo não-terminal mais à esquerda (caso haja mais de um), escolhe-se uma produção encabeçada por este símbolo e insere-a como filha deste símbolo não-terminal pai. O procedimento é assim repetido até que todos os símbolos não-terminais tenham sido resolvidos em símbolos terminais, formando então a sentença. De forma usual, a ordem de substituição dos símbolos não-terminais é realizada recursivamente em pré-ordem [15]. O Algoritmo 3.2, recursivo, mostra uma visão mais estruturada do processo de criação de um indivíduo.

Algoritmo 3.2 CriaÁrvore (<s>)
Entrada: símbolo inicial <s>

sorteie uma produção da forma <s> $\rightarrow \beta$, $\beta \in V^+$;
substitua <s> por <s> $\rightarrow \beta$, fazendo cada símbolo de β ser um filho de <s>;
para cada não-terminal <n> $\in \beta$ **faça**
  CriaÁrvore (<n>);
fim para

Nota-se que na formação do indivíduo, diferentemente da programação genética tradicional, não há distinção entre criação e inicialização, sendo, pois, conceitos tratados com sinônimos. Na PG, a fase de criação produz a forma estrutural do indivíduo, já a inicialização preenche a árvore com funções, constantes e variáveis, obviamente respeitando-se a estrutura previamente criada (número de argumentos). Na PGG a forma estrutural da árvore (indivíduo) é fixada juntamente com os símbolos, através da série de derivações.

Controle da explosão de crescimento da árvore

Um grave problema, de larga incidência para a maioria das gramáticas, inclusive a G_{iris}, é a seqüência de derivações intermináveis, criando ou indivíduos demasiadamente grandes ou inviáveis, fruto de uma série de derivações infinita. Este panorama ocorre porque o crescimento do número de símbolos não-terminais pode ser bem superior aos símbolos terminais, podendo causar discrepância em ordem de grandeza exponencial à medida que o processo de derivação avança. A solução é forçar, em um dado momento, que as únicas produções aceitáveis sejam aquelas que apresentem apenas terminais em

[15] O percurso em *pré-ordem* é também conhecido como percurso em *profundidade*. É realizado recursivamente, visitando-se primeiramente a *raiz*, o filho à *esquerda* (recursivamente) e então o filho à *direita* (também recursivamente).

seu corpo, ou seja, produções da forma <a> $\rightarrow a_1, a_2, \ldots, a_n$, sendo <a> $\in N$ e $a_n \in \Sigma^*$. O limite de profundidade da árvore declarado é que define o momento em que os símbolos não-terminais são imediatamente resolvidos em terminais. No entanto, a implementação desta abordagem requer um tratamento mais elaborado. Cada produção da gramática precisa ser rotulada de tal forma que indique o número mínimo de passos de derivação para que se crie somente símbolos terminais [16]. Isto permite ao sistema saber, em função do limite de profundidade máxima estabelecida, se uma determinada produção (contendo algum não-terminal) pode ser somada à árvore. Esta questão é melhor compreendida recorrendo-se a um exemplo hipotético em que o sistema precisa decidir, em uma árvore em trabalho de construção, qual tipo de produção será aceita, uma produção cujo corpo é constituído apenas de terminais ou uma outra que carrega também símbolos não-terminais. Isto dependerá de quantos níveis de profundidade a árvore ainda poderá expandir-se. No caso de apenas um nível, a escolha será pela produção que implica em símbolos terminais somente; caso maior do que um, dependerá se a produção poderá ser resolvida em símbolos terminais, com um total de passos de derivação menor ou igual ao espaço restante de expansão [17].

O Algoritmo 3.3 é do tipo *bottom-up* e executa a tarefa de rotulação das produções de uma GLC.

Inicia-se com a rotulação trivial de todas as produções cujo corpo é constituído somente de símbolos terminais, permitindo-se assim sustentar todo o procedimento de "baixo para cima". Em seguida, o algoritmo executa um laço até que todas as produções estejam rotuladas. Uma produção só está preparada para a rotulação se cada não-terminal do seu corpo encabeçar pelo menos uma produção previamente rotulada, caso contrário o algoritmo continua processando outras produções "bases" até que seja suficiente para rotulá-la. Em uma produção que contém mais de um símbolo não-terminal em seu corpo, o rótulo será sempre o maior valor dos rótulos de seus não-terminais acrescido de um. A explicação é simples, a maior profundidade (passos de derivação) para resolver todos os símbolos não-terminais é quem dita a rotulação mínima. A variável *maiorRótulo* auxilia neste propósito, e armazena sempre o maior valor dos rótulos dos símbolos não-terminais de uma determinada produção. Como um mesmo símbolo não-terminal pode encabe-

[16] Na notação derivativa compreende em descobrir o número mínimo de passos de derivação para se criar <a> $\overset{<a>\rightarrow\alpha}{\Rightarrow} \alpha \overset{*}{\Rightarrow} b$, onde $b \in \Sigma^*$.

[17] O espaço restante de expansão é o valor da subtração $m - n$, onde m é a profundidade máxima e n é o nível atual da árvore em formação.

Algoritmo 3.3 RotulaProduções (G)
Entrada: gramática livre de contexto contendo produções P

para cada produção $p_i \in G$ da forma <a> $\rightarrow a_1, a_2, \ldots, a_n$ **faça**
  rótulo$\{p_i\} \leftarrow 1$;
fim para
enquanto todas as produções não forem rotuladas **faça**
  para cada produção não rotulada $p_i \in G$ do tipo <a> $\rightarrow \alpha$, $\alpha \in V^*$ **faça**
    maiorRótulo $\leftarrow 1$;
    para cada não-terminal <b>$_j \in \alpha$ **faça**
      se a produção <b>$_j \rightarrow \beta$ já foi rotulada **então**
        maiorRótulo $\leftarrow \max(\arg\min(\text{rótulo}\{\text{<b>}_j \rightarrow \beta\}), \text{maiorRótulo})$;
      senão
        a produção p_i não pode ser rotulada ainda, voltar para o laço mais externo e continuar o processamento com a produção p_{i+1};
      fim se
    fim para
    rótulo$\{p_i\} \leftarrow$ maiorRótulo $+ 1$;
  fim para
fim enquanto

çar diversas produções, e estas não necessariamente possuem o mesmo valor de rótulo, a expressão $\arg\min(\text{rótulo}\{\text{<b>}_j \rightarrow \beta\})$ garante que sempre o rótulo de menor magnitude participará do cálculo. Este tratamento é relevante porque o objetivo é rotular com o somatório mínimo, evitando-se assim a eliminação (devido à restrição de profundidade) desnecessária de produções que poderiam ter um menor valor de rotulação.

A aplicação do Algoritmo 3.3 na gramática G_{iris} produz o esquema de rotulação mostrado na Tabela 3.4. Os valores dos rótulos estão cercados por parênteses.

Naturalmente, o algoritmo de construção tem que ser modificado para respeitar a restrição quanto à profundidade. O Algoritmo 3.4 é a versão com limite de profundidade adaptada do Algoritmo 3.2 de criação de indivíduo.

Algoritmo 3.4 CriaÁrvore (<s>, p_{max})
Entrada: símbolo inicial <s> e profundidade máxima p_{max}

sorteie uma produção da forma <s> $\rightarrow \beta$, tal que o **rótulo** $\leq p_{max}$, $\beta \in V^+$;
substitua <s> por <s> $\rightarrow \beta$, fazendo cada símbolo de β ser um filho de <s>;
para cada não-terminal <n> $\in \beta$ **faça**
  CriaÁrvore (<n>, $p_{max} - 1$);
fim para

P =	<se-então-senão>	→	**(2)** <classe> |
			(5) se-então-senão <lógico> <se-então-senão> <se-então-senão>;
	<lógico>	→	**(5)** e <lógico> <lógico> |
			(5) ou <lógico> <lógico> |
			(5) não <lógico> |
			(4) <relacional>;
	<relacional>	→	**(3)** maior <numérico> <numérico> |
			(3) menor <numérico> <numérico> |
			(3) igual <numérico> <numérico>;
	<numérico>	→	**(2)** <atrib-numérico> |
			(2) <const-numérica>;
	<atrib-numérico>	→	**(1)** atr_1 |
			(1) atr_2 |
			(1) atr_3 |
			(1) atr_4;
	<const-numérica>	→	**(1)** $const_1$ |
			(1) $const_2$ |
			(1) ... | $const_n$;
	<classe>	→	**(1)** *setosa* |
			(1) *versicolour* |
			(1) *virginica*;

Tabela 3.4: Produções da gramática G_{iris} rotuladas.

Métodos de criação da população

Os variados métodos de criação oriundos da PG (Capítulo 2), como criação completa, livre, entre outros, são utilizados na PGG sem necessidade de adaptações [18].

Garantia da diversidade populacional

A população inicial pode ter um forte impacto sobre o progresso da evolução. A diversidade é um componente essencial para a população inicial —e de fato para todo o processo evolutivo—, ajudando a cobrir de forma homogênea e abrangente o espaço de busca formado por programas. Assegurar que todos os indivíduos criados sejam distintos dos demais é um passo promissor nesta direção. A distinção pode referir-se a estrutura, funcionalidade, ou ambas, e pode ser introduzida como parte do processo de criação da

[18] Uma exceção pode ocorrer na utilização do método completo, que exige que todo comprimento entre o nó raiz e qualquer nó folha seja igual à profundidade máxima estipulada para a árvore (vide Seção 2.3.4). A possibilidade de cumprir fielmente esta exigência está condicionada à flexibilidade estrutural que a gramática impõe. De qualquer forma, pode-se instruir a criação a aproximar-se ao máximo do método completo.

população.

3.3.2.3 A Avaliação dos Indivíduos

A avaliação de um indivíduo na PGG, assim como na PG, se resume a medir o desempenho de um programa contra um certo conjunto de dados (treinamento). O número de amostras deste conjunto de dados depende da necessidade do problema. Um dado da amostra pode ser unidimensional, como um número real do problema de regressão simbólica de uma variável, bem como n-dimensional, como o caso da classificação de uma base de dados que envolvem vários atributos, por exemplo, a classificação das classes da planta íris.

A avaliação, assim, é a ponderação do resultado de várias execuções do programa para cada amostra de dados. Um objetivo usual é a minimização da soma das discrepâncias encontradas a cada amostra apresentada, entretanto, a escolha desta medida deve ser em função do caráter do problema.

O cerne de qualquer avaliação é a execução do programa. E para isto o sistema precisa interpretar de modo apropriado o programa, a árvore gramatical. Em termos genéricos, uma árvore de derivação possui três componentes:

- estrutura hierárquica - provê o fluxo de execução;
- símbolos não-terminais - mantêm a integridade estrutural;
- símbolos terminais - fornecem a funcionalidade.

Dentre os três componentes, o estágio de execução/avaliação requer apenas dois deles, a estrutura hierárquica e os símbolos terminais.

Até o momento, um indivíduo é uma árvore contendo símbolos não-terminais e terminais, mas sem uma semântica definida, ou seja, uma árvore não funcional. Fundamental para o processo de avaliação é poder executar o programa, e os componentes chaves que de fato importam nesta etapa são os símbolos terminais (hierarquicamente estruturados), que formam a sentença da derivação. No entanto, estes símbolos terminais não têm "vida", necessitam portanto serem vinculados às rotinas que computam funções, retornam constantes ou trabalham apropriadamente as variáveis (ou atributos). Em outras palavras, os não-terminais não participam do processo de execução.

A Tabela 3.5 mostra a amarração para a gramática G_{iris} que provém a semântica funcional dos símbolos terminais. Nota-se, como esperado, que os atributos (variáveis), cons-

Σ		Retorno	Função	Argumentos
se-então-senão	$\rightarrow$	classe	SeEntãoSenão	(lógico, classe, classe)
e	$\rightarrow$	lógico	E	(lógico, lógico)
ou	$\rightarrow$	lógico	Ou	(lógico, lógico)
não	$\rightarrow$	lógico	Não	(lógico)
igual	$\rightarrow$	lógico	Igual	(numérico, numérico)
maior	$\rightarrow$	lógico	Maior	(numérico, numérico)
menor	$\rightarrow$	lógico	Menor	(numérico, numérico)
$const_1 \ldots const_n$	$\rightarrow$	numérico		
$atr_1 \ldots atr_4$	$\rightarrow$	numérico		
$setosa, versic., virginica$	$\rightarrow$	classe		

Tabela 3.5: Ligações funcionais dos símbolos terminais da gramática G_{iris}.

tantes e as classes, funções de aridade zero, apenas retornam valores, isto é, não requerem argumentos. É conveniente ressaltar que para os atributos, ou variáveis do programa, os valores retornados são dinâmicos, ou seja, dependendo da amostra comparada, os atributos terão um conjunto próprio de valores retornados para o programa.

O Algoritmo 3.5 computa recursivamente o resultado da execução de um programa. O argumento requerido é um símbolo terminal, podendo ser uma função, constante, atributo ou classe (respeitando-se a estrutura imposta pela gramática). Inicia-se a execução passando como argumento o primeiro símbolo terminal da árvore sintática. Tendo a estrutura gramatical G_{iris} como referência, a função ExecutaPrograma poderia ser semeada com o terminal *se-então-senão* ou qualquer uma das classes, *setosa*, *versicolour* ou *virginica*.

Algoritmo 3.5 ExecutaPrograma (T)

Entrada: símbolo terminal T $\in \Sigma$
Retorno: o valor de T ou o resultado da execução da função ligada a T

```
se NúmeroArgumentos(T) > 0 então
   retorne Função[T](ExecutaPrograma (T.arg_1), ..., ExecutaPrograma (T.arg_n));
senão
   retorne T;
fim se
```

O cômputo da avaliação de um indivíduo pode ser realizado chamando-se a função ExecutaPrograma para cada exemplo do conjunto de treinamento e, então, calculando a

disparidade acumulada entre a saída da função e o valor esperado, ao longo das execuções.

3.3.2.4 Seleção dos Indivíduos Promissores

A seleção na PGG segue exatamente a forma como é realizada na programação genética convencional.

3.3.2.5 Operadores Genéticos

Os dois grandes operadores genéticos utilizados na PG e PGG são o cruzamento e a mutação padrão [19]. O primeiro trata da recombinação do material genético, enquanto o segundo causa mudanças aleatórias em algum ponto do genótipo do indivíduo (Capítulo 2).

As adaptações necessárias para a integração com a programação genética gramatical concernem em manter coesa a estrutura definida pela gramática. Isto é, os operadores genéticos só podem ser praticados respeitando-se as imposições definidas pelos símbolos não-terminais da árvore de derivação. As restrições visam garantir que os novos indivíduos criados/modificados também sejam membros da linguagem definida pela gramática do problema.

Diferentemente da avaliação, o papel dos símbolos não-terminais nas operações de cruzamento e mutação torna-se evidente, enquanto os símbolos terminais e a estrutura da árvore tornam-se dispensáveis.

Cruzamento

O *crossover* cria novos indivíduos misturando o conteúdo genético de seus pais. Na codificação por árvores de decisão, o cruzamento significa a troca de sub-árvores, gerando dois novos programas.

Os pontos de cruzamento estão restritos a operarem somente sobre os símbolos não-terminais, e ele é realizado da seguinte forma. Escolhe-se inicialmente um símbolo não-terminal do primeiro indivíduo. Em seguida, seleciona-se um não-terminal na árvore

[19] Outros operadores vistos no capítulo 2, como a mutação alelo, mutação de encolhimento (*shrink*), permutação e edição, podem também ser agregados à PGG. O requisito é que estas operações sejam realizadas obedecendo-se a estrutura gramatical da árvore, de forma análoga à adaptação dos operadores padrões de *crossover* e mutação.

do segundo pai, mas com a restrição de que este símbolo seja igual ao selecionado no primeiro indivíduo. Em outras palavras, as raízes das sub-árvores selecionadas dos indivíduos pais têm que compartilhar o mesmo símbolo não-terminal. A Figura 3.10 mostra graficamente uma aplicação do operador *crossover*. Os indivíduos **A** e **B** são selecionados para a recombinação genética, sob algum critério de seleção previamente definido. Sorteia-se um símbolo não-terminal em **A** e **B**, no exemplo o símbolo <*relacional*>. As sub-árvores enraizadas pelo símbolo escolhido são então comutadas, cada qual gerando um novo indivíduo, **A'** e **B'**.

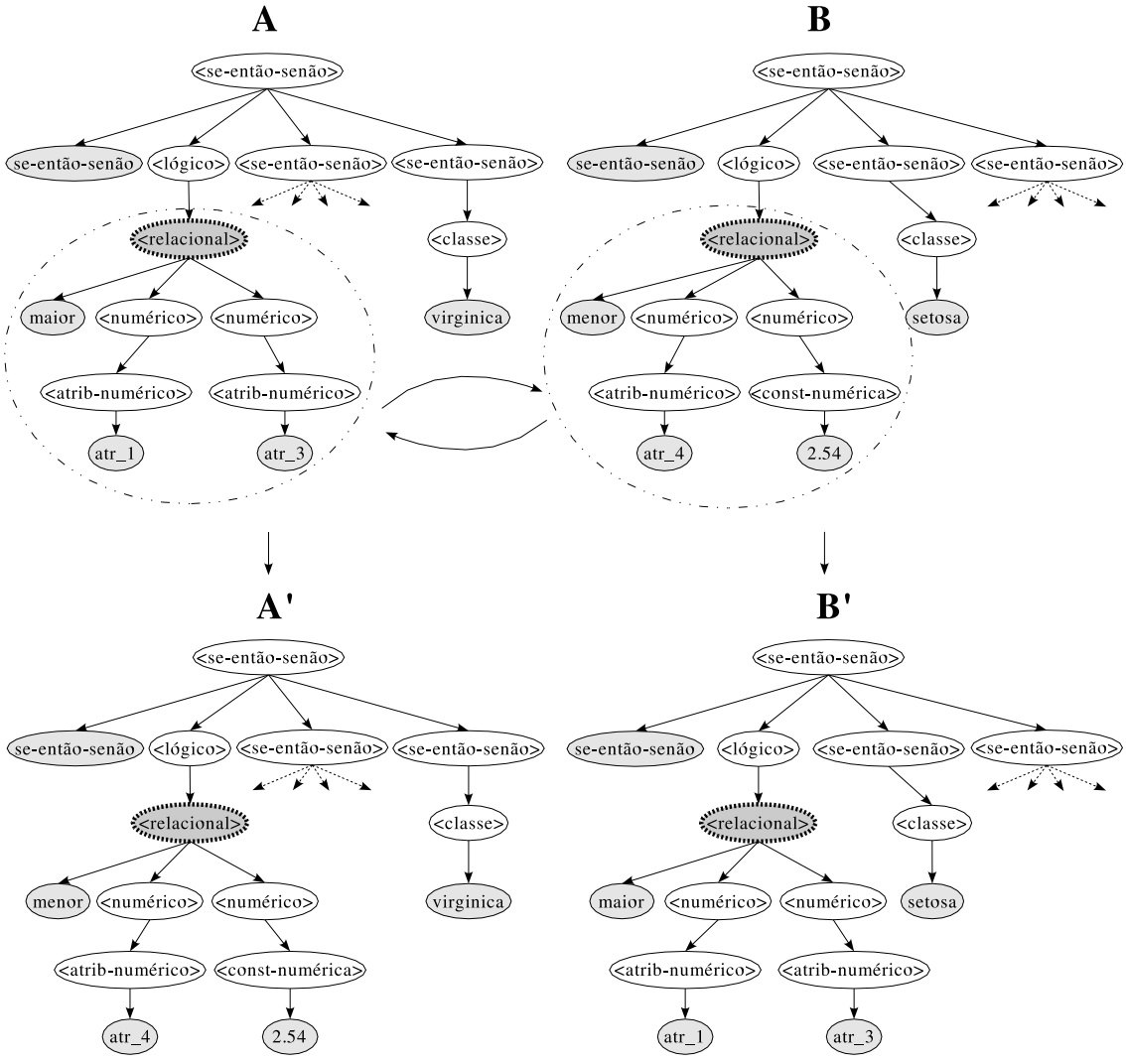

Figura 3.10: O cruzamento na programação genética gramatical.

É de se esperar que, ocasionalmente, o símbolo não-terminal escolhido no primeiro pai não faça parte do repertório de símbolos do segundo pai. Pode-se pensar em simplesmente

anular o cruzamento e seguir com a escolha de novos indivíduos pais. Alternativamente, é preferível tentar selecionar outros pontos de corte (símbolos não-terminais) dos indivíduos em processo de *crossover*. O Algoritmo 3.6 mostra o procedimento de cruzamento, incluindo o artifício de tentativas extras. Um novo argumento $tentativas_{max}$ é fornecido como parâmetro a fim de prevenir que o sistema desperdice consideráveis recursos na tentativa infrutífera de selecionar dois pontos compatíveis em indivíduos que não tenham não-terminais em comum [20]. É possível ainda descrever o processo de cruzamento de

Algoritmo 3.6 Cruzamento $(\Delta_1, \Delta_2, tentativas_{max})$

Entrada: Árvores de derivação pais Δ_1 e Δ_2, e $tentativas_{max}$
Retorno: Árvores de derivação geradas δ_1 e δ_2

$tentativas \leftarrow 0$;
enquanto não houver *crossover* e $tentativas < tentativas_{max}$ **faça**
  $ponto_1 \leftarrow$ SorteiaPonto(Δ_1);
  $ponto_2 \leftarrow$ SorteiaPonto(Δ_2) | NãoTerminal($ponto_2$) = NãoTerminal($ponto_1$);
  se encontrou em Δ_2 o mesmo símbolo não-terminal sorteado em Δ_1 **então**
    $\delta_1 \leftarrow \Delta_1 -$ SubÁrvore($ponto_1$) + SubÁrvore($ponto_2$);
    $\delta_2 \leftarrow \Delta_2 -$ SubÁrvore($ponto_2$) + SubÁrvore($ponto_1$);
    retorne δ_1 e δ_2;
  senão
    $tentativas \leftarrow tentativas + 1$;
  fim se
fim enquanto
retorne não foi possível realizar o *crossover*;

forma linear, em termos de como as sentenças são misturadas. Sejam duas árvores com as derivações,

$$S \stackrel{+}{\Rightarrow} x_1 \ldots x_n \text{ <a> } z_1 \ldots z_i \stackrel{+}{\Rightarrow} x_1 \ldots x_n y_1 \ldots y_m z_1 \ldots z_i$$

$$S \stackrel{+}{\Rightarrow} a_1 \ldots a_j \text{ <a> } c_1 \ldots c_k \stackrel{+}{\Rightarrow} a_1 \ldots a_j b_1 \ldots b_r c_1 \ldots c_k$$

o *crossover* sobre o símbolo não-terminal <a> produzirá duas novas sentenças da forma

$$S \stackrel{+}{\Rightarrow} x_1 \ldots x_n \text{ <a> } z_1 \ldots z_i \stackrel{+}{\Rightarrow} x_1 \ldots x_n b_1 \ldots b_r z_1 \ldots z_i$$

$$S \stackrel{+}{\Rightarrow} a_1 \ldots a_j \text{ <a> } c_1 \ldots c_k \stackrel{+}{\Rightarrow} a_1 \ldots a_j y_1 \ldots y_m c_1 \ldots c_k$$

[20] Na realidade, todo indivíduo construído sob a mesma gramática possui pelo menos um símbolo não-terminal em comum: o símbolo de partida. Entretanto, o símbolo inicial representa a raiz da árvore de derivação, e o cruzamento sobre a raiz é sinônimo de clonagem, ou seja, não são misturados os materiais genéticos. Sendo assim, o procedimento de *crossover* pode agregar de forma vantajosa a restrição de corte apenas em nós abaixo (filhos) da raiz.

O *crossover* seletivo $\otimes$, nomeado assim por Whigham, permite que o cruzamento atue em um subconjunto de símbolos não-terminais e/ou com diferentes proporções de seleção. Por exemplo, na Tabela 3.6 a operação de cruzamento ocorreria com a probabilidade de 40% para os não-terminais {<se-então-senão>, <lógico>, <classe>} e 50% para os símbolos {<relacional>,<atrib-numérico>}.

Subconjunto admitido	Probabilidade
$\otimes = \{$<se-então-senão>, <lógico>, <classe>$\}$	40%
$\otimes = \{$<relacional>, <atrib-numérico>$\}$	50%

Tabela 3.6: Exemplo de parâmetros para o *crossover* seletivo sob a gramática G_{iris}.

As alterações no Algoritmo 3.6 relativas à introdução do cruzamento seletivo são triviais, bastando delimitar a seleção de não-terminais aos estipulados nos parâmetros, bem como agir em função da probabilidade. Um detalhe, no entanto, refere-se a possibilidade de uma das árvores não possuir qualquer símbolo não-terminal apresentado nos parâmetros. Neste caso, o *crossover* é retornado como nulo.

Mutação

O objetivo da mutação é modificar parte do material genético de um indivíduo de forma aleatória, buscando assim explorar novas direções e aumentar a diversidade. De forma análoga à natureza, a ocorrência da mutação na programação genética geralmente é rara, ocorrendo sob baixas probabilidades.

A mutação segue a mesma filosofia do cruzamento na PGG, isto é, está condicionada ao símbolo não-terminal sorteado. Diferentemente do *crossover*, a mutação opera sobre um único indivíduo por vez. Sinteticamente, escolhe-se aleatoriamente um símbolo não-terminal da árvore de derivação do programa, então, remove-se esta sub-árvore indexada pelo símbolo e a substitui por uma outra, criada como na geração inicial do indivíduo, porém, o símbolo inicial é o não-terminal sorteado. A Figura 3.11 ilustra o processo de mutação na programação genética gramatical. Primeiro sorteia-se um símbolo não-terminal da árvore **A**, <*se-então-senão*>, elimina-se a sub-árvore enraizada pelo símbolo e então a substitui por outra cuja raiz também é o não-terminal <*se-então-senão*>, gerando a árvore **A'**.

O Algoritmo 3.7 realiza a mutação sobre uma árvore de derivação, aceitando também

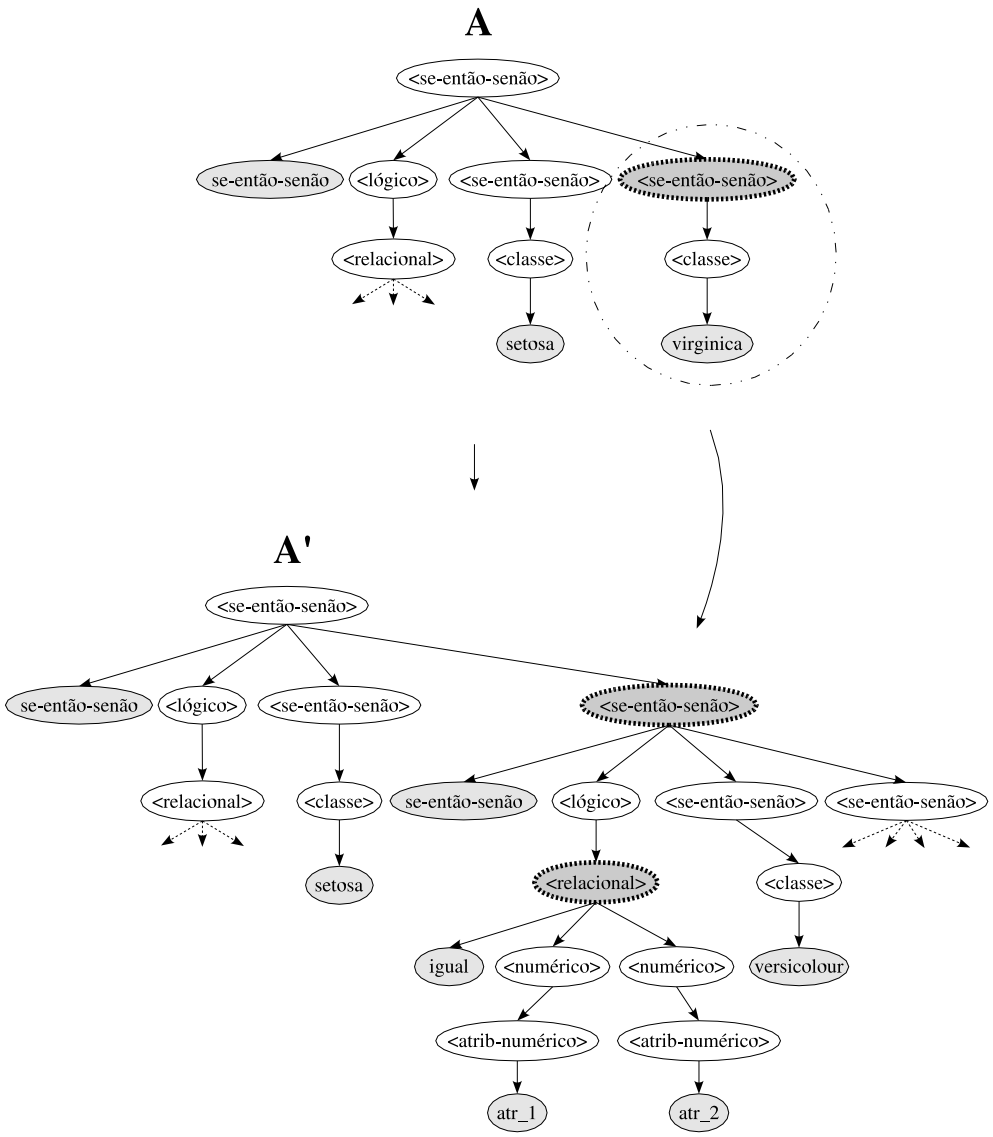

Figura 3.11: A mutação na programação genética gramatical.

como argumento a profundidade máxima de mutação. Nota-se que o procedimento de mutação recorre à função de criação de derivações CriaÁrvore, definida pelo Algoritmo 3.4.

Descrita linearmente, tem-se a mutação sobre o símbolo não-terminal <a> da sentença

$$S \overset{+}{\Rightarrow} x_1 \ldots x_n \text{ <a> } z_1 \ldots z_i \overset{+}{\Rightarrow} x_1 \ldots x_n y_1 \ldots y_m z_1 \ldots z_i$$

modificada para

$$S \overset{+}{\Rightarrow} x_1 \ldots x_n \text{ <a> } z_1 \ldots z_i \overset{+}{\Rightarrow} x_1 \ldots x_n a_1 \ldots a_m z_1 \ldots z_i$$

De forma semelhante ao cruzamento, pode-se “guiar” a aplicação do operador de

Algoritmo 3.7 Mutação ($\Delta, pmut_{max}$)

Entrada: Árvore de derivação Δ, e a profundidade máxima de mutação $pmut_{max}$
Retorno: Árvore de derivação mudada Δ'

$ponto \leftarrow$ SorteiaPonto(Δ);
$\Delta' \leftarrow \Delta - \text{SubÁrvore}(ponto)$ + CriaÁrvore (NãoTerminal($ponto$), $pmut_{max}$) ;
retorne Δ';

mutação sobre símbolos não-terminais específicos, sob uma certa probabilidade, como a definida na Tabela 3.7. Esta variação é chamada mutação seletiva, e é representada pelo símbolo $\odot$. A alteração no Algoritmo 3.7 é análoga ao do algoritmo de cruzamento e dispensa comentários.

Subconjunto admitido	Probabilidade
$\odot = \{$<lógico>,<classe>$\}$	10%
$\odot = \{$<relacional>$\}$	5%

Tabela 3.7: Exemplos de parâmetros para a mutação seletiva sob a gramática G_{iris}.

3.3.2.6 Reprodução: Inserção dos Novos Indivíduos

O método de inserção é tratado da mesma forma que na programação genética convencional. Basicamente, dependendo do modelo, ou a população é totalmente recriada e substitui a anterior (geracional) ou os novos indivíduos são inseridos juntamente com outros da geração anterior (*steady-state*).

3.3.3 Detalhes de Implementação

3.3.3.1 Gramática

É vantajoso manter a definição da gramática desacoplada do sistema, possibilitando-se assim modificá-la sem a necessidade da recompilação do programa. Contudo, alguns aspectos devem ser respeitados

- qualquer símbolo não-terminal das produções, obrigatoriamente, em algum caminho, leva a um símbolo terminal;

- todo símbolo terminal necessariamente possui uma ligação funcional, que fornece-lhe a capacidade de "execução".

3.3.3.2 Otimização da Execução da Árvore

Na esfera da computação evolucionária, tem-se como maior gargalo, em termos de processamento, o estágio de avaliação de um indivíduo. Na programação genética este fator é ainda mais evidente, pois geralmente uma avaliação é composta de numerosas execuções de um programa. Portanto, todo o esforço de otimização direcionado à execução é valioso e pode ser crucial para a viabilidade do sistema [21].

Na PGG, um indivíduo guarda em sua árvore de derivação tanto informações relativas aos símbolos não-terminais quanto aos terminais. Sabe-se que na etapa de execução os símbolos não-terminais são prescindíveis. A execução de um programa, considerando-se o modelo atual de representação, requer o percurso da raiz até os nós folhas (terminais), visitando-se inutilmente os nós referente aos não-terminais. Este custo aparentemente insignificativo torna-se considerável quando reflete-se sobre a repetividade da tarefa. Estima-se que número de símbolos não-terminais em uma árvore de derivação é bem superior ao número de terminais (vide exemplos da G_{iris}). Ademais, o cômputo de acesso a um nó pode ser mais caro que o próprio cômputo de uma função, por exemplo, relacional. Portanto, é interessante segmentar a árvore de derivação em duas inter-relacionadas, a árvore de símbolos não-terminais e a de símbolos terminais. A Figura 3.12 mostra como pode ser esta representação segmentada. Esta representação segmentada introduz um pequeno acréscimo no tamanho do "objeto" árvore, necessário para acomodar os apontadores (ligações) específicos da árvore de símbolos terminais. Todavia, este adicional pode ser desprezado. Um outro ponto é que esta nova estrutura exige um tratamento de implementação mais cuidadoso, pois o gerenciamento da árvore segmentada é mais complexo. Isto é, toda modificação sobre o indivíduo tem que agir de forma a preservar a forma da árvore segmentada, isto inclui a criação e aplicação dos operadores genéticos.

[21] O número de execuções (interpretações da árvore de derivação) pode ser calculado aproximadamente como o número de gerações $\times$ tamanho da população $\times$ tamanho da amostra de dados (conjunto de treinamento).

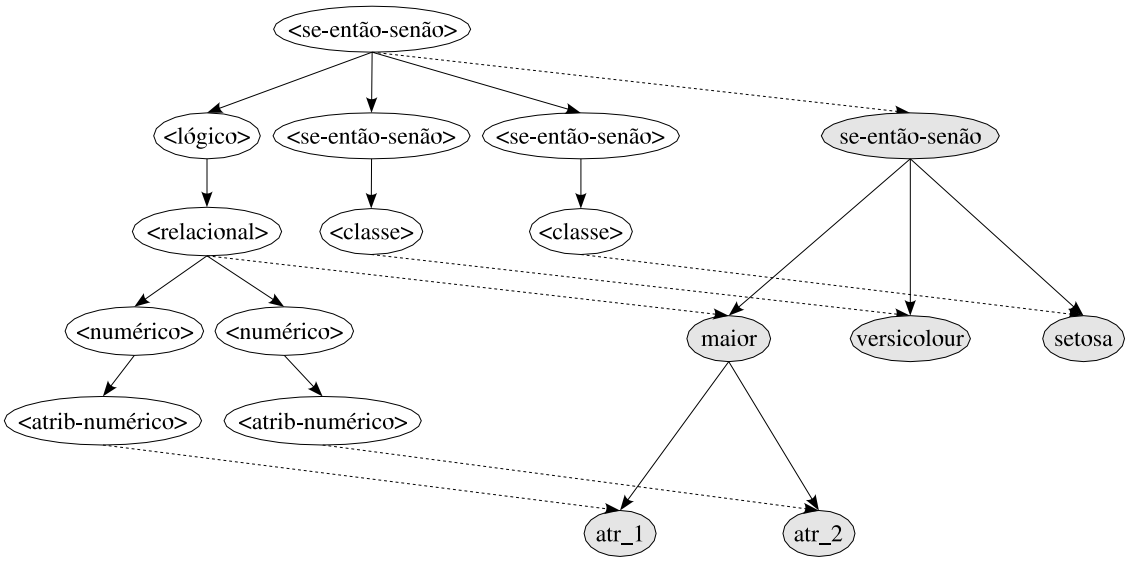

Figura 3.12: Árvore de derivação segmentada.

3.4 Conclusão

Em suma, este capítulo descreveu a união da programação genética com a codificação por gramática livre de contexto, bem como as adaptações necessárias nos estágios evolucionários, tendo a PG convencional como base. Iniciou-se com a introdução do conceito geral de gramática, para então focar-se na GLC. Seguidamente, algumas propostas de representação na PG foram revistas e levantados seus aspectos positivos e negativos. Também comentou-se algumas questões relativas à implementação, como a otimização do núcleo da avaliação.

Mostrou-se que a PGG fornece uma interface clara e exata de representação, permitindo que o processo evolutivo corra rigorosamente de acordo com a estrutura representativa do problema, seja pela distinção dos dados ou pela utilização de relações estruturais mais complexas.

Finalmente, a programação genética gramatical é um dos módulos deste trabalho, e trabalha em conjunto com o modelo co-evolutivo, o qual é assunto do próximo capítulo.

Capítulo 4

Co-evolução Amostra-Classificador

A co-evolução inserida no contexto da computação serve para pelo menos dois propósitos: o melhor entendimento da natureza através da simulação de um modelo simplificado de co-evolução, ou como inspiração para problemas outros, geralmente envolvendo otimização. Dentro do contexto deste trabalho, a co-evolução é utilizada na pretensão "egoísta" de trazer a inteligência biológica, construída durante milhões e milhões de anos, para o universo computacional, que ainda engatinha.

Uma das características louváveis da co-evolução é poder, além de enriquecer a resolução de problemas já atendidos por técnicas tradicionais, ampliar o domínio de aplicações, elevando a utilidade da computação evolucionária. Dentre esse grupo de novos problemas, estão aqueles que não lidam com referenciais absolutos de qualidade de solução, como a descoberta de estratégias de jogos.

O presente capítulo elucida, sob a ótica computacional, conceitos que tangem a co-evolução, destinando sua utilidade ao desenvolvimento de um sistema classificador de amostras de dados. Inicia-se com a explicação e formalização do termo co-evolução, discorrendo e categorizando os variados modelos co-evolutivos. Segue concentrando-se na co-evolução competitiva, citando e descrevendo importantes trabalhos desenvolvidos, inclusive o mais influente nesta dissertação. Também se discute a aptidão cumulativa, que é construída incrementalmente e se mostra promissora. Finalmente, detalha-se o sistema co-evolutivo amostra-classificador, considerando-se a forma acabada, integrado à programação genética gramatical.

4.1 Co-evolução

A interação entre indivíduos, cada qual afetando a probabilidade de sobrevivência/reprodução de outros indivíduos, é um fato intrínseco na natureza, virtualmente afetando qualquer espécie [38] [1]. As interações existem em variadas formas (como por exemplo a competição por recursos naturais ou, a cooperação) e graus (a caça de um predador sobre a presa, bem como a refeição do peixe rêmora obtida dos restos alimentares do tubarão). Na linguagem biológica, este fenômeno é conhecido como simbiose [2].

A seleção natural recíproca de organismos em simbiose é chamada de co-evolução. Em outras palavras, evoluem concomitantemente aqueles seres que de alguma forma são interdependentes. Se o indivíduo **A** afeta o sucesso de sobrevivência e reprodução de **B** (estão em simbiose), então a evolução de **A** implica em uma resposta co-evolutiva de **B**, e vice-versa. A resposta co-evolutiva depende do tipo de relação envolvida entre os organismos [3]. Por exemplo [16], algumas famílias de aves, como o cuco, depositam seus ovos em ninhos alheios a fim de terem seus ovos chocados por outras aves, poupando-se assim preciosos recursos. Entretanto, este comportamento induz perdas ao hospedeiro, que estaria desperdiçando esforços através do compartilhamento de recursos, tornando o processo mais dispendioso e possivelmente comprometendo a sobrevivência da própria prole. Esta tensão acaba criando uma "corrida armamentista", onde o cuco tende a aperfeiçoar o mimetismo dos ovos, enquanto a ave parasitada é forçada a adquirir melhores habilidades de discriminação de ovos. Os filhotes de cuco bem camuflados tenderão a sobreviver e conseqüentemente passar seus genes —de bons enganadores— adiante. Por sua vez, as aves hospedeiras com boa capacidade distintiva poderão distribuir mais recursos à própria prole, aumentando a expectativa de sobrevivência destes, contribuindo assim para que os genes "perspicazes" caminhem adiante.

Um outro exemplo de interação entre espécies é a constituição do líquen [4], como resultado da associação entre algas e fungos. Neste modelo, de benefício mútuo, a alga realiza a fotossíntese e cede ao fungo parte da matéria orgânica sintetizada. O fungo, por

[1] Teoricamente é possível simular um sistema baseado na evolução natural isento de "inter-relação", como no caso da evolução artificial via forma canônica dos algoritmos genéticos.

[2] Apesar da tendência atual de desuso, alguns autores ainda tratam a palavra simbiose no intuito de caracterizar o que é definido neste trabalho como mutualismo.

[3] A Seção 4.1.2 descreve os tipos de co-evolução.

[4] O líquen é um organismo vegetal composto, que vive em lugares inóspitos, como rochas nuas, casca de árvores, deserto e Ártico.

sua vez, fornece proteção e cede-lhe tanto umidade como sais minerais.

4.1.1 Definição Formal

Para tornar o conceito biológico de co-evolução mais preciso e direcionado a este trabalho, faz-se necessário formalizar o conceito de simbiose e co-evolução. Os conceitos definidos aqui seguem os propostos por Morrison [38], e não necessariamente tem caráter unânime.

4.1.1.1 Simbiose

De Bary [17] foi o primeiro a cunhar a palavra simbiose na noção de "convivência entre animais de diferentes espécies", em contraposição ao termo previamente definido por Van Beneden [53], cujo significado aproximava-se dos conceitos de mutualismo e comensalismo. Entretanto, o conceito de De Bary se restringe à ocorrência de simbiose entre espécies, excluindo a relação simbiótica entre indivíduos. A Definição 4.1.1 amplia esta visão e engloba interações inter-individuais, abrangendo inclusive relações de parentesco.

Definição 4.1.1 *A simbiose é definida como o relacionamento entre dois indivíduos onde a aptidão* [5] *de um indivíduo afeta diretamente a aptidão do outro.*

Entende-se por "afeta diretamente" as interações indivíduo-indivíduo, onde não há intermédio de terceiros, não importando ainda com que intensidade esta é apresentada. Um gnu está em simbiose com a vegetação de que se alimenta (e vice-versa), mas um eventual predador (carnívoro) de gnus não é simbionte com a vegetação que alimenta o gnu, apesar de afetá-la —indiretamente [6]. Por outro lado, a definição de simbiose permite também sua ocorrência além do relacionamento "corpo-a-corpo". Por exemplo, pode-se imaginar em um cenário de escassez de recursos hídricos, um animal **A** se hidrata em uma certa fonte, afetando diretamente a aptidão do animal **B**, que posteriormente alcança a mesma fonte. Neste caso, o animal **A** modifica o ambiente que repercute diretamente em **B**, sem a mediação de outro indivíduo, portanto, há simbiose.

[5] A aptidão de um indivíduo é referenciada neste contexto como a probabilidade/capacidade de sobrevivência e reprodução.

[6] Para efeitos didáticos, foi desconsiderada uma possível simbiose entre o predador e a vegetação, sem o intermédio do gnu. Esta relação poderia ser simbiótica se, por exemplo, o predador eventualmente ingerisse vegetais para limpeza intestinal.

Ainda, a simbiose não exige interações duradouras, podendo ser caracterizada mesmo por encontros casuais. Um exemplo típico é o observado no contágio de humanos com o vírus transmissor da caxumba (parotidite epidêmica), que pode levar à esterilidade. A relação pode ser única, no entanto lesa a capacidade de sobrevivência e/ou reprodução, por conseguinte é de natureza simbiótica.

4.1.1.2 Co-evolução

Angeline e Pollack [4], no contexto dos algoritmos genéticos, definem a co-evolução [7] como o processo evolutivo em que a função de aptidão dos indivíduos depende em algum grau da população. Todavia, esta concepção não engloba as relações inter-populacionais. A Definição 4.1.2 provê um conceito mais preciso, porém abrangente, valendo-se da noção de simbiose.

Definição 4.1.2 *A co-evolução intra-populacional ocorre quando parte de seus indivíduos estão em simbiose entre si. A co-evolução entre populações ocorre quando indivíduos de uma população estão em simbiose com indivíduos de outra.*

Nota-se que a Definição 4.1.2 também prevê a co-evolução interna à própria população, pelo fato da possível existência de simbiose entre seus membros. Por exemplo, é perfeitamente viável imaginar situações em que a relação entre indivíduos da mesma população possa cada qual contribuir com sua sobrevivência, afetando-se assim suas aptidões. Seria portanto insensato tratar a evolução desta população sem considerar a interação entre seus indivíduos.

4.1.2 Tipos de Co-evolução

Como levantado anteriormente, as relações simbióticas —e por conseqüência a co-evolução— podem se apresentar em variadas formas. Algumas destas interações conseguem ser mutuamente benéficas, enquanto outras beneficiam apenas uma parte sem detrimento da outra, bem como ainda relações que envolvem algum tipo de disputa, havendo conflito de interesses.

[7] Os autores referiam-se à co-evolução competitiva.

4.1.2.1 Notação Simbólica e Representação Gráfica

Aliados à descrição textual dos tipos co-evolutivos, estão a notação por meio de símbolos e a representação gráfica [8]. Ambos clarificam e ao mesmo tempo sintetizam o teor de uma forma particular de co-evolução.

Supondo que um indivíduo A esteja em simbiose com o indivíduo B, dependendo da natureza da relação de convivência de ambos, estas são representadas como:

- $A \rightharpoonup B$ quando a aptidão de A afeta diretamente a aptidão de B.
- $A \overset{+}{\rightharpoonup} B$ quando uma mudança na aptidão de A afeta no mesmo sentido a aptidão de B. Isto é, se a aptidão de A aumenta, a de B também aumenta; se a aptidão de A diminui, diminui-se a aptidão de B.
- $A \overset{-}{\rightharpoonup} B$ quando uma mudança na aptidão de A repercute inversamente na aptidão de B. Em outras palavras, se A tem sua aptidão aumentada, B tem sua aptidão diminuída; se a aptidão de A diminui, a aptidão de B aumenta.

Graficamente, a simbiose é representada por arcos orientados e sinalizados, onde os sinais denotam o mesmo significado dos da representação simbólica. A Figura 4.1 mostra uma relação simbiótica apresentada em forma de arcos.

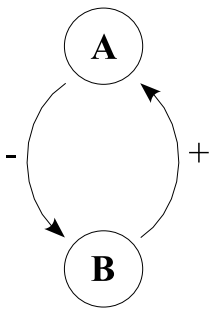

Figura 4.1: Representação gráfica da simbiose $A \overset{-}{\rightharpoonup} B$ e $B \overset{+}{\rightharpoonup} A$.

4.1.2.2 Comensalismo

Neste tipo de co-evolução, um organismo (comensal) se beneficia de outro (hospedeiro) sem que este seja prejudicado. Simbolicamente esta simbiose é representada por

[8] Também conhecida como gráfico de simbiose ou gráfico simbiótico.

$Hospedeiro \overset{+}{\rightarrow} Comensal$. A Figura 4.2 mostra graficamente esta relação simbiótica. Nota-se que é uma interação de fluxo único, não havendo resposta do comensal [9].

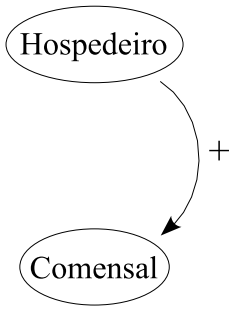

Figura 4.2: Comensalismo.

O exemplo citado inicialmente neste capítulo, envolvendo os peixes tubarão e rêmora, é um tipo de comensalismo. A rêmora astuciosamente segue o tubarão em suas viagens de caça, aproveitando-se dos restos alimentares deixados por ele. Uma mudança na aptidão do tubarão, resultando em um maior poder de caça, afeta diretamente as chances de sobrevivência/reprodução da rêmora. Se esta mudança é positiva, a rêmora ganha com a maior abundância de alimentos, melhorando assim sua aptidão; caso contrário, haverá uma menor oferta alimentícia, implicando em perdas para a rêmora.

Duas subdivisões do comensalismo podem ser destacadas:

- Inquilinismo – é a associação em que um organismo procura abrigo em seu hospedeiro, sem prejudicá-lo. Pode ser citada a relação entre o peixe-agulha e a holotúria (pepino-do-mar), onde o peixe-agulha abriga-se no interior do corpo da holotúria, saindo apenas para se alimentar.

- Foresia – ocorre quando um organismo é transportado de "carona" pelo seu hospedeiro. Observa-se, por exemplo, o transporte de sementes por pássaros ou insetos.

4.1.2.3 Amensalismo

O amensalismo é a forma de co-evolução que se apresenta quando dois indivíduos interagem entre si, sendo que uma melhora na aptidão do hospedeiro implica unilateralmente em prejuízo ao organismo amensal. Não há retaliação por parte da "vítima", ou

[9] Este tipo de relação é conhecida como desprovida de retroalimentação (*feedback*).

seja, assim como o comensalismo, possui fluxo único. Formalmente esta relação simbiótica é descrita por $Hospedeiro \overset{-}{\rightarrow} Amensal$. O gráfico simbiótico é apresentado pela Figura 4.3.

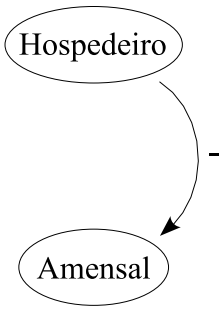

Figura 4.3: Amensalismo.

Algumas algas planctônicas dinoflageladas, quando vivem em ambientes favoráveis, alcançam alta taxa populacional, liberando grandes quantidades de substâncias tóxicas na água. Este fenômeno, também conhecido como maré vermelha, leva à morte por intoxicação vários seres aquáticos. Um outro exemplo de amensalismo vem de certas espécies de árvores que possuem substância inibidoras em suas folhas, que quando caem no solo impedem a germinação de sementes.

4.1.2.4 Mutualismo

A co-evolução por mutualismo envolve relações harmônicas onde os indivíduos são mutuamente beneficiados. É basicamente uma "troca de favores", ou "política de boa vizinhança". Alguns autores consideram o mutualismo estritamente como uma associação íntima, em que a sobrevivência dos seres envolvidos não seria viável se estes fossem divorciados. Às relações não dependentes —os organismos vivem se separados— chamam-na de *protocooperação* ou *mutualismo facultativo*.

Descreve-se a simbiose mutualista por $Simbionte\ A \overset{+}{\rightarrow} Simbionte\ B$ e $Simbionte\ B \overset{+}{\rightarrow} Simbionte\ A$. O gráfico de simbiose pode ser visualizado na Figura 4.4.

Percebe-se que o mutualismo possui um fluxo duplo, onde há realimentação. Considere o indivíduo **A** em simbiose com **B**. Uma melhora na aptidão de **A** repercute positivamente em **B**, que por sua vez retorna ao indivíduo **A** (também positivamente), como um "reflexo" [10]. O contrário também é verídico, uma baixa em **A** (ou **B**) ecoa negativamente

[10]Naturalmente, a intensidade do impacto da realimentação não é obrigatoriamente uma, tal que seria

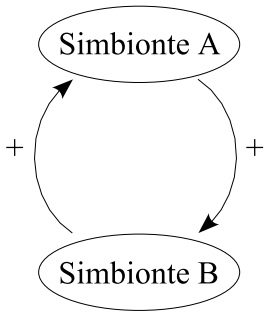

Figura 4.4: Mutualismo.

em **B** (ou **A**), e então realimenta **A** (ou **B**).

O pássaro-palito e o crocodilo africano são exemplo de espécies em mutualismo [11]. O pássaro-palito eventualmente penetra na boca do crocodilo alimentando-se de restos alimentares e vermes existentes na boca do réptil. Por outro lado, o crocodilo livra-se de organismos nocivos, como os vermes e parasitas. Uma maior acuidade do pássaro na captação de alimentos (fortemente ligado com a aptidão, adaptabilidade) faz de alguma forma com que os crocodilos, isentos de parasitas, tenham melhor disposição para a caça, e conseqüentemente obtêm mais refeições —ou parasitas/vermes no ponto de vista do pássaro-palito.

4.1.2.5 Predatismo

Também conhecido como parasitismo (parasita-hospedeiro) [12], este modelo co-evolutivo aplica-se nas relações simbióticas em que existam práticas predatórias. Em outras palavras, o organismo predador extrai vantagens da presa, enquanto esta sofre detrimentos.

Representa-se simbolicamente por $Predador \xrightarrow{-} Presa$ e $Presa \xrightarrow{+} Predador$. A Figura 4.5 mostra graficamente o fluxo de interação entre predador e presa.

Por tratar-se de relações freqüentes na natureza (no nível macroscópico), são inúme-

capaz de provocar um círculo vicioso de respostas e contra-respostas.

[11] O exemplo encaixa-se melhor como mutualismo facultativo (protocooperação).

[12] Alguns autores consideram o predatismo e parasitismo como formas distintas de simbiose. A diferença principal reside no fato de que o predatismo envolve o sacrifício da presa, enquanto que no parasitismo o parasita sobrevive às custas do hospedeiro, sendo este apenas haurido (mesmo que eventualmente custe-lhe a vida). Difere-se do predatismo também por: maior especialização, maior taxa de reprodução e habitat no hospedeiro.

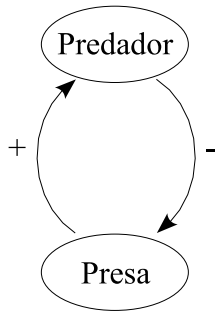

Figura 4.5: Predatismo.

ros os casos simbióticos que se encaixam no predatismo. Seja o exemplo da família de predadores felinos e suas presas comum, os antílopes. Se o número de predadores felinos aumenta, devido a um acréscimo na sua aptidão, os antílopes tendem a ser mais caçados, e portanto suas chances de sobrevivência e reprodução serão prejudicadas. Do contrário, se por algum motivo os felinos tenderem à extinção (baixa aptidão, pouca adaptabilidade), os antílopes inclinariam-se a prosperar (acréscimo na aptidão). Partindo-se da ótica das presas, se estas baixam suas aptidões, igualmente tenderiam os predadores felinos, pois a oferta alimentícia estaria comprometida. Ainda, o sucesso populacional dos antílopes — significando boa capacidade de sobrevivência e reprodução— disponibilizaria um número maior de refeições aos felinos, com conseqüente melhora na aptidão destes.

A co-evolução atua de maneira a tornar predador e presa os mais hábeis possíveis, porém com objetivos opostos. Ao mesmo tempo em que a seleção natural favorece os felinos mais ágeis, com visão mais aguçada, enfim, melhores caçadores, são selecionados também os antílopes com maior poder de fuga, seja os indivíduos com maior capacidade perceptiva, velozes, etc.

O conceito de predatismo não é preciso, podendo oscilar e se confundir com a co-evolução competitiva (vide 4.1.2.6). Seguindo o exemplo da simbiose entre felinos e antílopes, é plausível imaginar que um acréscimo na capacidade de reprodução/sobrevivência dos antílopes seja em decorrência da evolução de qualidades de escape, isto é, presas menos vulneráveis aos felinos. Portanto, espera-se que a aptidão dos predadores baixe, em função da menor disponibilidade alimentícia. Isto sugere que o fluxo presa-predador seja $Presa \overset{-}{\rightharpoonup} Predador$, análogo à forma competitiva.

4.1.2.6 Competição

A competição se apresenta quando indivíduos pleiteiam recursos, geralmente limitados. Os recursos podem ser alimentação, água, território, direito de acasalamento, entre outros. Isto significa que quanto maior a aptidão de um competidor, obrigatoriamente será menor a aptidão do outro, e vice-versa. Em outras palavras, quanto mais recursos um competidor obtém, menos restará ao concorrente, afetando assim suas chances de sobrevivência e reprodução.

A relação simbiótica de competição é, portanto, descrita por $Competidor\ A \overset{-}{\rightharpoonup} Competidor\ B$ e $Competidor\ B \overset{-}{\rightharpoonup} Competidor\ A$. Pode-se também exibi-la graficamente, como na Figura 4.6.

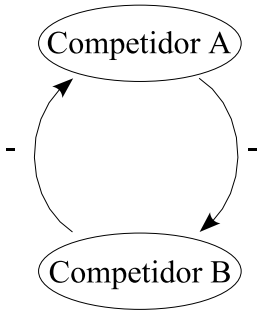

Figura 4.6: Competição.

A competição tende a se agravar quando espécies de animais ou plantas têm necessidades semelhantes e vivem sob recursos limitados. Neste caso a disputa torna-se freqüente e acirrada, obrigando muitas vezes espécies em desvantagem competitiva a migrarem para outros nichos ecológicos.

Os exemplos de competição na natureza também são abundantes, seja intra-específica (como disputas pelo harém) ou interespecífica (por território, alimentação, etc.). Entre espécies, tem-se também, por exemplo, as eternas batalhas entre felinos e hienas, que disputam, entre outras coisas, presas e territórios. Ou ainda, a competição por pastagens entre o gado e gafanhotos. Na forma intra-específica, a emigração forçada dos lêmingues [13] em resposta à superpopulação é um outro exemplo de competição.

Percebe-se que a co-evolução competitiva impõe uma alta pressão de seleção baseada

[13] Designação de pequenos roedores dos gêneros *Lemmus* e *Dicrostonyx*, das regiões árticas, que se alimentam de cascas de árvores, musgos, líquens e raízes. A espécie européia L. lemmus é conhecida pela migração em massa, e que, quando chega ao mar, nele se precipita em grande número.

em quão competitivo é o cenário de disputa. A competição conta com graduações de intensidades diferentes, variando da mais leve e esporádica disputa até constantes e mortais confrontos. Em cada uma destas, a natureza recompensa aquela parte melhor adaptada para angariar recursos, sejam quais forem. Esta pressão sobre os competidores força-os a desenvolverem novos e melhores arsenais a fim de vencerem a concorrência. Se por exemplo competidores de uma espécie **A** evoluem um tipo de arma eficaz, certamente estarão em vantagem sobre a espécie concorrente **B**, trazendo detrimento à aptidão desta. Todavia, este fenômeno acresce a pressão de seleção em **B**, que forçosamente privilegia nesta espécie aqueles indivíduos que de alguma forma possam fazer frente ao novo ferramental da espécie **A**, com o risco de extinção se nenhuma contra-arma de sucesso for proposta[14].

A noção de co-evolução por competição extrapola o universo "natural" e torna-se aplicável em diversos outros domínios, especialmente nas relações humanas. É por exemplo o caso notório e conhecido da chamada *corrida armamentista*, um termo cunhado durante a Guerra Fria, que descrevia a co-evolução armamentista entre os Estados Unidos e a União Soviética. Cada uma das nações objetivava ter em mãos o maior poderio bélico, visando a dominação em um possível confronto militar. Para tanto, além do recrutamento interno para pesquisa e desenvolvimento de novas armas de destruição, as superpotências também investiam em tecnologia de espionagem [15], possibilitando assim conhecer o potencial do inimigo e então focalizar-se em meios de anulá-lo ou superá-lo. O paralelo com a natureza biológica é direto. A soberania (aptidão) de uma das nações é inversamente proporcional ao poder bélico de outra.

Do ponto de vista da computação evolucionária, a co-evolução por competição é naturalmente uma ferramenta atraente. A pressão evolucionária criada pelo conflito de interesses tende a proporcionar indivíduos melhor adaptados, indivíduos capazes de realizar progressivamente melhor determinadas tarefas. A seção seguinte descreve os trabalhos em computação evolucionária inspirados no modelo competitivo de co-evolução.

[14]Em populações em co-evolução contínua ao longo do tempo, é incomum o extermínio total de uma população em função de um aperfeiçoamento na capacidade de competição de outra. Explica-se pela baixa probabilidade de evoluir em uma das populações meios infalíveis de captação de recursos, de maneira que a população concorrente não tenha tempo hábil de evoluir uma contra resposta. Entretanto, é eminente o risco de extinção em um possível encontro de populações com interesses semelhantes e que vivem afastadas fisicamente.

[15]Como não houve de fato o confronto "físico" entre as superpotências durante a Guerra Fria, a espionagem era o meio de se conhecer (medir) a força armamentista da nação inimiga. Na natureza encontram-se exemplos de competições sem o envolvimento de contato físico. Estas "batalhas" muitas vezes se dão por gestos e exibicionismo.

4.2 Trabalhos em Co-evolução Competitiva

A despeito de seu potencial virtuoso como ferramenta de otimização, a co-evolução não tem, historicamente, sido bem explorada [4, 38]. Contudo, dentre as formas de co-evolução usadas na literatura de computação evolucionária, a mais largamente empregada em problemas de otimização é a competitiva [16]. Seu sucesso pode ter origem no amplo leque de potenciais aplicações, conceito direto, e pela sua exímia capacidade como instrumento de otimização.

Esta seção segue com a descrição de alguns trabalhos em computação evolucionária considerados ícones no ramo de co-evolução por competição.

4.2.1 O Dilema do Prisioneiro Iterado

Um dos trabalhos pioneiros na área foi desenvolvido por Axelrod [6] em 1987. O objetivo de Axelrod era descobrir (evoluir) estratégias de comportamento diante do jogo "Dilema do Prisioneiro Iterado", ou simplesmente DPI [17].

O DPI é a versão iterativa do clássico Dilema do Prisioneiro (DP). No DP, dois homens são presos acusados de um crime em conjunto. No entanto, a polícia não tem provas suficientes sobre a real autoria e participação de cada um dos presos. Diante disso, os homens são isolados —de modo que não podem se dialogar— e postos sob interrogatório. Durante este processo, as autoridades policiais questionam cada suspeito sobre sua participação. Os presos são limitados a dois tipos de respostas: acusar o companheiro ou permanecer em silêncio. Após serem interrogados, as respostas dos dois homens são confrontadas, cada combinação resultando em apropriadas penas de detenção. Se um suspeito acusa seu companheiro, e este por sua vez fica em silêncio, o delator é absolvido, enquanto o acusado recebe uma pena de dez anos de prisão. Caso os dois permaneçam em silêncio, recebem igualmente uma pena de seis meses. Se porventura ambos os homens mutuamente se denunciarem, são condenados a seis anos. A Tabela 4.1 sintetiza as penalidades para cada combinação de comportamento.

[16] No entanto, bons resultados têm sido obtidos com outros modelos co-evolutivos. De Jong e Potter [18], por exemplo, reportaram sucesso na utilização de formas cooperativas de co-evolução.

[17] Do inglês *Iterated Prisoner's Dilemma* – IPD. O dilema do prisioneiro é um problema clássico da teoria dos jogos, formulado pela primeira vez pelo matemático Albert W. Tucker, que tem interessantes repercussões na filosofia política e mesmo na ciência política.

	A em silêncio	**A acusa B**
B em silêncio	A e B obtêm 6 meses	B obtêm 10 anos; A é solto
B acusa A	B é solto; A obtêm 10 anos	A e B obtêm 6 anos

Tabela 4.1: Penalidades no jogo Dilema do Prisioneiro.

O objetivo de cada suspeito é, naturalmente, minimizar sua pena de detenção. Entretanto, a condenação depende de como se comporta seu companheiro. Tem-se então criado o dilema: deveria um prisioneiro cooperar (permanecer-se em silêncio) esperando por uma pena mais branda ou acusar seu parceiro, na esperança de que este omita-se e arque solitariamente com a condenação máxima —correndo o risco entretanto de ambos serem egoístas e sofrerem com uma detenção maior?

No Dilema do Prisioneiro Iterado são realizadas repetidas rodadas de interrogatórios, objetivando-se portanto, a minimização da soma total das penalidades. Ainda, cada suspeito tem conhecimento do comportamento anterior de seu comparsa —memória. Isto permite o desenvolvimento de estratégias complexas, baseando-se no histórico de respostas do companheiro.

Em 1984, Axelrod, no seu livro intitulado *The Evolution of Cooperation* [5], promoveu uma competição de estratégias para o DPI, enviadas por especialistas da área de teoria dos jogos. Cada estratégia jogava contra todas as outras, em um número determinado de iterações —mas não de conhecimento dos estrategistas. Curiosamente, a estratégia vencedora era bastante simples, guiada pela reciprocidade: iniciava-se permanecendo em silêncio, e nas demais rodadas apenas repetia o comportamento do oponente. Esta estratégia é denominada *Tit-For-Tat*. Dentre algumas características da TFT, está o fato dela ser "bondosa" [18] (nunca acusa sem ser antes acusada), tem memória curta (apenas uma iteração) e retalia sempre quando o comparsa denunciar.

Em *The Evolution of Strategies in the Iterated Prisoner's Dilemma* [19] [6], Axelrod aplica a co-evolução competitiva, através dos algoritmos genéticos, objetivando evoluir estratégias para o jogo DPI. Cada indivíduo da população representa uma estratégia, e são avaliados por quão bem jogam contra outros indivíduos da população, sendo a avali-

[18] Nesta competição, as oito estratégias mais bem colocadas eram do tipo "bondosa". Uma das conclusões interessantes do trabalho de Axelrod é que além das questões morais de cooperação, é também vantajoso cooperar sob a ótica egoísta. Esta cooperação, todavia, obedece ao padrão "pagar na mesma moeda": se o parceiro não coopera, é castigado, sendo retaliado da próxima vez.

[19] *A Evolução de Estratégias no Dilema do Prisioneiro Iterado*.

ação relativa, variando em função da maestria dos demais membros. Axelrod reportou a convergência para estratégias cooperativas (bondosas), ao estilo *Tic-For-Tat*, sabidamente ótimas táticas de jogo.

Salvo o fato deste problema pertencer a uma classe de problemas bem estudados, onde são conhecidas soluções referenciais (ótimas ou quase ótimas), o seu tratamento seria inadequado usando-se ferramentas convencionais. Este grupo abrange os métodos de referencial estático (como o próprio algoritmo genético canônico), em que a medição de adequabilidade de uma solução candidata é fixa —e portanto impossível de ser determinada quando não se conhece o ponto ótimo. A co-evolução preenche esta deficiência à medida que ela por si só cria dinamicamente o referencial ótimo, que é revisto (geralmente de forma monótona) de acordo com a qualidade das soluções candidatas.

Percebe-se, contudo, que por adotar um esquema de confrontos onde a cada geração toda estratégia da população é testada contra todas as outras —se a população é de tamanho n, n^2 será o número de competições—, este modelo pode não ser escalável para domínios mais complexos, onde empregam-se populações maiores e/ou o custo da competição é caro. A Figura 4.7 transmite a noção gráfica deste tipo de avaliação, conhecida também por competição completa. Na figura, os nós compreendem os indivíduos da população, enquanto as linhas denotam os confrontos.

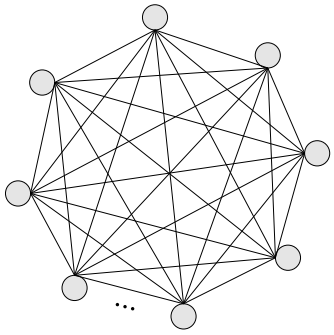

Figura 4.7: Esquema de avaliação por competição completa.

4.2.2 Redes de Ordenação

Hillis [27], também aplicou a co-evolução por competição em um problema bem estudado, as redes de ordenação com 16 dimensões.

Uma rede de ordenação, como o nome indica, efetua a tarefa de ordenação de um determinado número de entradas, através de uma rede que opera seqüencialmente comparações e trocas, como a mostrada graficamente na Figura 4.8, um exemplo com 16 dimensões. Inicialmente, um conjunto não ordenado de números [20] é disposto como entrada da rede. Nos pontos onde existem ligações verticais, o i-ésimo elemento é comutado com o j-ésimo ($i < j$) somente se o valor do elemento da posição j for menor do que o elemento em i.

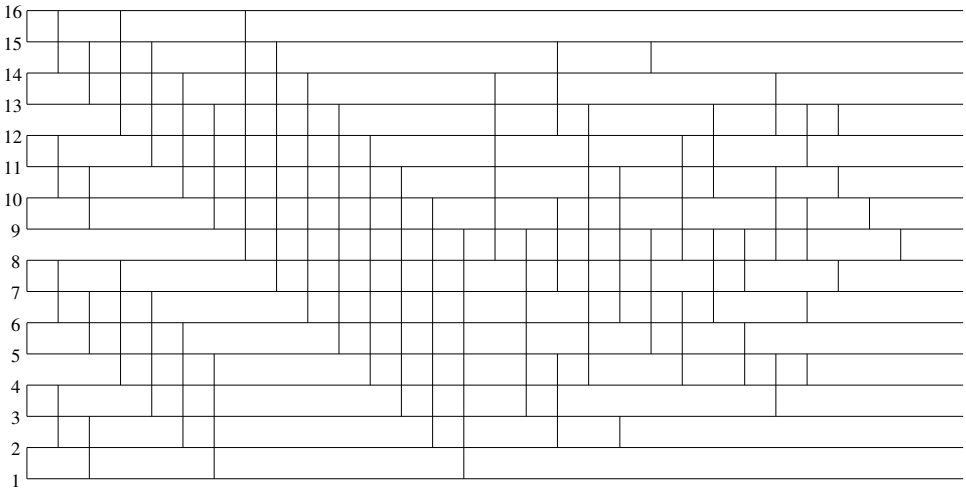

Figura 4.8: Exemplo de uma rede de ordenação.

O problema redes de ordenação compreende dois desafios: 1) definição de um padrão correto de ordenação, isto é, ligações verticais dispostas de tal forma e número a produzirem uma saída correta (todas as entradas devidamente arranjadas); 2) minimizar o número de comparações-trocas (ligações), visando a satisfação de domínios onde soluções parcimoniosas são fundamentais (como circuitos eletrônicos).

Sabe-se que a solução mais econômica até então conhecida, para a rede de ordenação de 16 entradas, compreende um padrão com 60 comparações/trocas, e foi descoberta por Green [31] no ano de 1969.

Hillis codificou as soluções candidatas (redes de ordenação) de modo que o espaço de busca definido englobasse redes que não ordenassem corretamente todas as entradas. Hillis argumentou esta abordagem [21] pela facilidade de se aplicar operadores genéticos

[20] Números ou qualquer entidade em que o conceito de ordenação exista, como palavras ou letras do alfabeto.

[21] Alternativamente, poderia usar-se soluções candidatas que sempre fossem funcionais (ordenação completa), e então procurar por redes com padrões minimalistas.

como mutação e/ou combinação [22] sem criar soluções inválidas, e ainda, pela conveniência de se ter medições parciais de qualidade (baseadas no número de casos ordenados corretamente). O objetivo era, então, encontrar redes pequenas, primando-se por redes com altos índices de sucesso na ordenação.

Os experimentos foram realizados tanto por evolução simples (objetivo fixo) como por co-evolução competitiva (objetivo dinâmico).

No primeiro método, Hillis empregou entradas binárias [23] e verificava-se a discrepância da ordenação realizada por uma rede candidata com os valores esperados (corretamente ordenados). O melhor resultado obtido pela abordagem canônica foi uma rede de ordenação com 65 ligações —5 comparações/trocas a mais que a descoberta por Green.

No intuito de suprir as deficiências do método clássico, como a perda de diversidade (culminando em ótimo local) e avaliação lenta (2^{16} testes para cada candidato), Hillis utilizou a abordagem por co-evolução competitiva. Além da população de redes de ordenação, uma segunda população contendo casos de testes (entradas) foi incluída no processo [24]. As duas populações funcionam ao estilo competidor-competidor, cada qual evoluindo para superar a outra. Os membros das duas populações são confrontados, ao modo apresentado pela Figura 4.9, em que uma rede é emparelhada com um elemento da população de testes. Enquanto as redes de ordenação são avaliadas positivamente por quão bem ordenam as entradas (indivíduos da segunda população), os casos de testes são avaliados de acordo com a habilidade de "reprovar" as redes de ordenação. Os melhores indivíduos da população de redes tendem a reproduzir-se com mais freqüência, inserindo na população boas redes ordenadoras, induzindo concomitantemente a segunda população a desenvolver testes mais complicados. Dito de outro modo, os casos de testes mais difíceis tendem a se espalhar e forçar as redes de ordenação a adquirirem atributos para também ordenarem as entradas mais árduas.

Usando o modelo competitivo, Hillis obteve uma rede de ordenação com 61 comparações/trocas, com apenas uma ligação a mais do que a solução "ótima" de Green, porém expressivamente superior ao resultado obtido pela abordagem tradicional. Não obstante,

[22] Basicamente, a mutação foi definida como a troca de ligações, enquanto a recombinação comutava parte de uma rede de ordenação com parte de outra.

[23] Uma rede de ordenação capaz de ordenar corretamente todas as combinações de 0 e 1 conseqüentemente ordena qualquer outra seqüência. Portanto, uma rede com n entradas pode ser avaliada com 2^n testes.

[24] Cada indivíduo da população de testes traz entre 10 e 20 conjuntos de entrada.

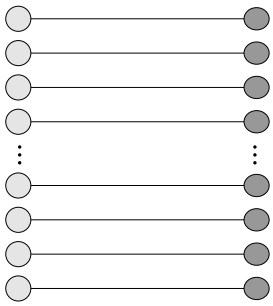

Figura 4.9: Esquema de confronto tipo parelho.

relatou ainda a aceleração considerável do processo de evolução.

Observa-se no método por emparelhamento a economia de avaliações, notoriamente quando comparado com o modelo de competição de Axelrod. Supondo n como o número de indivíduos das duas populações ($n/2 + n/2$), são realizados $n/2$ encontros por geração.

4.2.3 Aptidão via Torneio

Angeline e Pollack [4] usam um modelo co-evolutivo de única população onde a competição é promovida através de torneios, atribuindo-se assim aptidões proporcionais ao *ranking* dos indivíduos. Esta espécie de "olimpíadas" é realizada a cada geração. Inicialmente, todos os indivíduos estão aptos a competir, sendo escolhidos aleatoriamente, em torneios de dois membros por vez. Os vencedores seguem adiante, passando para a próxima fase da competição, enquanto os perdedores cessam suas participações e são classificados igualmente pelo nível que alcançaram, isto é, indivíduos com a mesma colocação terão probabilidades iguais de seleção para recombinação. O processo de confrontos se repete, restritos, porém, aos vencedores da etapa anterior. Finalmente, a competição é concluída quando não há mais desafiantes, restando apenas o vencedor geral —que naturalmente terá a melhor aptidão e conseqüente maior probabilidade de reprodução. A Figura 4.10 ilustra este procedimento, cada seqüência horizontal representando um estágio e as setas indicando os vencedores.

O custo de avaliação por geração para uma população de tamanho n é dado por:

$$\sum_{i=1}^{\lceil \log_2(n) \rceil} \left\lceil \frac{n}{2^i} \right\rceil = n - 1$$

que ainda é um valor superior ao do modelo de Hillis, no entanto, linear e consideravel-

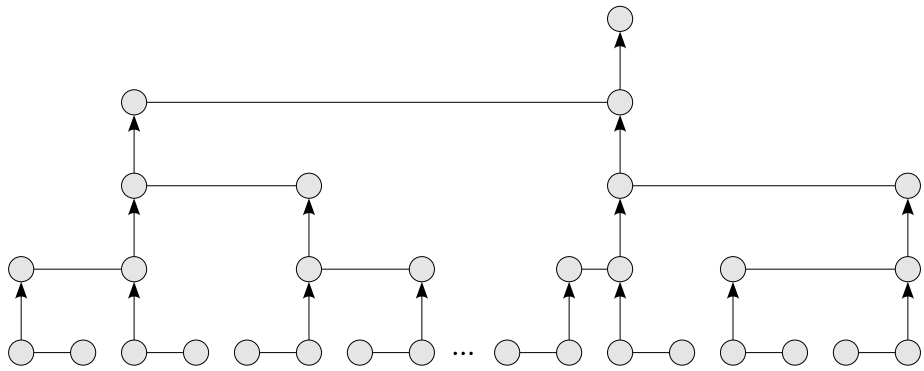

Figura 4.10: Esquema do torneio entre membros da população.

mente inferior ao do método de Axelrod.

Os autores aplicaram este modelo de co-evolução competitiva no clássico e conhecido jogo-da-velha, usando uma variação co-evolutiva da programação genética. Comparou-se ainda com os resultados obtidos pela implementação típica da programação genética.

O experimento contou com três estratégias referenciais de jogadas, a saber: *aleatória*, escolhe ao acaso qualquer posição válida no tabuleiro; *quase ótima*, permite a vitória do oponente somente se este conseguir alcançar um estado em que haja duas posições que garantiriam a vitória, ou seja, ganharia na próxima rodada; *ótima* não admite que o oponente vença, mas pode ceder o empate.

Conhecido os *experts* para o problema em questão, pôde-se praticar o experimento não co-evolucionário, que foi realizado baseado nas três estratégias de referência. Cada uma delas fornece uma diferente função de aptidão —onde os indivíduos são avaliados pela desenvoltura nos jogos—, implicando em três sub-experimentos sobre a evolução canônica. Evoluiu-se, assim, três estratégias, relativas aos distintos "tutores".

No tocante à versão co-evolucionária competitiva, os autores executaram como descrito no início desta seção, delegando ao próprio processo evolutivo o ajuste do referencial.

Quanto postos para competir [25] contra as estratégias *aleatória*, *quase ótima* e *ótima*, os candidatos oriundos do processo não co-evolucionário saíram-se razoavelmente contra as duas primeiras (salvo o confronto entre o indivíduo evoluído pela estratégia *aleatória*, que foi derrotado em todos os jogos contra o *quase ótimo*), mas foram sumariamente

[25] Foram realizadas 2000 competições para cada estratégia evoluída.

vencidos contra a estratégia *ótima*, não conseguindo sequer empatar.

Diferentemente, o método co-evolutivo mostrou-se capaz de criar estratégias mais refinadas, desenvolvendo-se bem contra as referências *aleatória* e *quase ótima*, empatando ainda em cerca de 25% das jogadas contra a estratégia *ótima*. Apesar de ter sido derrotada em 75% dos casos [26], os resultados foram bem mais convincentes do que os apresentados do método tradicional, além disso, o tempo de cômputo foi significativamente menor.

4.2.4 Co-evolução de Redes Neurais Classificadoras

Dentre as produções expostas neste capítulo, esta é sem dúvida a maior contribuinte para a confecção da presente dissertação.

Paredis, no trabalho intitulado *Steps Towards Co-Evolutionary Classification Neural Networks* [43], desenvolve um sistema híbrido e aplica-o no domínio de classificação (supervisionada). O autor une as redes neurais aos algoritmos genéticos co-evolucionários tipo *steady-state* [27], focando-se na soma das características eminentes de cada técnica; os AG's como bom otimizadores e as RN's como habilidosas na tarefa de associação de padrões. Desde modo, o algoritmo genético ajusta os pesos das conexões das redes neurais, enquanto estas se encarregam de classificar as entradas. Paredis também elaborou um tipo de aptidão "memória", que recorda —e considera no cálculo geral— o registro de avaliações passadas de um determinado indivíduo. Este conceito será elucidado no decorrer desta seção.

4.2.4.1 Modelo de Co-evolução

Inspirado na abordagem desenvolvida por Hillis [27], Paredis serviu-se de duas populações, a primeira composta por redes neurais e a segunda de padrões de entrada, em que cada indivíduo constitui uma amostra (ponto). Porém, de fato, esta última não envolve uma real população, no sentido de que não existe modificação no material genético de seus indivíduos, isto é, não há recombinação ou mutação. A explicação é trivial, dados em classificação (como os deste trabalho) são pontos fixos e imutáveis, perdem o sentido se modificados.

[26] Os autores acreditam que a linguagem genérica usada para a descrição das estratégias tenha contribuído para uma maior dificuldade de aprendizado, e portanto, baixo índice de vitórias.

[27] Vide Capítulo 2.

Apesar de não haver alteração genética na população de amostras, esta provê uma pressão contrária (co-evolutiva) à população de redes neurais, à medida que os pontos mais dificilmente classificados tendem a ser mais ofertados para disputar com as redes. Os confrontos dão-se emparelhando dois indivíduos por vez, um representante das redes classificadoras e outro dos dados de entrada. Análogo ao método criado por Hillis, uma rede neural (encapsulada como um indivíduo) recebe uma nota proporcional à sua capacidade de classificação, ao passo que os padrões de entrada são premiados quando induzem à classificação incorreta. A tensão surge quando as melhores redes neurais são mais freqüentemente escolhidas para competir contra, justamente, as mais difíceis amostras. Basicamente, quanto mais um indivíduo se destaca como melhor adaptado, mais terá que provar seu valor, do contrário é derrubado.

A competição redes neurais *versus* padrões de entrada pode ser elucidada graficamente, como disposto na Figura 4.11. Neste esquema, os círculos à esquerda representam a população de redes classificadoras, enquanto os da direita as de amostras de dados. Ambas estão ordenadas, com os indivíduos mais adaptados no topo. As ligações representam os encontros entre os indivíduos da população de classificadores e os membros da população de pontos de entrada. Sutilmente, nota-se que os indivíduos mais ao topo tendem a

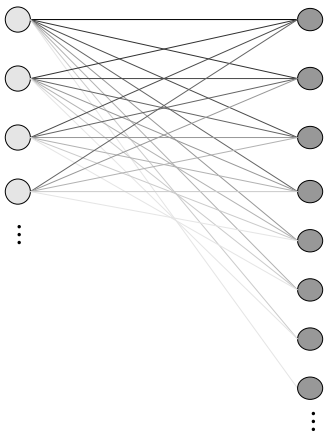

Figura 4.11: Esquema gráfico da competição entre redes neurais e amostras.

passar por maior provação (mais encontros), enquanto os menos adaptados são pouco convocados, devido à condição "comprovada" de baixa qualidade, portanto irrelevante para a evolução. Este processo é interessante e demonstra um ponto importante: a co-evolução direciona a busca para os testes mais difíceis, os exemplos não resolvidos, poupando-se assim recursos computacionais que certamente seriam desperdiçados na tentativa repetida

de resolver casos fáceis.

4.2.4.2 Aptidão Cumulativa

Em vez da tradicional aptidão atribuída ao indivíduo após sua avaliação, Paredis desenvolveu um sistema cumulativo, originado no conceito de "energia", fruto de um trabalho prévio bem sucedido, de própria autoria [42]. Neste modelo, uma rede neural representando um animal mapeia um ambiente simulado em ações motoras, nos mesmos moldes do mapeamento entre uma amostra e sua respectiva classe. Este animal está em constante interação com o ambiente, "acumulando" energia de acordo com sua capacidade de, por exemplo, alimentar-se. Por outro lado, o animal pode sofrer baixas em sua energia, seja por algum ferimento ou incapacidade de angariar recursos. Um detalhe interessante é que, assim como na computação evolucionária, o animal tem sua probabilidade de reprodução proporcional à sua quantidade de energia; isto indica uma animadora similaridade com o modelo evolucionário e, portanto, passível de ser adaptado.

No contexto dos algoritmos genéticos, a aptidão cumulativa funciona da seguinte maneira. Inicialmente, um número finito, também conhecido como *ciclo*, determinando a "memória" cumulativa é definido, possibilitando que a aptidão do indivíduo concentre-se nos resultados dos encontros mais recentes [28]. Não existe uma forma determinística para se definir o valor ótimo do ciclo (que pode depender do variáveis outras do domínio em questão), entretanto, Paredis, apesar de não revelar o meio, parece ter encontrado um "número mágico" [29], um ciclo de tamanho 20. Dessa forma, a aptidão dos indivíduos, isto é, a grandeza usada na seleção natural, é calculada pela soma dos valores acumulados nos últimos 20 encontros.

Em termos biológicos, a aptidão cumulativa, em comparação com a aptidão única e invariável, promove um paralelo mais fiel, quando permite ao objeto em evolução uma avaliação freqüente e dinâmica. É o que se nota na natureza, onde organismos estão sempre postos à prova pelo ambiente, e suas chances de reprodução e sobrevivência estão diretamente ligadas ao histórico de situações pelas quais o organismo em questão enfren-

[28] Poder-se-ia também definir uma função (linear ou exponencial) que decaísse ao longo do tempo, tornando-se assim os resultados de confrontos antigos com menor peso na soma total da aptidão do indivíduo.

[29] Na gíria da programação computacional, o *número mágico* é uma constante especial (numérica) designada para algum propósito. O seu uso geralmente indica má estilo de programação, pois torna o código de difícil leitura, entendimento e manutenção.

tou —que obviamente é também um reflexo da condição genética.

4.2.4.3 Pseudo Algoritmo

O cerne do processo co-evolucionário de Paredis é mostrado no Algoritmo 4.1, onde se observa uma iteração. Assume-se que ambas as populações estão inicializadas e com os indivíduos avaliados. Na primeira avaliação, cada membro da população de classificadores é avaliado contra 20 (valor do ciclo) exemplos aleatórios do conjunto de treinamento, automaticamente atualizando-se também as aptidões das amostras. Espera-se também que para a operação de seleção, as populações estejam previamente ordenadas, favorecendo-se assim a escolha de indivíduos mais aptos. As funções *acumule vitória em* e *acumule derrota em* inserem o registro de vitória (um) e derrota (zero), respectivamente, no histórico de encontros dos membros em competição. Obviamente, o valor mais antigo é automaticamente descartado, como em uma fila —o primeiro a entrar é o primeiro a sair (*FIFO*). Logo após a geração dos filhos, por não possuírem qualquer indicação prévia de aptidão, são avaliados confrontando-se (*ciclo* vezes) com amostras escolhidas ao acaso.

Algoritmo 4.1 Pseudo-algoritmo co-evolucionário com aptidão cumulativa.

```
enquanto ciclo estiver incompleto faça
    competidor A ← selecione rede neural;
    competidor B ← selecione amostra;
    promova competição entre A e B;
    se A classifica B então
        acumule vitória em A;
        acumule derrota em B;
    senão
        acumule derrota em A;
        acumule vitória em B;
    fim se
fim enquanto
P1 ← selecione rede neural para acasalamento;
P2 ← selecione rede neural para acasalamento;
cruze P1 e P2 e gere F1 e F2;
aplique operadores genéticos em F1 e F2;
avalie F1 e F2 e insira-os na população de redes;
```

Na reprodução *steady-state*, embora não haja o conceito propriamente dito de geração, entende-se que uma "geração" foi completada quando se insere na população um número de descendentes igual ao tamanho da população. Em outras palavras se a população contém n indivíduos, após serem criados (e inseridos) n filhos, é dito que a geração atual

foi concluída. De posse disto, uma "geração" *steady-state*, co-evolucionária e com aptidão cumulativa, atuando sobre uma população de tamanho *n*, contabiliza $tamanho_{ciclo}.n$ confrontos. Supondo $tamanho_{ciclo}$ como 20, tem-se $20.n$ avaliações. Ainda que configure mais iterações que o trabalho de Hillis ou Angeline e Pollack, é preciso salientar que no escopo do problema de classificação, esta avaliação representa o encontro entre o classificador e apenas uma amostra.

4.2.4.4 Experimentos

Paredis realizou experimentos onde a tarefa é evoluir redes neurais com habilidades para classificar (mapear) pontos pertencentes a quatro classes, espalhados no domínio $[-1,1] \times [-1,1]$, como na Figura 4.12.

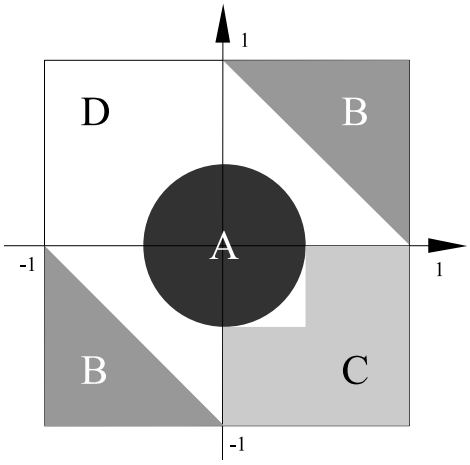

Figura 4.12: Domínio $[-1,1] \times [-1,1]$ contendo quatro classes.

A Figura 4.12 mostra a geometria das quatro classes, sendo a classe **A** consistindo de uma região circular convexa; **B** a união de duas regiões disjuntas; **C** uma região não convexa; e, finalmente, a classe **D** representando a união de duas regiões não convexas.

O conjunto de treinamento é formado por um total de 200 exemplos, distribuídos nas quatro classes descritas. Adotou-se nas RN's a arquitetura de múltiplas camadas, sendo a camada escondida formada por 12 neurônios. A entrada é bi-dimensional, captando as coordenadas dos pontos dos exemplos. Trivialmente, há quatro saídas, cada uma associada a uma classe. Também, o autor avaliou o comportamento dos experimentos em seis diferentes tipos de cruzamento, sendo eles o *crossover* de um ponto, dois pontos, quatro pontos, seis pontos, doze pontos e uniforme.

Os testes foram comparativos e divididos em três categorias, avançando da forma mais simples para a mais elaborada:

1. abordagem tradicional – usou-se apenas o algoritmo genético *steady-state* para otimização dos pesos das conexões das redes neurais. Todos as amostras disponíveis foram usadas na avaliação de cada indivíduo.

2. evolução por aptidão cumulativa – basicamente da forma anterior, exceto pelo fato da avaliação ser dada em um sub-conjunto de exemplos de entrada, usando a noção de ciclo, aptidão cumulativa.

3. co-evolução de redes neurais e amostras – o modo mais elegante, adicionando à abordagem anterior o método de co-evolução competitiva, como descrita nesta seção.

Não surpreendentemente, os resultados obtidos pela abordagem tradicional foram os menos atrativos. Usando uma população de 100 redes neurais, e critério de parada sendo a geração máxima de 50.000 indivíduos durante a evolução, nenhuma rede evoluída [30] ultrapassou 95% de precisão na classificação.

No segundo modelo, aliado à aptidão cumulativa, os resultados foram mais promissores, alcançando, pelo menos no cruzamento de doze pontos, marca superior a 95% de classificações corretas. O curioso é que apesar da melhora na qualidade das soluções não ter sido expressiva, a demanda computacional foi significativamente reduzida, caindo pela metade o tempo de processamento.

Resultados bem melhores foram alcançados com a abordagem mais completa, servindo-se da aptidão cumulativa e, principalmente, da co-evolução competitiva. O tempo de execução foi compatível com o experimento anterior, porém, a precisão de classificação das redes neurais evoluiu substancialmente e, excetuando-se a recombinação de um ponto, todos os testes ultrapassaram a marca classificadora de 95%.

O sucesso deste último experimento sugere o comportamento esperado para um sistema co-evolucionário: criar uma pressão contrária bidirecional que incita cada vez mais o desenvolvimento de artimanhas a fim de superar a população rival. Diferentemente do ideal da natureza, por tratar-se de uma população de amostras, onde não há recombinação

[30] Considerou-se a média entre 5 execuções independentes.

genética, a evolução tem fim —geralmente teórico— quando a população (ou membros desta) em real evolução tem a proeza de superar toda a população de exemplos.

4.2.4.5 Aptidão Cumulativa e Ruídos

No algoritmo genético *steady-state*, um determinado indivíduo mantém-se na população até ordem contrária, ou seja, é exterminado apenas se não estiver suficientemente adaptado. Na implementação usual deste tipo de política geracional, a aptidão é dada "à vista" e permanece assim, imutável, ao longo de toda a vida do indivíduo. O problema surge quando a função de aptidão pode não ser precisa (esta característica é inerentemente dependente do domínio), incidindo em uma avaliação equivocada. Daí, dois possíveis cenários podem ocorrer: o indivíduo ter sua adaptabilidade subestimada, resultando em morte precoce; ou, pior, o indivíduo ser sobrestimado, tornando-se automaticamente imortal (dependendo da intensidade do erro da função de aptidão) e espalhando falsos bons genes por toda população. O impacto provocado por um ruído na aptidão seria algo parecido com o que ocorreria se em uma dada otimização a função objetivo fosse alterada, (re)direcionando a busca para um ponto diferente.

Na área de classificação de padrões ocorrem com certa freqüência ruídos durante a avaliação de indivíduos. O mais comum é a presença de dados deturpados [31] no conjunto de treinamento.

A aptidão cumulativa é uma forma de reduzir substancialmente o efeito de amostras/aptidão ruidosos. Por tratar-se de avaliações parciais e possuir uma memória volátil (reescrita) e limitada, a incidência de um ruído é suavizada. Entretanto, ainda assim dados ruidosos não são ignorados pelo processo co-evolutivo, ao passo que o esforço de classificação tenderá a focar-se nos dados menos classificáveis (onde estão incluídos os *outliers*), atrasando o processo. Não obstante, reduzindo-se a pressão de seleção, faz-se com que a busca seja menos obsessiva e torne-se abrangente, não concentrando-se apenas nos casos estritamente difíceis.

[31] *Outliers.*

4.3 Co-evolução Amostra-Classificador

O trabalho de Paredis —como apresentado neste capítulo—, juntamente com a codificação por gramática (vide Capítulo 3) são as duas maiores referências para o tema desta dissertação. Enquanto a gramática provê um método extensível e preciso para a codificação das árvores classificadoras, o modelo co-evolucionário de Paredis (em conjunto com a aptidão cumulativa) fornece o "motor" para o desenrolar da evolução.

A co-evolução amostra-classificador é um sistema co-evolutivo competitivo (competidor-competidor), baseado no modelo de Paredis e transcrito para a programação genética gramatical. Sinteticamente, a entidade classificadora é substituída por uma árvore de classificação, representada sob a especificação gramatical. Sua tarefa é, naturalmente, a classificação de dados em um número finito de classes.

Nas leituras subseqüentes do termo "co-evolução amostra-classificador" fica implícito a designação do sistema completo, isto é, supõe-se o emprego da programação genética gramatical.

O fluxograma apresentado pela Figura 4.13 exibe a dinâmica do funcionamento da co-evolução amostra-classificador.

4.3.1 Criação da População de Classificadores

Primeiramente, uma gramática é definida, gerando uma linguagem para codificação das árvores classificadoras. Em seguida, a população de classificadores é criada, seguindo a "forma" definida pela gramática. Basicamente, os indivíduos assemelham-se ao exemplo apontado pela Figura 4.14, onde consta o material genético (uma árvore [32]) e sua aptidão cumulativa. Estas duas etapas [33] foram suficientemente exploradas no Capítulo 3, e portanto não serão estendidas aqui.

[32] Na realidade, sob o aspecto de implementação, existem duas árvores, uma contendo a estrutura gramatical e a segunda armazenando de fato o "programa". Mais detalhes em 3.3.3.2, do Capítulo 3.

[33] Vide Capítulo 3, Seções 3.3.2.1 e 3.3.2.2.

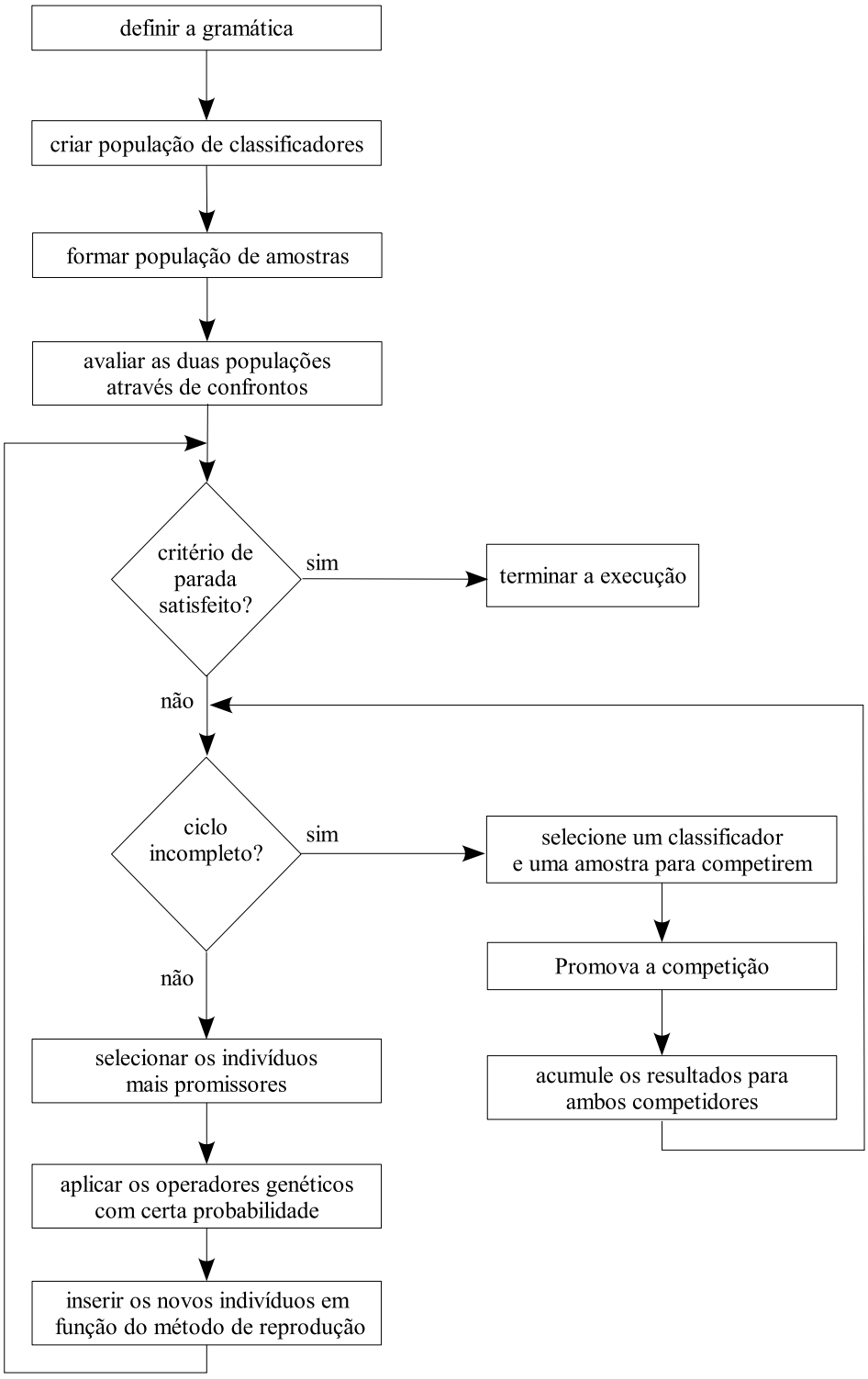

Figura 4.13: Fluxograma do sistema amostra-classificador.

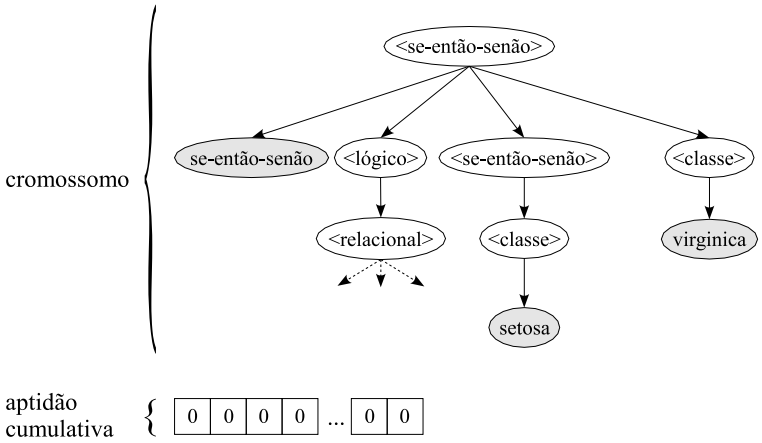

Figura 4.14: Um indivíduo da população de classificadores.

4.3.2 Formação da População de Amostras

O próximo passo é a formação (leitura) da população de amostras/treinamento, que é constituída de todos os padrões disponíveis na base de dados, geralmente da forma atributos-valores e sua respectiva classe. Cada indivíduo da população de amostras constitui uma linha/registro da fonte de dados, limitando-se assim o seu tamanho ao número de linhas do conjunto de treinamento. O cromossomo dos indivíduos pode ser estruturalmente composto por um conjunto de variáveis heterogêneas, comportando assim diferentes tipos de dados, como valores *numéricos* (ex. salário), *lógicos* (verdadeiro, falso), *ordinais* (ex. nível escolar) e *nominais* (como o mês, ano). Graficamente, um indivíduo amostra pode ser visualizado como na Figura 4.15.

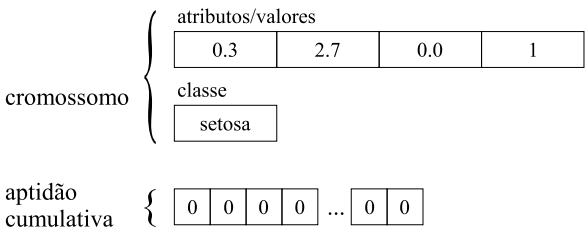

Figura 4.15: Um indivíduo da população de amostras.

4.3.3 Avaliação dos Indivíduos Recém Concebidos

O processo continua com a avaliação dos indivíduos de ambas as populações recém criadas. Este é o momento em que acumulam-se os registros do desempenho nos confrontos. Como não existe conhecimento prévio da qualidade dos indivíduos, as competições ocorrem uniformemente entre os membros da população de árvores classificadoras e os membros da população de amostras, como apontado pelo Algoritmo 4.2.

Algoritmo 4.2 Algoritmo de avaliação inicial.

```
tam_ciclo ← 20;
i ← 0;
enquanto i ≤ tamanho da população de árvores faça
   j ← 0;
   enquanto j ≤ tam_ciclo faça
      competidor A ← i-ésima árvore;
      competidor B ← selecione aleatoriamente uma amostra;
      promova competição entre A e B;
      acumule o resultado em A e B;
      j ← j + 1;
   fim enquanto
   i ← i + 1;
fim enquanto
```

4.3.3.1 Verificação de Falsas Soluções

Imediatamente após as avaliações mútuas, as duas populações são ordenadas e o critério de parada é checado. Havendo um indivíduo da população de árvores com aptidão (cumulativa) satisfatória, isto é, uma candidata a solução, uma etapa adicional de confirmação é efetuada. Nesta confirmação, o candidato submete-se a todas as amostras do conjunto de treinamento, evitando-se que o critério de parada seja satisfeito precipitadamente. Este procedimento de certificação é recomendado pois a aptidão do indivíduo armazena uma pequena parcela dos padrões de entrada, e não necessariamente generaliza para os demais casos de teste. Todavia, deduz-se ainda que, durante a evolução, situações de falso positivo tendem a ser absolutamente raras, pois ao menor sinal de prosperidade de um candidato, a co-evolução trata de confrontá-lo com adversários cada vez mais hostis, diminuindo-se assim as chances de possuir um histórico perfeito de classificações. E caso o consiga, o candidato certamente seria, sob um funcionamento co-evolucionário esperado, capaz de classificar qualquer outra amostra do conjunto de treinamento.

4.3.3.2 Medida de Complexidade

Faz-se útil —e talvez necessária— a consideração de mais uma grandeza para o cálculo final da aptidão de um indivíduo: uma medida de complexidade. É um tópico subjetivo e inerentemente humano, porém, pode-se aproximar seu efeito tendo em conta alguns valores. Dois valores são ponderados neste sistema: o tamanho da entidade classificadora (número de nós da árvore) e o número de atributos do qual ela faz uso. A primeira medida é trivial, e contribui para a simplicidade [34] por meio da penalização de árvores grandes. A moderação do número de atributos tem a função de incentivar soluções que adotem uma menor quantidade de entradas. Em outras palavras, uma árvore classificadora que se restringe a usar somente o atributo *idade* e *salário*, terá melhor aptidão quando comparada a outra que emprege *idade*, *salário* e *condição social*, se ambas classificam com mesma precisão. A consideração desta última medida promove a redução da dimensão de entrada, dispensando atributos pouco significantes. O peso de penalização pode ser ajustado de modo a regular com que intensidade estas medidas serão moderadas, levando-se em conta que muitas vezes acuidade na classificação e economia podem ser objetivos conflitantes.

Finalmente, as propostas mais parcimoniosas são inclinadas a deterem melhores capacidades de generalização, isto é, são mais propensas a serem bem sucedidas na classificação de exemplos não vistos, amostras que não pertencem ao conjunto de treinamento.

4.3.4 Ciclo Co-evolutivo

O ciclo co-evolutivo funciona como o descrito na Seção 4.2.4.3. Limita-se aqui à descrição das particularidades referentes à co-evolução amostra-classificador.

4.3.4.1 Seleção dos Indivíduos

A seleção de indivíduos de ambas populações, visando o confronto, dá-se pela seleção *ranking*, que privilegia os indivíduos mais adaptados, mas assume a "distância" entre membros da mesma população como constante e igual a um (vide 2.2.4).

Para a população de amostras, a seleção opera sobre a função de aptidão f, normali-

[34] Por conseqüência promove a redução de demanda computacional.

zada, dada por:

$$f_{amostra} = \frac{\sum_{i=1}^{n_{ciclo}} H_i}{n_{ciclo}} \tag{4.1}$$

onde n_{ciclo} é o tamanho do ciclo/histórico (geralmente 20), e H_i o i-ésimo valor do histórico de confrontos (0 se não classificou corretamente e 1 caso contrário).

Todavia, para a população de árvores, uma medida ponderada de cálculo de aptidão torna-se necessária, devido ao critério adicional de complexidade. Supondo-se α o peso da complexidade, tem-se a função de aptidão f de uma árvore classificadora como:

$$f_{classf} = (1-\alpha)\frac{\sum_{i=1}^{n_{ciclo}} H_i}{n_{ciclo}} - \frac{\alpha}{2}\left(\frac{tam}{tam_{max}} + \frac{atributos}{atributos_{max}}\right) \tag{4.2}$$

onde tam e tam_{max} são o tamanho da árvore e o tamanho máximo, respectivamente. Similarmente, $atributos$ e $atributos_{max}$ são, nesta ordem, o número de atributos contidos na árvore classificadora e o número de atributos disponíveis na base de dados.

4.3.4.2 Confronto e Atualização da Aptidão Cumulativa

Quando uma árvore classificadora enfrenta uma amostra, os atributos desta última, (isto é, o cromossomo), são substituídos nos nós apropriados da árvore, aqueles que fazem referência aos atributos de entrada. Então, a árvore é executada [35] e a classe predita é averiguada com a classe apontada pela amostra.

Uma classificação bem sucedida implica no crédito de um ponto na aptidão cumulativa do classificador, enquanto o valor zero é inserido na aptidão da amostra. A analogia contrária realiza-se quando a árvore não pôde classificar o exemplo. O Algoritmo 4.3 provê uma visão um pouco mais formal.

É válido ressaltar que a cada alteração no registro de encontros, os indivíduos precisam ter seu *ranking* atualizado, de acordo com sua nova aptidão (Equação 4.2). São preferíveis os métodos de ordenação que caracterizam-se pela facilidade de inserção de valores na posição correta sem a necessidade de reordenação. Do contrário, poderia-se formar um gargalo computacional, devido à alta intensidade de execução deste passo.

[35] Algoritmo 3.5.

Algoritmo 4.3 Algoritmo de avaliação direcionada.

```
tam_ciclo ← 20;
i ← 0;
enquanto i ≤ tam_ciclo faça
    competidor A ← selecione um classificador promissor;
    competidor B ← selecione uma amostra promissora;
    promova competição entre A e B;
    acumule o resultado em A e B;
    i ← i + 1;
fim enquanto
```

4.3.5 Seleção, Aplicação dos Operadores Genéticos e Inserção de Classificadores

Este procedimento restringe-se à população de árvores classificadoras, dada a peculiaridade imutável da população de amostras.

Seleção

Inicialmente, selecionam-se dois indivíduos a fim de reprodução. Novamente, a seleção é guiada pela aptidão como definida na Equação 4.2. No entanto, uma diferente função de seleção é adotada, a seleção torneio de dois oponentes. Em suma, sorteia-se dois indivíduos da população, compara-se suas aptidões e então escolhe-se o indivíduo detentor da melhor aptidão.

Aplicação de Operadores Genéticos

De posse dos pais, a recombinação do material genético dá-se sob uma certa probabilidade —tipicamente próxima de 100%. Se não satisfeito esta condição, os indivíduos são simplesmente clonados.

Com uma chance menor de ocorrência, o operador genético mutação é aplicado nos indivíduos recém gerados.

Os detalhes internos destas duas operações podem ser vistos em 3.3.2.5.

Inserção

Enfim, a prole toma lugar na população. Contudo, por tratar-se de um método de re-

produção tipo *steady-state*, dois indivíduos obrigatoriamente serão eliminados. O critério adotado nesta execução é basicamente o da máxima "sobrevivência do mais adaptado", em outras palavras, após a inserção dos descendentes, os indivíduos de menor aptidão são aniquilados.

Esta etapa sinaliza o final do laço co-evolucionário, retornando o processamento ao ponto de checagem do critério de parada.

4.4 Conclusão

Foi visto neste capítulo o conceito e aplicação da abordagem co-evolutiva. Introduziu-se o tema, passando pela revisão de trabalhos prévios e então foi apresentado o sistema de co-evolução amostra-classificador. Destacaram-se as seguintes vantagens do modelo co-evolutivo por competição:

- aceleração do processo evolucionário – através da focalização nos casos difíceis/não resolvidos.
- manutenção da diversidade populacional – pela co-evolução incitar um constante estado de fluxo, evitando-se assim a estagnação em um ponto fixo.
- ampliação da área de aplicação da computação evolucionária – por permitir que sejam abordados problemas em que não se pode determinar um referencial estático de qualidade de solução.

Não obstante, empregou-se o procedimento de aptidão cumulativa, que visa tanto reduzir o custo computacional como abrandar ruídos que eventualmente possam existir em determinados domínios de problemas.

Enfim, foi discutido em pormenores o sistema de classificação co-evolução amostra-classificador que, dentre outras associações, adota a programação genética gramatical a fim de codificar consistentemente as árvores classificadoras.

O próximo capítulo dedica-se a por na prática a teoria acerca dos conceitos de programação genética, gramática e co-evolução competitiva, através de experimentos de classificação de dados realizados em diferentes domínios.

Capítulo 5

Estudo de Casos

Este capítulo descreve algumas aplicações do sistema aplicado a variados domínios de problemas de classificação. O objetivo foi tentar colher bases de dados cuja amostragem fosse razoavelmente representativa no universo não delineado de possíveis problemas desta natureza.

Seguem então casos tradicionais, como o reconhecimento das classes da planta íris; algumas bases de dados artificiais com propósitos específicos, representadas pela coleção *monks*; outras de caracterização real, como as bases de diagnóstico de câncer de mama, predição partidária em função dos votos, e detecção de tipos de imagens. Os estudos estão detalhados nas respectivas seções.

5.1 Metodologia dos Experimentos

A fim de atingir um grau estatístico mais confiável, para cada problema foram realizadas 10 execuções independentes. Isto é, com sementes distintas e diferentes formações conjuntos treinamento e teste. Este montante foi analisado e extraiu-se alguns componentes da estatística descritiva, tais como *média*, *desvio padrão*, valor *mínimo* e valor *máximo*.

Como medida de desempenho global, foram avaliados os seguintes indicadores: precisão de classificação para o conjunto de treinamento e teste, número de avaliações e complexidade da solução obtida.

Conceitos introduzidos, cabe aqui uma breve explanação mais específica.

- conjunto de treinamento – parte da base de dados responsável pelo aprendizado, tem influência direta no sistema.

- conjunto de teste – composição de dados com a finalidade de avaliar a generalização do "conceito" aprendido durante a fase de treinamento. Não influencia o aprendizado.

- precisão de classificação – indicador da proficiência de um classificador. Índices altos para o conjunto de treinamento, seguidos de baixos valores para o conjunto de teste são um forte indicativo de perda de generalização (*over-training*).

- número de avaliações – quantidade de avaliações até se obter uma solução. Diferentemente do uso de praxe desta medida[1], no corrente trabalho uma avaliação significa um confronto entre uma tupla (registro) da base de dados com um dado classificador (solução candidata).

- complexidade da solução – a complexidade de um classificador passível de compreensão humana não é trivialmente definida. Deste modo, faz-se necessário a adoção de uma medida básica que tende a ser indicativo de complexidade, a saber: o tamanho. Em árvores de decisão o tamanho pode ser calculado pelo número de regras[2] ou simplesmente pelo número de nós. Esta última é a adotada na presente implementação. Ademais, com o intuito de dispensar atributos de entrada irrelevantes, tentou-se minimizar a adoção destes durante o processo evolutivo, fazendo-se uso de pressão contrária à abundância de atributos.

Deve-se mencionar agradecimento ao repositório de *Machine Learning* da Universidade da Califórnia [11], pela disponibilização pública de todas as bases de dados utilizadas nos experimentos deste trabalho.

[1] Habitualmente uma avaliação vem sendo definida como a passagem de todos os dados do conjunto em um determinado classificador.

[2] As regras de uma árvore classificadora são obtidas percorrendo-se todos os caminhos entre o nó raiz e os nós classes (terminais).

5.1.1 Determinação dos Parâmetros

5.1.1.1 Parâmetros Numéricos

Uma característica pouco desejável encontrada em alguns algoritmos é a quantidade excessiva de parâmetros que requerem definição manual. Isto é, aqueles que necessitam ter seus valores ajustados por operadores. Um ajuste mal concebido pode degradar significativamente o desempenho do sistema, resultando em maior tempo de processamento, solução pouco desenvolvida ou, em alguns casos, impossibilidade de obtenção de qualquer solução satisfatória. A determinação destes, por si só representa um problema de otimização. Infelizmente, os algoritmos evolucionários (AE), incluindo algoritmos genéticos e programação genética, compartilham deste empecilho. Não obstante, estes parâmetros geralmente são interdependentes, exigindo o tratamento destes como um todo, sendo, pois, o espaço de busca definido como qualquer combinação dos possíveis valores de cada um deles. Basta que um parâmetro seja contínuo para valer-se de um espaço infinito.

No domínio dos AE algumas técnicas têm sido propostas para lidar com a determinação automática dos parâmetros. Dentre elas, duas merecem destaque.

- **auto-adaptativa –** o algoritmo é capaz de auto adaptar os parâmetros ao longo da evolução, isto é, concomitantemente, seus valores vão sendo ajustados de acordo com o cenário em vigência no processo evolutivo. Tem se obtido sucesso com parâmetros codificados diretamente no cromossomo de cada indivíduo da população [49, 24, 52, 30], adotando-se assim um perfil distribuído de parametrização. A auto-adaptação tem como destaques a adequação dinâmica dos parâmetros e o baixo custo computacional agregado.

- **meta-algoritmo –** esta técnica faz uso de uma camada extra de um algoritmo de otimização trabalhando especificamente no ajuste dos parâmetros [55, 13]. A medida de proficiência é determinada por quão bom o algoritmo AE da camada inferior (AE tradicional) responde a um determinado conjunto candidato de parâmetros. Geralmente os parâmetros são definidos estaticamente, todavia, o nível superior pode evoluir funções capazes de descrever a dinâmica dos parâmetros ao longo da evolução —via programação genética, por exemplo. Seu conceito é facilmente absorvido e sua implementação pode ser dissociada do algoritmo principal. Tem como ponto

negativo o alto custo computacional, porém, sua arquitetura inerentemente paralelizável pode ser explorada.

Como quaisquer destas técnicas foge do escopo do trabalho, tentou-se empiricamente atribuir valores aos parâmetros de entrada, manualmente. No intuito de facilitar a visualização, mostrou-se em forma de tabela os parâmetros relevantes. Na Tabela 5.1 segue um modelo de apresentação.

Parâmetro	Valor
Número de gerações	$\mathbb{N}$
Tamanho da população	$\mathbb{N}$
Profundidade inicial	$\mathbb{N}$
Profundidade de mutação	$\mathbb{N}$
Peso penalização complexidade	$\mathbb{R}$

Tabela 5.1: Modelo de parâmetros.

A descrição destes parâmetros pode ser encontrada nos Capítulos 3 e 4. Alguns dados de entrada permaneceram constantes ao longo de todos os casos estudados, mostrados na Tabela 5.2.

Parâmetro	Valor
Probabilidade de cruzamento	90%
Probabilidade de mutação	5%
Tamanho do ciclo	20
Seleção de competição	*ranking*
Seleção da população de árvores	torneio (2 indivíduos)
Proporção conjunto de treinamento	$2/3$
Proporção conjunto de teste	$1/3$

Tabela 5.2: Parâmetros fixos.

O valor do *ciclo* é definido como proposto por Paredis (vide 4.2.4). Em seu trabalho [43], o autor não menciona o porquê do valor 20, contudo, testes informais realizados na corrente dissertação sugerem que esta quantidade tende a tornar o processo evolucionário mais eficaz.

A *seleção de competição* refere-se ao método empregado para a seleção dos indivíduos classificadores e indivíduo de amostras, para a promoção do confronto. A *seleção da população de árvores* designa o método de seleção para a população principal, isto é, durante a fase de recombinação.

É importante salientar que a divisão $2/3$ e $1/3$ para os conjuntos de treinamento e teste, respectivamente, só prevalece para problemas que não possuem proporções pré-definidas. Quando couber, comentários indicarão os experimentos que fazem uso de proporções próprias.

5.1.1.2 Gramática

Optou-se para estes experimentos por uma definição de gramática que fornecesse uma expressiva gama de possibilidades combinatórias. A gramática exibida na Tabela 5.3 possui estas propriedades e, dentre outras, é capaz de:

- garantir a consistência entre todos os argumentos de funções, ou seja, uma função especifica *a priori* quais tipos de dados pode manipular;
- tratar cada tipo de atributo adequadamente, por exemplo, restringindo as operações sobre os tipos nominais em *igualdade* e *desigualdade* apenas;
- comparação inter-atributos;
- criar constantes numéricas aleatórias (efêmeras);
- permitir o uso de expressões aritméticas em valores numéricos, podendo envolver constantes ou até mesmo atributos.

Provavelmente esse poder de expressão pode ser redundante em alguns problemas mais simples, prova disso é a considerável eficiência de métodos clássicos, como os geradores de árvores de decisão [44], que, apesar de limitarem-se à comparação atributo–valor, conseguem ótimos resultados. Por outro lado, exceto para problemas artificiais, é virtualmente impossível dizer de antemão o grau de complexidade expressiva necessária. Sem dúvida, o melhor desempenho para qualquer classificador é alcançado quando o repertório de ferramentas se aproxima do estritamente necessário para executar a tarefa.

5.1.2 Apresentação dos Resultados

5.1.2.1 Interpretação das Tabelas

Para que seja possível a leitura das tabelas de resultados, será preciso tecer sobre como os dados de saída foram gerados pelo sistema. A cada geração de uma execução em par-

N =	{<se-então-senão>, <lógico>, <numérico>, <nominal>, <ordinal>, <relacional>, <atrib-lógico>, <atrib-numérico>, <atrib-nominal>, <atrib-ordinal>, <classe>, <const-lógica>, <const-numérica>, <const-nominal>, <const-ordinal>}		
Σ =	{se-então-senão, e, ou, não, igual, diferente, maior, maior-igual, menor, menor-igual, soma, subtração, multiplicação, divisão, $efemero_n$, $atrnum_n$, $atrnom_n$, $atrord_n$, $atrlog_n$, $classe_n$, $constnum_n$, $constnom_n$, $constord_n$, $verdadeiro$, $falso$}		
S =	<se-então-senão>		
P =	<se-então-senão>	→	<classe> | se-então-senão <lógico> <se-então-senão> <se-então-senão>;
	<lógico>	→	e <lógico> <lógico> | ou <lógico> <lógico> | não <lógico> | <atrib-lógico> | <const-lógica> | <relacional>;
	<relacional>	→	maior <numérico> <numérico> | maior-igual <numérico> <numérico> | menor <numérico> <numérico> | menor-igual <numérico> <numérico> | igual <numérico> <numérico> | diferente <numérico> <numérico>;
	<relacional>	→	igual <nominal> <nominal> | diferente <nominal> <nominal>;
	<relacional>	→	igual <ordinal> <ordinal> | diferente <ordinal <ordinal> | menor <ordinal> <ordinal> | menor-igual <ordinal> <ordinal> | maior <ordinal> <ordinal> | maior-igual <ordinal> <ordinal>;
	<relacional>	→	igual <lógico> <lógico> | diferente <lógico> <lógico>;
	<numérico>	→	<atrib-numérico> | <const-numérica> | soma <numérico> <numérico> | subtração <numérico> <numérico> | multiplicação <numérico> <numérico> | divisão <numérico> <numérico> | negação <numérico>;
	<nominal>	→	<atrib-nominal> | <const-nominal>;
	<ordinal>	→	<atrib-ordinal> | <const-ordinal>;
	<atrib-numérico>	→	$atrnum_1$ | $atrnum_2$ | ... | $atrnum_n$;
	<const-numérica>	→	$constnum_1$ | $constnum_2$ | ... | $constnum_n$ | $efemero_1$ | $efemero_2$ | ... | $efemero_n$;
	<atrib-nominal>	→	$atrnom_1$ | $atrnom_2$ | ... | $atrnom_n$;
	<const-nominal>	→	$constnom_1$ | $constnom_2$ | ... | $constnom_n$;
	<atrib-ordinal>	→	$atrord_1$ | $atrord_2$ | ... | $atrord_n$;
	<const-ordinal>	→	$constord_1$ | $constord_2$ | ... | $constord_n$;
	<atrib-lógico>	→	$atrlog_1$ | $atrlog_2$ | ... | $atrlog_n$;
	<const-lógica>	→	$verdadeiro$ | $falso$;
	<classe>	→	$classe_1$ | $classe_2$ | ... | $classe_n$;

Tabela 5.3: Gramática para árvore classificadora.

ticular, foram coletados, do melhor indivíduo [3] da população, os seguintes indicadores: índice [4] de classificação nos conjuntos de **treinamento** e **teste**, **tamanho**[5], **número de avaliações** e, finalmente, a média simples do poder de classificação no conjunto de **treinamento** e **teste** (classificação "global"). Por exemplo, tem-se na Tabela 5.4 uma típica listagem da saída de uma execução.

Geração	Treinamento	Teste	Tamanho	Num. Aval.	Trein./Teste
0	0.597	0.583	7	10500	0.590
1	0.742	0.750	13	28980	0.746
2	0.984	0.990	14	47480	0.987
3	1.000	1.000	11	65500	1.000
⋮	⋮	⋮	⋮	⋮	⋮

Tabela 5.4: Listagem da saída de uma execução particular.

Por basear-se nos indivíduos de melhor *aptidão*, não é garantido que as gerações reportem —na listagem— índices monótonos de classificação, isto é, a saída de uma geração posterior pode conter índices menores em comparação com a anterior. Em outras palavras, a última linha da listagem (última geração) não necessariamente indicará os valores do melhor indivíduo de todo o processo de evolução. Diante disso, percebe-se que para resgatar os melhores índices (e o melhor indivíduo), é preciso ordenar a listagem de acordo com o valor de interesse, por exemplo, ordenar pelo *melhor índice de classificação no conjunto de treinamento*, ou ainda, pelo *menor indivíduo* (tamanho da árvore). Portanto, para cada execução, a listagem é ordenada e os valores desejáveis são coletados; por fim, são realizadas médias aritméticas (em cada índice) cobrindo todas execuções.

É fácil perceber que desta coleção de dados pode-se extrair variadas informações, dependendo exclusivamente do foco da análise. Certamente um especialista na área do problema poderia definir seus objetivos e isolar os índices que lhe interessam.

No intuito de aproximar-se da completude, sem no entanto buscar as incontáveis possibilidades teóricas de combinações (o que seria inviável ou no mínimo cansativo), a apresentação foi dividida em três grupos: treinamento, teste e treinamento/teste. Objetivando, respectivamente, melhores classificadores para o conjunto de treinamento, teste

[3] Não necessariamente o melhor classificador, mas aquele detentor da maior aptidão —histórico restrito de confrontos. Ver Capítulo 4.

[4] Os índices foram gerados usando-se todas as amostras de dados de cada conjunto, diferentemente do processo evolucionário onde se utilizava apenas uma "janela" de amostras selecionadas.

[5] O tamanho é dado pelo número de nós da árvore classificadora, sendo portanto um valor sempre superior ao número de regras potencialmente extraídas da árvore.

e a média treinamento/teste [6]. Poder-se-ia pensar em descrever outros grupos, como por exemplo, alguma média ponderada entre complexidade da solução e precisão de classificação no conjunto de teste. Porém, qualquer tentativa de uma abordagem sistemática não teria valor prático e levaria a degradação da compreensão dos resultados.

Em cada linha da tabela está representado um grupo (ou objetivo). Por questões de *lay out* e restrição da largura da tabela, alguns títulos foram abreviados. Naturalmente, cada objetivo (linha) contém todas as demais informações (colunas), por exemplo, ao buscar sobre o desempenho do melhor classificador no conjunto de treinamento, pode-se também conhecer sua precisão no conjunto de teste, seu tamanho, etc. Objetivando uma fácil leitura direcionada aos valores mais relevantes da tabela, os índices de classificação relativos ao conjunto de treinamento, teste e "global", foram destacados em negrito (diagonal da tabela).

Cada valor de entrada na tabela é a média aritmética entre as 10 execuções, seguida pelo respectivo desvio padrão, bem como seu valor mínimo e máximo.

5.1.3 Ambiente Computacional

O sistema foi escrito na linguagem **C++** em conjunto com a expoente biblioteca **GAlib** [54]. O código foi compilado com o software **gcc** e os problemas foram executados sob o sistema operacional **GNU/Linux** em uma máquina **AMD Athlon 1.4GHz** com 512MB de memória RAM.

5.2 Classes das Plantas Íris

Provavelmente este é o problema mais referenciado na literatura de reconhecimento de padrões, assim como em *Knowledge Discovery in Databases* [7].

A base de dados soma 150 exemplos, contendo informações sobre medidas de alguns atributos da planta íris, a saber: comprimento e largura da sépala, e comprimento e largura da pétala. As grandezas são contínuas e estão em centímetros. Estes dados estão igualmente divididos em três classes, são elas: íris *setosa*, *versicolour* e *virgínica*, sendo

[6] Isto significa que em uma execução particular, a listagem foi ordenada para cada um desses objetivos, então, seus melhores valores foram coletados e considerados para o cálculo estatístico.

[7] Descoberta de Conhecimento em Base de Dados.

que a primeira classe é linearmente separável das demais.

5.2.1 Experimentos

Não há um consenso na literatura no tocante ao uso do conjunto de teste para este problema [8]. Em prol de uma abordagem mais ampla, adotou-se os dois perfis de experimentação.

5.2.1.1 Experimento com Base Segmentada

A Tabela 5.5 apresenta o conjunto de parâmetros para os experimentos com a base de dados íris.

Parâmetro	Valor
Número de gerações	500
Tamanho da população	300
Profundidade inicial	15
Profundidade de mutação	15
Peso penalização complexidade	$0,2$

Tabela 5.5: Parâmetros para o problema íris base segmentada.

A Tabela 5.6 sintetiza os índices de desempenho para o sistema proposto. Cada execução consumiu em média 6 minutos.

	Treinamento	Teste	Trei./Teste	Tamanho	N. Aval. ($\times 10^6$)
Trein.	**0,976 ± 0,012**	$0,970 \pm 0,022$	$0,973 \pm 0,007$	$42,1 \pm 36,5$	$2,84 \pm 1,55$
Mín.	$0,960$	$0,940$	$0,965$	$13,0$	$0,364$
Máx.	$0,990$	$1,000$	$0,985$	$140,0$	$4,69$
Teste	$0,961 \pm 0,013$	**0,988 ± 0,010**	$0,975 \pm 0,009$	$42,5 \pm 36,7$	$2,32 \pm 1,59$
Mín.	$0,930$	$0,980$	$0,955$	$13,0$	$0,364$
Máx.	$0,980$	$1,000$	$0,985$	$140,0$	$5,05$
T./T.	$0,973 \pm 0,008$	$0,988 \pm 0,010$	**0,981 ± 0,006**	$47,6 \pm 34,8$	$2,36 \pm 1,45$
Mín.	$0,960$	$0,980$	$0,970$	$13,0$	$0,364$
Máx.	$0,980$	$1,000$	$0,990$	$140,0$	$4,20$

Tabela 5.6: Resultados para a base de dados íris segmentada.

[8] Não que de fato exista uma indecisão sobre a utilização do conjunto de teste. A orientação é pela segmentação da base nos dois conjuntos, porém, pela freqüência de experimentos usando-se inteiramente a base de dados, esta prática foi a adotada no corrente trabalho.

Os resultados indicam uma boa capacidade de generalização, superando inclusive os índices no conjunto de treinamento. A classificação global (treinamento e teste) acompanha o desempenho, refletindo ótima competitividade. Na literatura encontra-se vasto material relativo a testes com a base de dados íris. A Tabela 5.7 mostra um apanhado destes, de cima para baixo tem-se os algoritmos C4.5 [9] [44], BGP [10] [47], GATree [11] [1], CGP/SA [12] [21] e FDT [13] [39]. Os resultados apresentados na tabela são relativos ao conjunto de teste e foram colhidos dos respectivos trabalhos dos autores referenciados.

Algoritmo	**Acurácia**
C4.5	$0,952$
BGP	$0,941$
GATree	$0,938$
CGP/SA	$0,962$
FDT	$0,960$

Tabela 5.7: Desempenho de outros algoritmos sobre a base de dados íris.

Na Figura 5.1 tem-se um exemplo de um classificador para o problema da planta íris. Este conta com um índice médio de classificações corretas de 98,5% [14], sendo que obteve 97% para o conjunto de treinamento e 100% para o conjunto de teste. Uma característica intrigante foi o demérito das constantes numéricas. Mesmo que estas estivessem ao dispor do processo evolutivo, como previsto pela definição da gramática, estas não foram recrutadas. Este desvínculo de constantes tende a deixar a solução mais generalizável, migrando de referenciais fixos (valores constantes) para os variáveis (os valores dos atributos de entrada). Por exemplo, neste caso, mesmo que todos os dados da base fossem multiplicados por um escalar, a árvore da Figura 5.1 obteria o mesmo êxito na classificação.

Uma solução mais enxuta, com 13 nós, também foi encontrada (Figura 5.2). Nesta, os índices foram de 96% para o conjunto de treinamento e 98% para o conjunto de teste.

[9] Construção de árvores de decisão por divisão-e-conquista utilizando informação de ganho (*gain ratio*).
[10] Evolução de árvores de decisão por programação genética.
[11] Árvores de classificação via algoritmos genéticos.
[12] Programação genética e *simulated annealing* na evolução de árvores de decisão.
[13] Geração de regras *fuzzy*.
[14] A média entre os conjuntos de treinamento e teste.

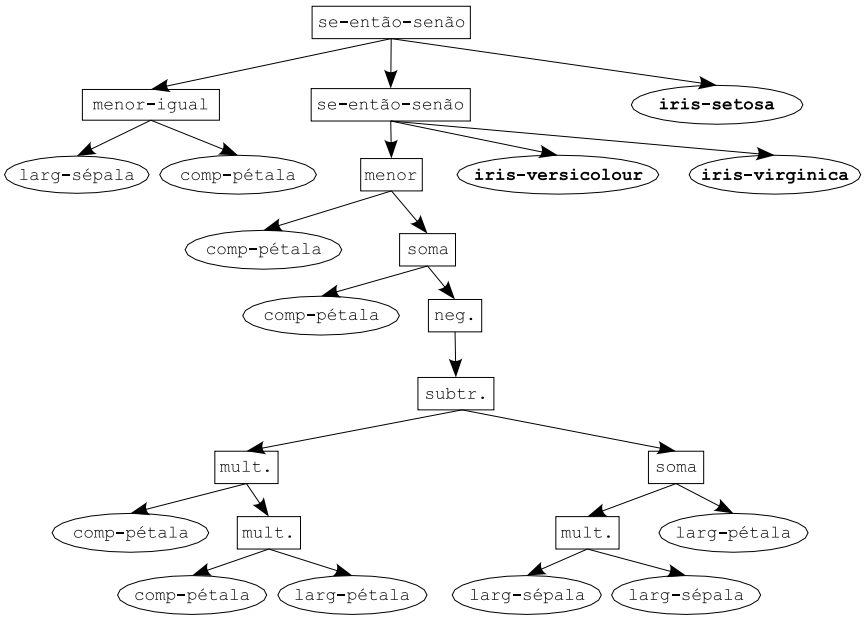

Figura 5.1: Árvore de classificação da planta íris.

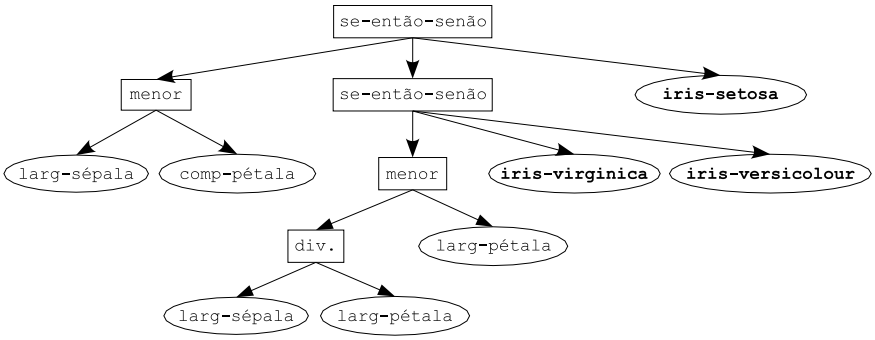

Figura 5.2: Árvore de classificação da planta íris.

5.2.1.2 Experimento com a base completa

Objetivando um maior refinamento e maior precisão no conjunto de treinamento, sem no entanto atentar-se para o "super treinamento", dobrou-se o número máximo de gerações, de 500 para 1000 iterações. Entretanto, a fim de evitar um crescimento improdutivo, aumentou-se o peso de penalização por complexidade para 0,25. Os demais parâmetros permanecem os mesmos, como mostrados na Tabela 5.8.

Os resultados mostrados na Tabela 5.9 refletem muito bem o poder de adaptação do algoritmo proposto. Demonstrou ótima estabilidade em termos de acurácia, com pouca variabilidade nos resultados. Por outro lado, no quesito complexidade (tamanho), ob-

Parâmetro	Valor
Número de gerações	1000
Tamanho da população	300
Profundidade inicial	15
Profundidade de mutação	15
Peso penalização complexidade	$0,25$

Tabela 5.8: Parâmetros para o problema íris base completa.

teve uma discrepância acentuada, variando de 30 nós na menor árvore para 222 na maior. Observa-se ainda, em relação ao problema com a base segmentada, o acréscimo do tamanho médio, apesar do rigor superior em relação à penalização. Isto induz a uma conclusão natural: à medida que o número de refinamentos cresce (gerações), mais inclinada a evolução tende à especialização. Tentativas foram feitas com pesos de complexidade maiores, infelizmente levando o sistema a estagnação prematura. Acredita-se, portanto, em um melhor desempenho com uso de ponderação dinâmica, variando em função da tendência evolutiva. Como era de se esperar, o tempo médio aproximado para cada execução aumentou, passando para 18 minutos.

	Treinamento	**Tamanho**	**N. Aval.** ($\times 10^6$)
	$0,997 \pm 0,005$	$86,2 \pm 54,6$	$7,12 \pm 2,77$
Mín.	$0,987$	$30,0$	$3,44$
Máx.	$1,000$	$222,0$	$10,6$

Tabela 5.9: Resultados para a base de dados íris completa.

No trabalho de Duch *et al.* [19], os autores colecionam experimentos de sete algoritmos distintos atuando sobre a base de dados íris completa. Os resultados de classificação variam entre 95,7% e 100%. Com virtualmente 100% de precisão média, fica demonstrado, portanto, a excelente atuação do sistema deste trabalho.

A Figura 5.3 apresenta um exemplo de árvore solução isenta de erros de classificação, isto é, foi flexível o suficiente para absorver todo o conceito da base de dados. Cabe observar a ausência do primeiro atributo de entrada, o `comprimento da sépala`. O processo evolutivo julgou-o desnecessário e evitou-se assim a redundância, a favor de uma alternativa mais parcimoniosa.

Ainda é possível atentar-se ao fato de como facilmente a primeira classe é separável das demais. Esta propriedade está caracterizada nos três exemplos apresentados. Na Fi-

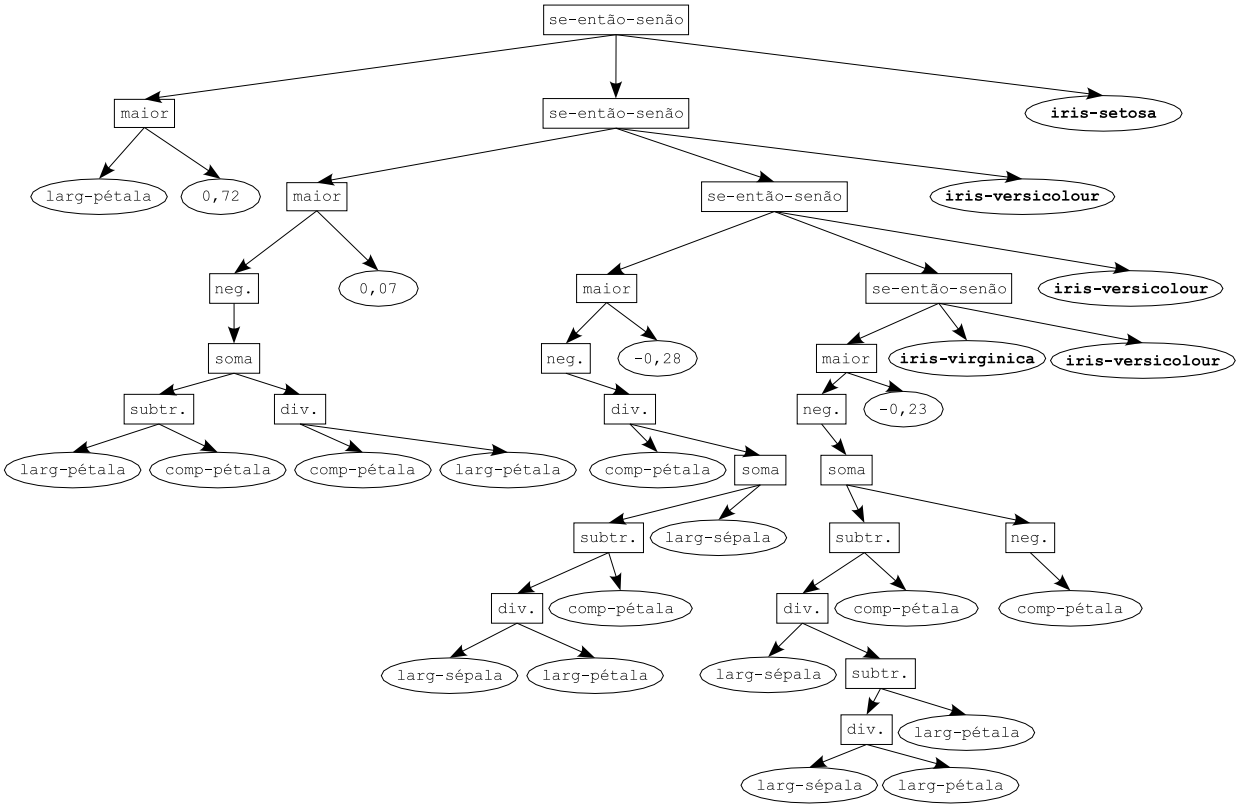

Figura 5.3: Árvore de classificação da planta íris sobre o conjunto de treinamento.

gura 5.1 esta separação é alcançada pela comparação `largura sépala` $\leq$ `comprimento pétala`. A Figura 5.2 provê algo bastante semelhante. Já na Figura 5.3 a largura da pétala é comparada com uma constante numérica (`largura pétala` $>$ `0,72`).

5.3 Diagnóstico de Câncer de Mama

A área médica vem sendo alvo de grandes pesquisas computacionais, muitas destas destinadas à automação do diagnóstico de doenças. Dentre estas doenças, muito tem sido feito para o diagnóstico do câncer, especificamente o câncer de mama.

Nódulos presentes na massa mamária podem ser indicativos de câncer, mas não são suficientes para determinar a gravidade do tumor, se este for confirmado. A análise por FNA (*Fine-Needle Aspiration*) —procedimento de extração de fluidos realizado diretamente sobre o tecido canceroso [36]— é capaz de fornecer medidas sobre a "forma" do tumor instaurado no paciente. Estes dados são posteriormente examinados e poderão sentenciar a gravidade do câncer. A base de dados de câncer de mama, criada pelo Dr.

William H. Wolberg (*University of Wisconsin Hospitals*), traz amostras de casos clínicos coletadas no período compreendido entre os anos de 1989 e 1991. O objetivo é o diagnóstico do grau do tumor, distribuídos em duas classes: benigno ou maligno.

A base possui 699 registros, sendo que cada atributo representa alguma medição (via FNA) de uma particularidade da célula em questão; todos escalares contínuos, compreendidos entre 0 e 1. Os atributos são, nesta ordem, *clump thickness*, *uniformity of cell size*, *uniformity of cell shape*, *marginal adhesion*, *single epithelial cell size*, *bare nuclei*, *bland chromatin*, *normal nucleoli* e *mitosis*.

Naturalmente, uma característica desejável e necessária nesses sistemas autônomos é a capacidade de boa generalização, pois, neste domínio, o problema real encontra-se nos dados que estão por vir, restando àqueles da base de dados a função exclusiva de aprendizado.

5.3.1 Experimentos

A Tabela 5.10 traz o conjunto de parâmetros utilizados neste experimento.

Parâmetro	Valor
Número de gerações	500
Tamanho da população	500
Profundidade inicial	15
Profundidade de mutação	15
Peso penalização complexidade	$0,05$

Tabela 5.10: Parâmetros câncer de mama.

Nota-se que o peso de penalização por complexidade está relaxado, espera-se com isto um melhor índice de classificação para o conjunto de treinamento em detrimento ao tamanho da solução e poder de generalização. Tem-se na Tabela 5.11 o sumário das execuções deste experimento. Em cada execução gastou-se por volta de 15 minutos de processamento.

De fato as previsões foram confirmadas, principalmente no quesito complexidade, onde claramente houve um crescimento exagerado. Entretanto, em relação ao desempenho no conjunto de teste, os resultados foram suficientemente satisfatórios. Muito provavelmente as soluções, em sua grande maioria, adquiriram partes redundantes, como por

	Treinamento	**Teste**	**Trei./Teste**	**Tamanho**	**N. Aval.** ($\times 10^6$)
Trein.	**0,981 ± 0,017**	$0,935 \pm 0,016$	$0,958 \pm 0,015$	$124,1 \pm 59,8$	$6,58 \pm 2,64$
Mín.	0,952	0,911	0,939	21,0	1,41
Máx.	1,000	0,957	0,979	198,0	9,03
Teste	$0,956 \pm 0,016$	**0,960 ± 0,014**	$0,958 \pm 0,012$	$84,4 \pm 73,1$	$3,29 \pm 2,88$
Mín.	0,933	0,939	0,936	13,0	0,158
Máx.	0,993	0,979	0,977	234,0	8,26
T./T.	$0,974 \pm 0,018$	$0,951 \pm 0,011$	**0,962 ± 0,013**	$123,2 \pm 79,3$	$5,28 \pm 2,76$
Mín.	0,947	0,925	0,941	13,0	0,194
Máx.	0,998	0,961	0,979	218,0	8,90

Tabela 5.11: Resultados câncer de mama com tamanho relaxado.

exemplo sub-árvores que nunca serão avaliadas. Em contrapartida, se o índice de classificação no conjunto de teste fosse relativamente baixo, seria sensato concluir a presença de especialização no conjunto de aprendizado.

No intuito de podar os ramos inúteis e valorizar a generalização, o parâmetro peso de penalização por complexidade foi elevado para $0,15$ e um novo experimento foi realizado, como mostra a Tabela 5.12. A pressão sobre o tamanho da solução, juntamente com o menor número de avaliações para se obter soluções satisfatórias, fez com que o tempo de processamento caísse drasticamente, ficando estabelecido em 4 minutos.

	Treinamento	**Teste**	**Trei./Teste**	**Tamanho**	**N. Aval.** ($\times 10^6$)
Trein.	**0,961 ± 0,006**	$0,958 \pm 0,012$	$0,960 \pm 0,006$	$20,6 \pm 16,9$	$1,90 \pm 2,75$
Mín.	0,953	0,940	0,947	8,0	0,047
Máx.	0,970	0,974	0,967	58,0	7,70
Teste	$0,953 \pm 0,00$	**0,967 ± 0,011**	$0,960 \pm 0,008$	$19,6 \pm 14,3$	$1,85 \pm 2,58$
Mín.	0,942	0,944	0,943	8,0	0,283
Máx.	0,970	0,983	0,972	50,0	8,73
T./T.	$0,959 \pm 0,007$	$0,965 \pm 0,012$	**0,962 ± 0,007**	$20,0 \pm 12,6$	$1,69 \pm 1,76$
Mín.	0,948	0,940	0,947	8,0	0,10
Máx.	0,970	0,979	0,972	50,0	5,3

Tabela 5.12: Resultados câncer de mama com maior peso complexidade.

Em comparação com o experimento prévio, percebe-se um declínio na acurácia relativa ao conjunto de treinamento. Em compensação houve uma sensível melhora na avaliação pelo conjunto de teste e uma queda vertiginosa tanto no tamanho médio da solução como no número de avaliações.

Em [19], é realizado, para este problema, testes com variados algoritmos, reportando

percentagens de classificação entre 34,5% e 97,1%. Quinlan [44], utilizando o algoritmo C4.5 consegue o índice de 94,7%. Em todos os casos, os resultados foram obtidos via validação cruzada.

Com o propósito ilustrativo, a Figura 5.4 mostra uma árvore classificadora com duas propriedades interessantes: é enxuta e utiliza-se da operação soma (entre valores de atributos) para, então, comparar com uma constante numérica. Este tipo de arranjo é impossível via algoritmos canônicos de árvores de decisão (como o C4.5), pois estes se limitam à comparação atributo-valor. Na ocasião, esta solução classificou corretamente 97,4% do conjunto de teste e 95,6% da base completa. Um ótimo resultado dado a simplicidade desta.

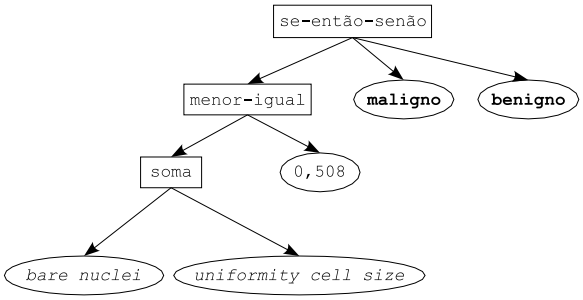

Figura 5.4: Árvore de diagnóstico do câncer de mama.

Já a Figura 5.5, de maior complexidade, obteve melhores resultados —97,8% no conjunto de teste e 97% de classificação global— em detrimento de uma solução mais parcimoniosa. Nota-se, portanto, o largo emprego de operações aritméticas, como soma, subtração, multiplicação e negação.

5.4 Predição Partidária dos Congressistas

É curiosa a análise do perfil de congressistas de acordo com a filiação partidária. Percebe-se, naturalmente, que membros de uma certa aliança tendem a ter comportamentos semelhantes, traduzindo assim a ideologia do partido. O problema da predição partidária busca deduzir, a partir de uma série de posições políticas, a qual segmento um determinado congressista pertence. O conjunto de votos de um político em pontos estratégicos da administração (por exemplo, gasto em educação, repressão do crime, entre outros) define sua posição, que por sua vez é um bom indicativo de sua filiação partidária.

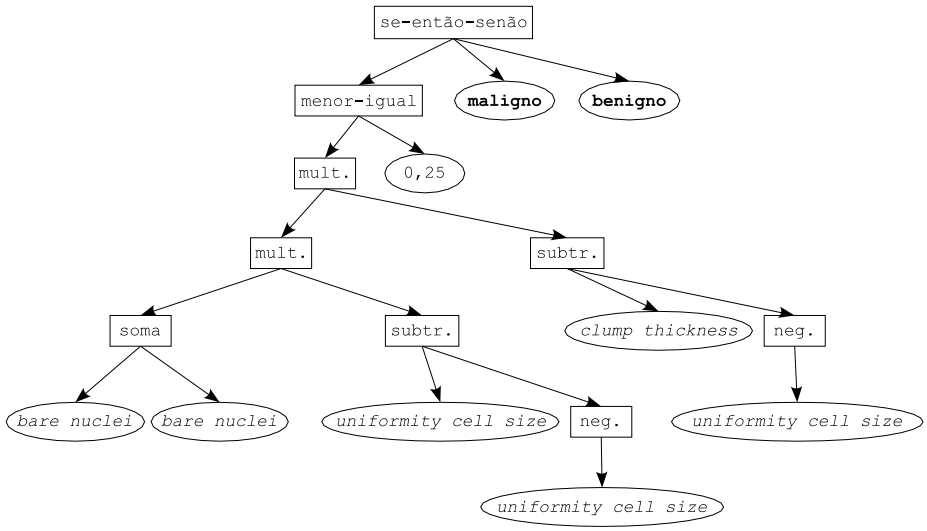

Figura 5.5: Árvore de diagnóstico do câncer de mama.

A base de dados de registros de votação dos congressistas norte-americanos, oriunda do relatório trimestral do congresso (CQA [15]) do ano de 1984, traz a coleção de votos dos representantes em 16 questões chaves da gestão pública. A cada uma destas questões três variantes [16] de votos são permitidas, a saber: voto **a favor**, **contra** e **não definido**. A possibilidade partidária se restringe a duas: **democratas** e **republicanos**. A base conta com 435 registros de votação, cada qual correspondendo a participação de um político do congresso. Os pontos da administração passíveis de votos são mostrados a seguir.

1. *handicapped infants* – HI
2. *water project cost sharing* – WPCS
3. *adoption of the budget resolution* – ABR
4. *physician fee freeze* – PFF
5. *el salvador aid* – ESA
6. *religious groups in schools* – RGS
7. *anti satellite test ban* – ASTB
8. *aid to nicaraguan contras* – ANC
9. *mx missile* – MM
10. *immigration* – I

[15] *Congressional Quarterly Almanac, 98th Congress, 2nd session 1984, Volume XL: Congressional Quarterly Inc. Washington, D.C., 1985.*

[16] As três possibilidades de votos foram simplificadas do conjunto de nove, originalmente previstas pelo CQA.

11. *synfuels corporation cutback* – SCC
12. *education spending* – ES
13. *superfund right to sue* – SRS
14. *crime* – C
15. *duty free exports* – DFE
16. *export administration act south africa* – EAASA

5.4.1 Experimentos

Para a primeira bateria de execuções utilizou-se os parâmetros listados na Tabela 5.13.

Parâmetro	Valor
Número de gerações	1000
Tamanho da população	300
Profundidade inicial	10
Profundidade de mutação	15
Peso penalização complexidade	$0,15$

Tabela 5.13: Parâmetros iniciais para a predição partidária.

Observou-se para este problema que a criação inicial de árvores menos complexas cooperou sensivelmente para um melhor desempenho do processo evolutivo como um todo. Optou-se, portanto, em fixar a profundidade máxima inicial em um patamar inferior à profundidade de mutação.

Os resultados obtidos podem ser vistos na Tabela 5.14. Para cada execução foram necessários em média cerca de 17 minutos de tempo de processamento.

O algoritmo proposto gerou ótimos resultados, com índices consideravelmente superiores a alguns trabalhos prévios encontrados na literatura. A Tabela 5.15 sumariza alguns destes resultados.

No anseio de soluções mais compactas, decidiu-se por aumentar o peso de penalização por complexidade, passando, então, de $0,15$ para $0,2$. Os demais parâmetros permanecem os mesmos.

A Tabela 5.16 apresenta os resultados com esta nova abordagem. Esta alteração também refletiu positivamente no tempo médio de execução, reduzindo-o para 8 minutos.

	Treinamento	**Teste**	**Trei./Teste**	**Tamanho**	**N. Aval.** ($\times 10^6$)
Trein.	**0,982 ± 0,015**	$0,943 \pm 0,023$	$0,962 \pm 0,006$	$105,3 \pm 61,6$	$6,85 \pm 4,47$
Mín.	0,962	0,910	0,953	20,0	0,268
Máx.	1,000	0,979	0,971	199,0	10,6
Teste	$0,956 \pm 0,015$	**0,976 ± 0,011**	$0,966 \pm 0,009$	$59,6 \pm 28,4$	$5,12 \pm 2,24$
Mín.	0,924	0,959	0,945	16,0	0,201
Máx.	0,972	0,993	0,978	103,0	8,16
T./T.	$0,972 \pm 0,010$	$0,970 \pm 0,012$	**0,971 ± 0,003**	$92,8 \pm 51,2$	$6,77 \pm 2,71$
Mín.	0,962	0,945	0,969	20,0	0,268
Máx.	0,993	0,986	0,978	186,0	9,09

Tabela 5.14: Resultados da execução inicial para a predição partidária.

Algoritmo	**Acurácia**
C4.5	0,961
GATree	0,956
CGP/SA	0,976

Tabela 5.15: Desempenho de outros algoritmos sobre a predição partidária.

Com um pequeno desgaste no índice de classificação foi possível encontrar soluções em média três vezes menores que as previamente encontradas. A ênfase e escolha da melhor abordagem estaria sujeita ao desejo do especialista, optando pela melhor compreensibilidade ou uma acurácia sensivelmente mais precisa.

Interessante notar que apenas um atributo (*physician fee freeze - PFF*) da base de dados deste problema tem uma ampla relevância, sendo, portanto, capaz de discernir corretamente uma grande massa de dados. Foi o que a evolução descobriu, concebendo a minimalista árvore da Figura 5.6, classificando 416 registros no total de 435 (95,6%).

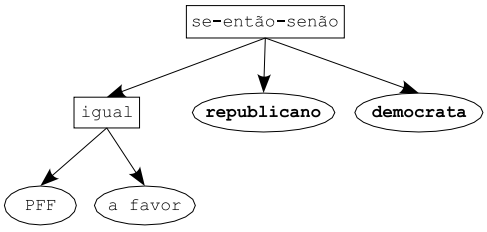

Figura 5.6: Árvore classificadora predição partidária.

A árvore da Figura 5.7 basicamente usa o mesmo princípio. Utiliza-se do atributo discriminador para atingir uma grande separação nos dados, então busca um maior refinamento (fazendo uso de outros atributos) na tentativa de gerenciar os dados restantes.

	Treinamento	**Teste**	**Trei./Teste**	**Tamanho**	**N. Aval.** ($\times 10^6$)
Trein.	**0,973 ± 0,008**	$0,952 \pm 0,013$	$0,963 \pm 0,006$	$28,9 \pm 6,6$	$3,07 \pm 1,57$
Mín.	0,962	0,931	0,953	21,0	0,683
Máx.	0,983	0,972	0,972	41,0	5,85
Teste	$0,957 \pm 0,009$	**0,972 ± 0,010**	$0,964 \pm 0,006$	$28,3 \pm 9,5$	$2,72 \pm 1,77$
Mín.	0,941	0,959	0,953	16,0	0,662
Máx.	0,972	0,986	0,972	46,0	6,36
T./T.	$0,965 \pm 0,010$	$0,970 \pm 0,011$	**0,967 ± 0,005**	$34,0 \pm 16,2$	$3,84 \pm 1,68$
Mín.	0,948	0,952	0,960	16,0	1,24
Máx.	0,979	0,986	0,974	71,0	6,60

Tabela 5.16: Resultados mais parcimoniosos sobre a predição partidária.

Esta versão conta com 420 dos 435 registros classificados corretamente, correspondendo a 96,5% de precisão.

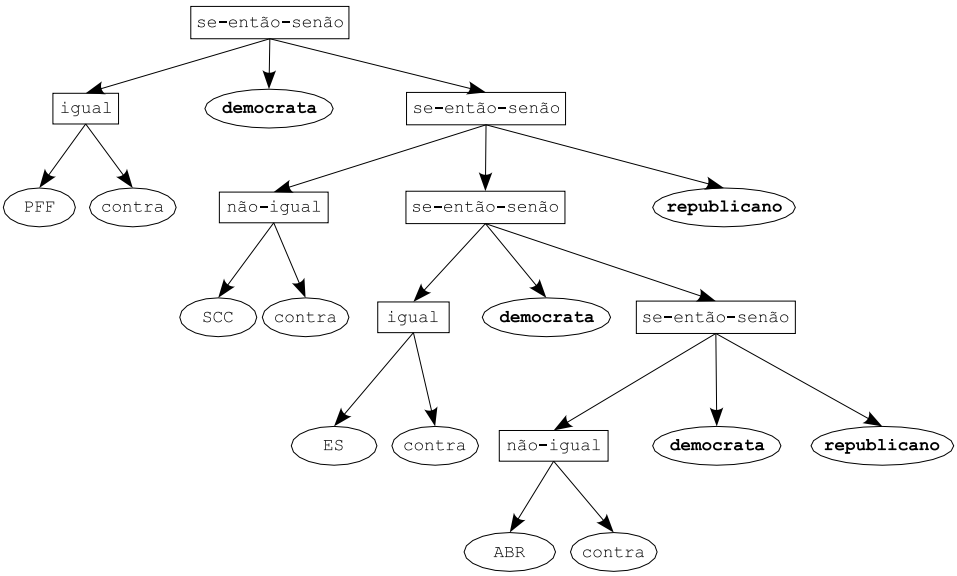

Figura 5.7: Árvore classificadora predição partidária.

5.5 Bases de Dados Artificiais *Monks*

Esta coleção de problemas tem uma história interessante [51]. Em julho de 1991, o convento de monges Corsendonk, na Bélgica, foi sede da segunda escola européia de verão em aprendizado de máquina [17]. Após mais de uma semana de exposição sobre variados algoritmos de aprendizagem, os participantes ficaram se perguntando quais seriam

[17] *Machine Learning.*

os melhores algoritmos e quais deveriam ser descartados. Como conseqüência desta inquietação, foram criados três problemas nos quais os algoritmos propostos poderiam ser comparados. Surgiu então o conjunto de problemas denominado *Monk's Problem*.

Os problemas são relativos a área de robôs artificiais, onde os robôs são descritos por seis atributos (Tabela 5.17), cada registro culminando em uma classificação binária.

Atributo	**Valores possíveis**
forma da cabeça	redonda, quadrada ou octogonal
forma do corpo	redonda, quadrada ou octogonal
está sorrindo?	sim ou não
objeto atachado	espada, balão ou bandeira
cor do casaco	vermelho, amarelo, verde ou azul
tem laço?	sim ou não

Tabela 5.17: Descrição dos atributos para a coleção monks.

Todas as possíveis combinações entre valores de atributos resultam em 432 tuplas [18]. Cada problema é criado artificialmente por descrições lógicas, sendo que um dado robô (registro) pode pertencer ou não a esta descrição. Esta sentença é responsável por segregar a base em duas classes: as que satisfazem a expressão lógica e as que não satisfazem. A aplicação do conceito lógico sobre o total de 432 registros define, então, a base de dados para o problema. Diferentemente dos outros experimentos, a divisão de dados para o conjunto de treinamento e teste foram previamente fixadas, apesar do conjunto de treinamento ter sido extraído aleatoriamente. Os criadores do *monks* optaram por atribuir ao grupo de teste todos os 432 registros, portanto, como esta quantidade descreve todo o problema, o grupo de treinamento é um subconjunto do conjunto de teste. Outra característica desta coleção é o fato do conjunto de treinamento ser expressivamente menor que o conjunto de teste. Isto significa que o sucesso de qualquer algoritmo sobre *monks* depende de uma boa qualidade de generalização.

5.5.1 *Monks* I

Este primeiro problema da série é construído por uma simples expressão lógica, na forma normal disjuntiva (FND): **(forma da cabeça** $=$ **forma do corpo)** $\vee$ **(cor do casaco**

[18] Na ordem apresentada na Tabela 5.17, tem-se as seguintes cardinalidades dos valores dos atributos: 3,3,2,3,4,2. A multiplicação destas ($3 \times 3 \times 2 \times 3 \times 4 \times 2$) fornece toda a gama de possibilidades combinatórias para o problema *monks*.

= **vermelho)**. Com esta sentença formou-se, então, a base de dados contendo 432 registros, sendo que 124 destes foram destinados ao conjunto de treinamento. O conjunto de teste engloba todos os registros, isto é, 432 linhas.

Os parâmetros utilizados nesta tarefa podem ser visualizados na Tabela 5.18.

Parâmetro	**Valor**
Número de gerações	100
Tamanho da população	500
Profundidade inicial	15
Profundidade de mutação	15
Peso penalização complexidade	0,2

Tabela 5.18: Parâmetros para o problema *monks* I.

A Tabela 5.19 mostra o desempenho do sistema proposto para este primeiro problema da coleção *monks*. Cada execução foi realizada em um curtíssimo espaço de tempo, gastando em média apenas 20 segundos.

	Trei./Teste	**Tamanho**	**N. Aval.** ($\times 10^5$)
	$1,000 \pm 0,000$	$13,3 \pm 5,4$	$1,36 \pm 0,86$
Mín.	1,000	10,0	0,288
Máx.	1,000	28,0	2,83

Tabela 5.19: Resultados sobre a base artificial *monks* I.

O algoritmo deste trabalho obteve 100% de aproveitamento em todas as execuções. Thrun e outros[51], criadores do relatório de desempenho de diversos algoritmos sobre estes problemas, listam resultados de 25 diferentes algoritmos, em cada um dos três problemas. Para o *monks* I, especificamente, nove algoritmos foram unânimes na classificação, catorze ficaram abaixo de 100% e outros dois não esboçaram resultado.

Observa-se nos resultados o tamanho simplório das soluções obtidas. A evolução pôde formar árvores que correspondem exatamente à forma como a base de dados foi definida, como se uma regressão estivesse sendo realizada. Facilidade esta provida pela flexibilidade da gramática definida, principalmente pelo fato de permitir comparação entre atributos. Algoritmos baseados no conceito estrito de árvores de decisão (*à la* C4.5) seriam obrigados a desenvolver soluções "acrobáticas" para aprender o problema, resultando em soluções mais complexas e curva de aprendizado proporcional à cardinalidade do conjunto de valores possíveis para os atributos envolvidos.

5.5.2 *Monks* II

O segundo problema da coleção *monks* assemelha-se aos problemas de paridade [19]. É descrito por: **quaisquer dois atributos, e somente estes, têm como valor o primeiro elemento do domínio de possíveis valores**. Por exemplo, se "forma da cabeça" e "forma do corpo" forem "redondas", o robô pertenceria a classe se, e somente se os atributos "está sorrindo?", "objeto atachado", "cor do casado" e "tem laço?" possuissem valores diferentes de "sim", "espada", "vermelho" e "sim", respectivamente. Certamente uma dura tarefa para sistemas que descrevem soluções por meio de forma normal disjuntiva ou conjuntiva, o que abrange este trabalho.

Dos 432 registros, 169 foram selecionados ao acaso para compor o grupo de treinamento. Os parâmetros utilizados estão presentes na Tabela 5.20.

Parâmetro	Valor
Número de gerações	5000
Tamanho da população	500
Profundidade inicial	15
Profundidade de mutação	20
Peso penalização complexidade	$0,05$

Tabela 5.20: Parâmetros para o problema *monks* II.

Devido a grande complexidade da tarefa em questão, algumas medidas em relação aos parâmetros foram tomadas:

- o número máximo de gerações foi elevada a 5000 gerações, buscando-se assim um refinamento sistemático;
- julgou-se necessário a criação de árvores maiores [20] durante a evolução, portanto, a profundidade de mutação passou para 20 níveis;
- em conformidade com o anseio de soluções complexas, a penalização por tamanho foi relaxada.

[19] O problema da paridade objetiva determinar se em um dado conjunto de valores binários (verdadeiro ou falso), o número de ocorrências de um dos valores é par (ou ímpar). É um problema inerentemente árduo, não-linear e não-monotônico [33, 29]

[20] De fato, uma análise intuitiva revela que este problema requer a construção de árvores capazes de testarem metodicamente todas as possibilidades. Isto é, não é viável a criação de soluções genéricas restringindo-se à utilização de formas normais conjuntivas e disjuntivas.

Os resultados obtidos através dos experimentos podem ser visualizados na Tabela 5.21. A dificuldade de resolução deste problema exigiu um grande esforço computacional, repercutindo-o no elevado tempo de cômputo, sendo necessário, para cada execução, aproximadamente 65 minutos.

	Treinamento	**Teste**	**Tamanho**	**N. Aval.** ($\times 10^7$)
Trein.	**0,999 ± 0,002**	$0,966 \pm 0,023$	$156,4 \pm 126,9$	$1,74 \pm 0,472$
Mín.	0,994	0,912	91,0	1,30
Máx.	1,000	0,995	513,0	3,01
Teste	$0,996 \pm 0,006$	**0,971 ± 0,022**	$241,5 \pm 348,8$	$1,67 \pm 0,50$
Mín.	0,988	0,917	91,0	1,28
Máx.	1,000	0,995	1225,0	3,01

Tabela 5.21: Resultados sobre a base artificial *monks* II.

Apesar do avantajado tamanho médio das soluções obtidas (o que já era previsto), observou-se um ótimo desempenho em relação ao índice de classificação. Percebe-se ainda um sutil indicativo de super treinamento, motivado pela configuração consciente dos parâmetros. Em especial, os "vilões" são a complacência pela complexidade e o alto ciclo de refinamento.

Em relação ao trabalho de referência [51], apenas quatro algoritmos conseguiram a classificação de 100%, vinte ficaram abaixo desse teto e um não declarou qualquer resultado. Dentre os que não conseguiram a classificação perfeita, o melhor índice obtido foi de 93,1%. No tocante aos quatro absolutos, três algoritmos não são simbólicos (baseados em redes neurais). O único algoritmo simbólico com acurácia de 100%, denominado **AQ17-DCI**, atua sobre o nível dos atributos, sendo projetado para derivar variáveis quando necessário, como um jogador com "cartas na manga".

O melhor resultado alcançado pelo algoritmo deste trabalho, sobre o problema *monks* II, foi capaz de classificar 99,5% (430/432) das amostras no conjunto de teste, possuindo a complexidade de 101 nós.

5.5.3 *Monks* III

Finalmente, o último problema da série *monks* é descrito na forma normal disjuntiva (FND), assim como o primeiro, porém, com um adicional: são inseridos ruídos na classificação. Isto é, alguns registros do conjunto de treinamento são propositalmente

classificados erroneamente. Mais precisamente, 5% das amostras de treinamento estão equivocadas quanto a classe. A sentença que define a base de dados é seguinte: **(cor do casaco** $=$ **verde** $\wedge$ **objeto atachado** $=$ **espada)** $\vee$ **(cor do casaco** $\neq$ **azul** $\wedge$ **forma do corpo** $\neq$ **octogonal)**.

O objetivo primordial deste problema é averiguar o poder de generalização dos algoritmos candidatos. Algoritmos "gulosos" por classificar todo e qualquer dado estão fadados ao fracasso em *monks* III. Exige-se, então, a virtude da robustez para com as amostras enganosas. Não obstante, este é o problema da série com a menor razão dados treinamento/teste, utilizando-se apenas 122 registros para o aprendizado.

As configurações de valores de entrada para a seqüência de execuções deste problema estão dispostas na Tabela 5.22.

Parâmetro	**Valor**
Número de gerações	500
Tamanho da população	500
Profundidade inicial	15
Profundidade de mutação	15
Peso penalização complexidade	$0,15$

Tabela 5.22: Parâmetros para o problema *monks* III.

Na Tabela 5.23 estão sumarizados os índices de desempenho para o algoritmo deste trabalho. Cerca de 8 minutos foram precisos para a obtenção dos resultados de cada execução.

	Treinamento	**Teste**	**Tamanho**	**N. Aval.** ($\times 10^6$)
Trein.	**0,963** $\pm$ **0,027**	$0,971 \pm 0,033$	$46,5 \pm 35,5$	$4,12 \pm 3,26$
Mín.	$0,934$	$0,907$	$11,0$	$0,59$
Máx.	$1,000$	$1,000$	$110,0$	$9,04$
Teste	$0,948 \pm 0,007$	**0,994** $\pm$ **0,012**	$32,6 \pm 21,8$	$3,29 \pm 2,78$
Mín.	$0,934$	$0,972$	$11,0$	$0,59$
Máx.	$0,951$	$1,000$	$87,0$	$8,96$

Tabela 5.23: Resultados sobre a base artificial *monks* III.

Constata-se a boa atuação do sistema, absorvendo muito bem os dados ruidosos. Dentre os experimentos, em 80% das execuções o algoritmo mostrou-se capaz de classificar corretamente todas as amostras do conjunto de teste. Na listagem provida por Thrun [51],

apenas cinco algoritmos atingiram classificação perfeita, quatro não se pronunciaram, e os dezesseis restantes alcançaram no máximo 97,2% de precisão.

5.6 Detecção de Tipos de Imagens

O problema de detecção de imagens envolve o reconhecimento de alguns padrões usualmente encontrados na natureza e construções, por exemplo "folhagem", "cimento", entre outros. Os registros da base de dados foram criados, aleatoriamente, a partir de sete imagens, cada qual representando uma região de 3×3 *pixels*. Para cada amostra, 19 atributos contínuos fornecem dados sobre propriedades da região de imagem representada, como por exemplo a saturação, densidade, intensidade, *hue* e outros. Uma dada região pode ser enquadrada em sete classes distintas, a saber: **tijolo**, **céu**, **folhagem**, **cimento**, **janela**, **estrada** e **grama**. A base de dados totaliza 2310 registros.

Este problema possui pelo menos três características que o tornam atraente e desafiador: expressivo número de registros, alta dimensionalidade dos atributos e significativa quantidade de classes.

5.6.1 Experimentos

Inicialmente tentou-se enfrentar o problema com os parâmetros apresentados na Tabela 5.24. Devido à complexidade intrínseca da tarefa em questão, ao tamanho da população foi atribuído um valor consideravelmente maior que os até então praticados. Nota-se também que o valor designado ao peso de penalização por complexidade tende a induzir a soluções mais enxutas. Os resultados desta experiência podem ser vistos na Tabela 5.25. A dificuldade impôs um tempo médio perto de 80 minutos por execução.

Parâmetro	Valor
Número de gerações	1000
Tamanho da população	1000
Profundidade inicial	10
Profundidade de mutação	10
Peso penalização complexidade	0,15

Tabela 5.24: Parâmetros para o problema de detecção de imagens.

	Treinamento	**Teste**	**Trei./Teste**	**Tamanho**	**N. Aval.** ($\times 10^7$)
Trein.	**0,926 ± 0,041**	$0,834 \pm 0,039$	$0,880 \pm 0,018$	$84,3 \pm 5,7$	$2,96 \pm 0,785$
Mín.	0,857	0,757	0,857	75,0	0,915
Máx.	0,979	0,900	0,914	93,0	3,61
Teste	$0,881 \pm 0,043$	**0,884 ± 0,033**	$0,883 \pm 0,029$	$86,2 \pm 8,6$	$2,38 \pm 1,12$
Mín.	0,821	0,814	0,832	65,0	0,835
Máx.	0,964	0,929	0,925	95,0	3,61
T./T.	$0,915 \pm 0,047$	$0,870 \pm 0,046$	**0,893 ± 0,022**	$87,2 \pm 6,7$	$2,74 \pm 0,910$
Mín.	0,821	0,771	0,868	75,0	0,821
Máx.	0,971	0,929	0,929	94,0	3,61

Tabela 5.25: Resultados iniciais para o problema de detecção de imagens.

Apesar do maior tamanho da população, o índice médio de classificação das soluções ficou aquém do desejado, principalmente os relativos ao conjunto de teste. A fim de atingir um melhor resultado, um segundo experimento foi então realizado, variando-se apenas o peso da penalização, que agora corresponde a $0,05$. O sumário desta tentativa está mostrado na Tabela 5.26. Os ganhos em termos de acurácia são bastante salientes quando comparados ao experimento anterior, tanto nos conjuntos de treinamento e teste a melhora ficou evidente. O interessante é que apenas um parâmetro foi modificado, a penalização por complexidade. Permitiu-se assim uma maior folga no tamanho das árvores geradas. Deduz-se, portanto, que o limite teórico na complexidade, da primeira abordagem, mostrou-se rudimentar para descrever o "conceito" da base de dados, funcionando como um gargalo. Contudo, o afrouxo da penalização sobre soluções maiores influenciou de forma negativa o tempo médio de processamento, exigindo agora 110 minutos por execução.

	Treinamento	**Teste**	**Trei./Teste**	**Tamanho**	**N. Aval.** ($\times 10^7$)
Trein.	**0,956 ±0,029**	$0,861 \pm 0,071$	$0,909 \pm 0,037$	$165,9 \pm 24,1$	$3,12 \pm 0,423$
Mín.	0,900	0,700	0,829	129,0	2,60
Máx.	0,993	0,943	0,957	193,0	3,62
Teste	$0,900 \pm 0,033$	**0,924 ±0,018**	$0,912 \pm 0,016$	$150,5 \pm 29,7$	$2,19 \pm 0,745$
Mín.	0,857	0,900	0,889	102,0	0,943
Máx.	0,943	0,957	0,936	186,0	3,28
T./T.	$0,941 \pm 0,036$	$0,913 \pm 0,022$	**0,927 ±0,018**	$168,9 \pm 25,2$	$2,71 \pm 0,835$
Mín.	0,871	0,871	0,900	128,0	0,943
Máx.	0,986	0,943	0,957	189,0	3,54

Tabela 5.26: Resultados para o problema de detecção de imagens.

Quando defrontado com alguns resultados encontrados na literatura, percebe-se que os índices encontrados neste trabalho superam algoritmos fomentados pelo mesmo motor (programação genética) mas ainda estão inferiores aos algoritmos de aprendizado de máquina. Esta síntese pode ser vista na Tabela 5.27. Os algoritmos ***Parallel* GP** [22] e **GP** [12] são da área de programação genética. Os restantes [21] pertencem ao domínio de aprendizado de máquina, geradores de árvores de decisão. Os valores contidos na tabela foram obtidos por validação cruzada.

Algoritmo	Acurácia
Parallel GP	$0,862$
GP	$0,833$
OC1	$0,954$
C4.5	$0,968$
C5.0	$0,968$
M5'	$0,970$

Tabela 5.27: Desempenho de outros algoritmos sobre o problema de detecção de imagens.

5.7 Conclusão

Foi visto neste capítulo a aplicação prática do sistema simbólico de classificação de dados implementado nesta dissertação, cujo desenvolvimento envolveu basicamente três técnicas: programação genética, gramática livre de contexto e co-evolução competitiva. Realizou-se experimentos sobre diferentes bases de dados, cada qual apresentando desafios e características próprias.

Os resultados mostraram-se bastante animadores, manifestando índices de classificação superiores à maioria dos algoritmos comparados. No entanto, no experimento com a base de dados de imagens, os resultados ficaram razoavelmente aquém do esperado, embora tenham sobrepujados significativamente os índices dos outros algoritmos baseados em programação genética. Enfatiza-se, ainda, o fato de que foi adotada uma gramática genérica e possivelmente redundante, sem qualquer otimização ou direcionamento para um perfil de problema.

No tocante ao custo computacional, devido à dificuldade de dados fáticos disponíveis

[21] Com exceção do C4.5 [44], os demais algoritmos de aprendizado de máquina tiveram seus índices de classificação retirados em [12].

na literatura —agravado pela natureza distinta de medida nos algoritmos comparados—, não se pôde mensurar comparativamente o tempo de processamento. É sabido, entretanto, que o sistema, por descender do paradigma da computação evolucionária, inclina-se a ter um custo de processamento relativamente alto.

O próximo e último capítulo traça um panorama comentado dos prós e contras das abordagens adotadas neste trabalho, bem como indicações para trabalhos futuros.

Capítulo 6

Conclusões

Este trabalho de dissertação foi construído sobre três principais alicerces: a programação genética, a representação gramatical e a co-evolução. Estes pilares foram apresentados, descritos separadamente, e então reunidos no contexto global do sistema. Desta integração conceitual nasceu a implementação prática, cujo objetivo visou a classificação (supervisionada) de amostras de dados, possibilitando-se ainda a descoberta de regras teoricamente passíveis de interpretação humana. A implementação foi colocada em atividade pela atuação em variados problemas, de casos triviais, passando-se pelos artificiais e, finalmente, problemas reais. Os resultados foram bastante promissores, comparando-se positivamente em relação a outras implementações/abordagens encontradas na literatura.

O restante deste capítulo trata da avaliação e comentários sobre as técnicas envolvidas nesta dissertação. Prossegue então apontando algumas direções para trabalhos futuros.

6.1 Programação Genética

A programação genética, possivelmente representando a condição conceitual mais básica deste trabalho, destacou-se por trazer favoravelmente, dentre outros pontos:

- robustez – evidenciada pela habilidade natural de exploração "global" do espaço de busca, reforçando-se assim sua adequabilidade para problemas de domínios não triviais, como os da área de classificação de dados.
- pouca exigência sobre o domínio – assim como as técnicas da computação evolucionária, a PG exige relativamente pouco conhecimento acerca do domínio do

problema, construindo-se arranjos complexos através de especificações iniciais bastante relaxadas. Portanto, traduz-se em um processo mais automatizado.

- evolução de soluções simbólicas – fator importantíssimo para se resgatar o conhecimento implícito em uma massa "fria" de dados.

- flexibilidade e integração – a PG é bastante receptiva à integração com outras técnicas e modelos, como a adoção de diferentes estruturas de codificação, bem como o enriquecimento da evolução por outras técnicas. Este ponto foi bem explorado neste trabalho, inserindo-se na PG canônica a estrutura gramatical e a co-evolução competitiva.

Raramente um conceito está isento de contra-pontos, a programação genética não foge à regra. Um dos pontos mais criticados em relação à PG —e em geral aos algoritmos da computação evolucionária— é o elevado custo computacional, principalmente quando comparada às técnicas não estocásticas. Felizmente, duas características da PG podem amenizar esta fragilidade: o paralelismo intrínseco e a possibilidade de utilização de abordagens mais eficazes, como a co-evolução.

Outro aspecto da programação genética passível de crítica é o emprego inevitável de um número considerável de parâmetros de entrada, podendo inclusive atingir a casa das dezenas. A má determinação destas variáveis pode acarretar uma perda substancial na potência evolutiva, por conseguinte, expondo-se ao retardo no processo ou restringindo-se às soluções sub-ótimas. Certamente, os parâmetros de entrada empregados nos experimentos deste trabalho, determinados empiricamente, impactaram negativamente, em algum grau, nos resultados —quando comparados com os possíveis resultados obtidos através dos valores ideais (teóricos) dos parâmetros.

6.2 Programação Genética Gramatical

Substituiu-se a linguagem representativa típica da PG (baseada nos limitados conceitos de conjuntos funções e terminais) por uma estrutura formal, a gramática livre de contexto. A GLC dedica-se, através de um conceito sólido e amplo, a gerar a linguagem de representação dos indivíduos —programas implementados na estrutura árvore.

Este casamento entre a PG e a GLC foi denominado programação genética gramatical. A inserção da PGG introduziu benefícios como:

- formalização – traduziu-se na clareza e exatidão da definição das relações por meio de uma gramática, tratando das regras de interações entre os variados componentes de um classificador, como as restrições operacionais quanto ao tipo de dados. Limitando-se as possibilidades combinatórias entre as funções e terminais, através das restrições, reduz-se o espaço de busca, que por sua vez tende a acelerar o processo de otimização como um todo.

- semântica – trouxe a habilidade de estruturação focando-se no significado das relações e não tão somente nas restrições de compatibilidade de tipos.

- maleabilidade – responsável pela capacidade de definir/construir qualquer espectro de linguagem, das mais simples às complexas. É esta habilidade que permite, por exemplo, a utilização de expressões aritméticas como parte das decisões de uma árvore de classificação, e não tão somente comparações relacionais ao estilo "atributo-valor".

- expansibilidade – além de permitir de forma adequada abraçar problemas que exigem uma estrutura melhor definida, o uso da gramática ainda facilita a adoção de novos ingredientes de otimização, como a própria evolução da gramática [1]. O valor da expansibilidade recai sobre o fato de que necessidades futuras de melhoramentos podem ser supridas pela própria técnica em questão, ao invés de recorrer a outros métodos.

A avaliação quanto ao uso deste tipo de codificação foi bastante vantajosa, enfraquecendo-se talvez no aspecto de implementação, que é consideravelmente mais requintada em relação ao método padrão de representação. Contudo, a gramática proporciona facilidades únicas de pós-implementação, como alterações na estrutura gramatical sem muito esforço.

Naturalmente, a definição da gramática é de responsabilidade do especialista, assim como também o são os conjuntos funções e terminais da programação genética canônica. Muitas vezes uma gramática pode ser genérica de tal maneira a suportar uma certa

[1] A evolução da gramática não foi abordada nesta dissertação, entretanto, pretende-se estudá-la futuramente.

variação de problemas, como no caso da gramática única utilizada nos experimentos (Capítulo 5) desta dissertação. Todavia, a definição da gramática visando as peculiaridades do domínio faculta o surgimento de melhores soluções.

6.3 Co-evolução Amostra-Classificador

O terceiro principal componente envolvido neste trabalho foi a co-evolução competidor-competidor, cujo duelo foi protagonizado de um lado pelas amostras de dados, e do outro pelas árvores classificadoras. Esta técnica, além prover uma aproximação melhor emparelhada com a natureza biológica da evolução, agregou ao sistema as seguintes virtudes:

- busca "esperta" – focando-se nas amostras com baixo índice de classificação (e vice-versa nas melhores árvores classificadoras), a co-evolução direciona os esforços de busca para "regiões" ainda não dominadas, ao invés de insistir nas fáceis ou vencidas. O efeito imediato desta abordagem é estimular a aceleração do processo evolutivo e/ou produzir resultados mais refinados. Os resultados deste trabalho, obtidos pelo classificador, exploraram primariamente o quesito otimização (focado na precisão de classificação), com demérito no que tange o possível ganho em custo computacional.

- manutenção da diversidade – a evolução inclinava-se a bombardear (com as mais árduas amostras) as árvores classificadoras não ótimas, livrando-se assim das estagnação e conseqüente amenização de soluções parciais ou locais. A recíproca também é verdadeira, isto é, a "janela" de amostras, postas como conjunto de avaliação das árvores, estava em constante atualização, selecionando-se os casos de teste menos dominados pelos classificadores.

- ajuste dinâmico do critério de aptidão – na programação genética tradicional, uma função de aptidão é definida inicialmente, e a partir de então, torna-se imutável. Isto significa que indivíduos da primeira geração serão avaliados pelo mesmo critério dos indivíduos —teoricamente mais adaptados— das gerações avançadas. Esta avaliação imutável sobre indivíduos iniciais pode ser demasiadamente severa, inibindo-se assim o início da evolução. Por outro lado, a co-evolução fornece um ajuste gradativo da pressão de avaliação à medida que ambas populações evoluem. A

avaliação torna-se então automaticamente compatível com o nível da aptidão das populações.

Observa-se, contudo, que no contexto da classificação de dados, o ajuste dinâmico do critério de aptidão impõe pouca relevância à aceleração da evolução inicial, que teoricamente seria prejudicada pelo modelo canônico [2]. De fato, dado que geralmente na avaliação tradicional —envolvendo todas as amostras do conjunto de treinamento— há diversidade quanto à dificuldade de classificação das variadas amostras, um indivíduo (classificador) inicial teria alguma "nota" gradual e não tão somente a completa falha. Portanto, qualquer indivíduo, incluindo os fracassados, teriam uma medição de aptidão relativa a sua real proficiência, e não uma avaliação maniqueísta.

- expansão do domínio de aplicação da PG – apesar de não ter tido qualquer influência neste trabalho, cabe mencionar uma característica relevante da co-evolução. Algumas tarefas de otimização não possibilitam a definição de uma função de avaliação fixa. Dentre estas tarefas estão os problemas de otimização de estratégias onde não são conhecidos *experts*, como por exemplo jogos de dama, xadrez, entre outros. A co-evolução, pela adaptação dinâmica, pode levar a PG nestes domínios, através de, por exemplo, a evolução por competição entre variadas estratégias.

Não nas mesmas proporções como na PG, porém não ignorável, é a introdução de alguns novos parâmetros específicos à co-evolução. Por exemplo, as informações pertinentes à segunda população, como tamanho e método de seleção, memória do registro de encontros, etc.

Não obstante, a co-evolução competitiva, dada sua natureza, abre a possibilidade para uma questão delicada, conhecida como "o problema do ciclo", e será vista a seguir.

6.3.1 O Problema do Ciclo

Basicamente, o ciclo é a ocorrência alternada e repetitiva de estratégias (ou soluções) previamente descobertas pelo processo evolucionário. Considere duas populações **A** e **B** em co-evolução. Suponha que em um determinado momento do processo co-evolutivo, a

[2] No âmbito da classificação de dados via programação genética, uma definição típica da função de aptidão emprega a avaliação de cada indivíduo (classificador) contra todas as amostras do conjunto de treinamento, e não apenas um conjunto inteligentemente selecionado.

população **A** evolva uma estratégia E_{A1}, que derrota a estratégia corrente em **B**, seja qual for. A população **B**, em resposta à **A**, evolui a estratégia E_{B1}, que por sua vez supera E_{A1}, isto é, $E_{B1} > E_{A1}$. Não satisfeita, a população **A**, em contra resposta, evolui a estratégia E_{A2}, tal que $E_{A2} > E_{B1}$. A população **B**, pressionada por **A**, consegue evoluir a estratégia E_{B2}, vencendo a concorrente E_{A2} ($E_{B2} > E_{A2}$). Por um azar do processo co-evolutivo, a estratégia mais recente da população **B** (E_{B2}) coincidentemente é inferior à primeira estratégia desenvolvida pela população **A** (E_{A1}), isto é, $E_{A1} > E_{B2}$. Sendo assim, devido ao material genético ainda remanescente em **B**, a estratégia mais trivial a ser evoluída é a anteriormente descoberta E_{B1}, que domina E_{A1}. O ciclo portanto está formado, possivelmente caindo em um laço, alternando-se entre as estratégias prévias. A Figura 6.1 mostra visualmente a dinâmica do problema do ciclo.

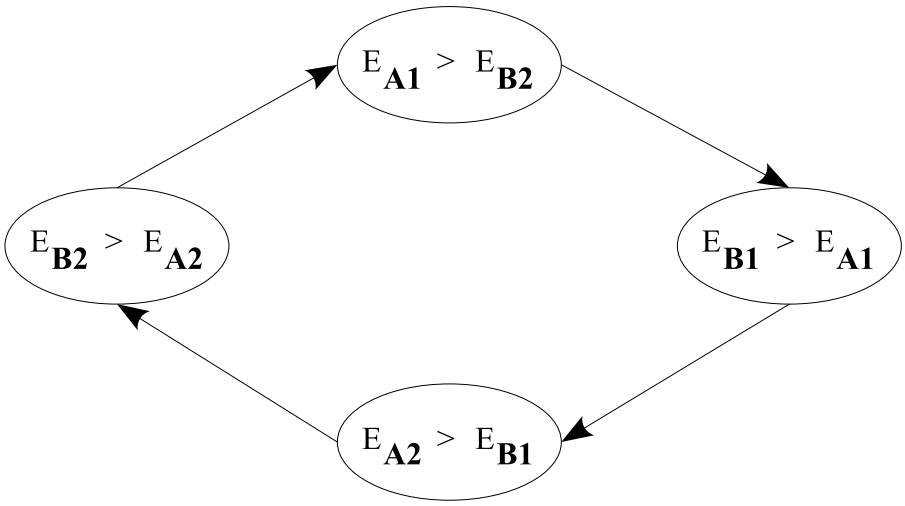

Figura 6.1: Ocorrência do ciclo na co-evolução. E_{xn} representa a estratégia n da população x.

No contexto deste trabalho, envolvendo a classificação de dados, o problema do ciclo basicamente se apresenta da seguinte forma. Assuma **A** como a população de árvores classificadoras e **B** como a população de amostras. Suponha que a população **A** evolva uma estratégia (estrutura/árvore) de classificação E_{A1}. Rapidamente, a população **B** organiza seu "time" de modo a derrubar E_{A1}. Chame esta "estratégia" de E_{B1}, portanto $E_{B1} > E_{A1}$. Em seqüência, **A** responde e evolui a estratégia E_{A2}, que é capaz de derrubar E_{B1}. Mais uma vez, **B** responde e cria E_{B2} a fim de derrotar E_{A2}, entretanto, E_{B2} é vencido por E_{A1}, e então ocorre o ciclo.

Tornando a explicação mais didática e real, pode-se atribuir "formas" às estratégias.

Seja a estratégia E_{A1} descrita como *classifique qualquer amostra como classe um* [3]. Reciprocamente, a estratégia E_{B1} é definida como *derrube qualquer classificador que classifique como classe um*, ou em outras palavras, *selecione amostras que não pertençam à classe um*. Analogamente, temos E_{A2} interpretada como *classifique qualquer amostra como classe dois* [4]. E ainda, E_{B2} descrita por *derrube qualquer classificador que classifique como classe dois*, ou *selecione amostras que não pertençam à classe dois*.

Sendo assim, primeiro a população **A** descobre a estratégia *classe um*, que é então vencida por *amostras que não pertençam à classe um*, da população **B**. Logo, surge a estratégia *classe dois*, de **A**, que por sua vez também é derrubada por *amostras que não pertençam à classe dois*, de **B**. Por fim, a população **A** evolui (resgata) a estratégia *classe um*, que vence *amostras que não pertençam à classe dois*, mas **B** também redescobre *amostras que não pertençam à classe um*, que por sua vez derrota *classe um*. Inicia-se o ciclo, como exibido graficamente na Figura 6.2.

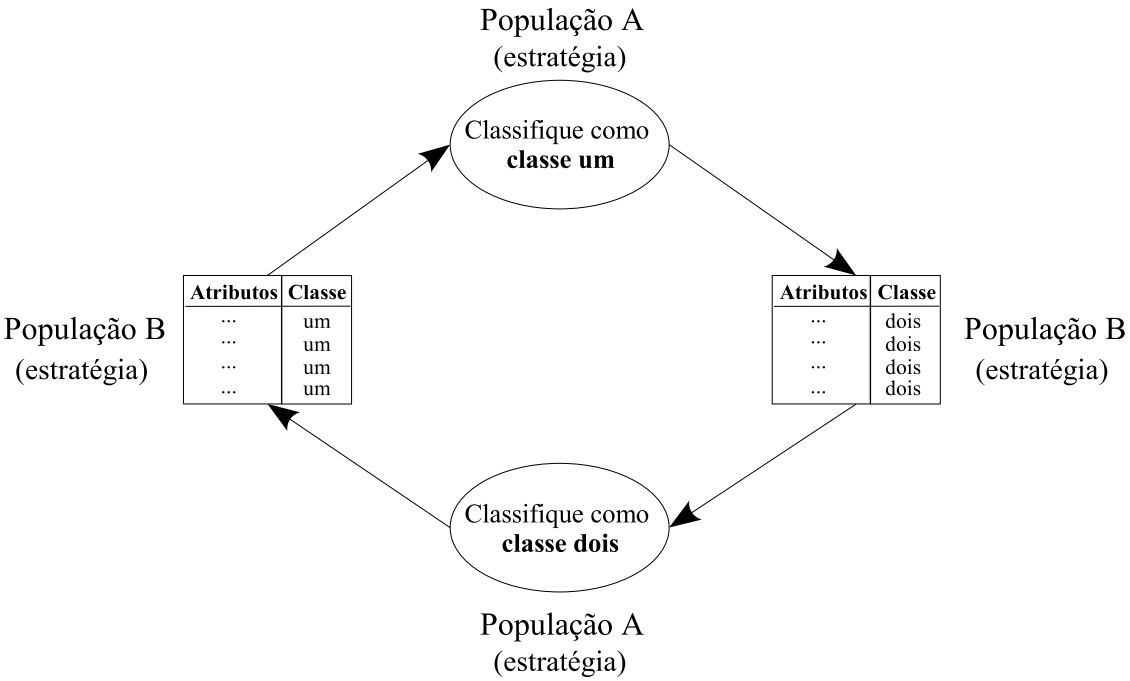

Figura 6.2: Ocorrência do ciclo na classificação amostra-classificador.

O aspecto que torna o problema do ciclo delicado é o fato da sua não trivial observação. É fácil quando se recorre a situações extremamente simplificadas, mas quando se trata de estratégias/estruturas complexas, sua identificação torna-se impraticável.

[3] Simplesmente: *retorne classe um*.
[4] *Retorne classe dois*.

Os possíveis efeitos colaterais da ocorrência do ciclo são o atraso no processo evolutivo e/ou estagnação em mínimos locais. Rosin e Belew, em *New Methods for Competitive Coevolution* [46], apesar de não mencionarem explicitamente o problema do ciclo, propuseram uma espécie de "elitismo temporal" ou "hall da fama", onde se adota uma avaliação ponderada com estratégias/soluções descobertas anteriormentes no decorrer da evolução. Em suma, testa-se na avaliação de cada indivíduo da geração corrente sua habilidade em lidar com soluções —não necessariamente todas— evoluídas em gerações passadas. Os autores ainda relembram outro benefício decorrente da conservação de soluções passadas, a preservação de material genético.

Stefano Nolfi e Dario Floreano [40] observam que a proposta de Rosin e Belew apresenta alguns pontos colaterais, como a não plausibilidade biológica e a redução da dinâmica co-evolucionária [5]. Entretanto, estes autores também observaram e relataram através de experimentos que o elitismo temporal contribui para a amenização do problema do ciclo.

Ainda, Nolfi e Floreano apontam um caminho alternativo para diminuir o problema do ciclo: enriquecer o ambiente. Os autores, utilizando o modelo predador-presa no contexto de robótica evolucionária, perceberam que a população de presas cujo sensores perceptivos são limitados —e portanto dificultando a fuga, bem como a evolução de estratégias mais aprimoradas— tendem a alternar repentinamente de estratégia, em vez de refinar a atual. Ampliando-se o campo perceptivo das presas, os autores observaram que a população de presas mais facilmente se inclinava a refinar a estratégia corrente ao invés de resgatar estratégias prévias. Os experimentos demonstraram resultados satisfatórios, assim como o elitismo temporal.

De fato, intuitivamente se espera que um ambiente rico, por possibilitar um rol maior de diferentes construções, bem como a melhoria destas, seja de certa forma ríspido para com a ocorrência do problema do ciclo.

Na implementação do trabalho desta dissertação, notou-se a incidência de ciclos em alguns experimentos, majoritariamente no início do processo evolutivo, nos moldes simplórios exibidos pela Figura 6.2. Isto é, dada a simplicidade das construções iniciais, a tensão co-evolutiva era atraída à alternância de soluções, favorecida pela facilidade e trivi-

[5] Nolfi e Floreano referiam-se ao fato de que a (contra)resposta a uma dada estratégia evoluída pela população concorrente seria mais lenta, pois o objetivo estaria disperso, dividido entre o "ataque" contra a estratégia corrente e contra as passadas.

alidade destas mudanças. Pareceu, entretanto, que nas gerações futuras, com construções mais complexas na população, e portanto maior abundância de material genético, o ciclo não se apresentava [6].

6.4 Trabalhos Futuros

Esta seção conta com algumas perspectivas de trabalhos futuros, visando aprimorar a eficiência de otimização (tempo e qualidade), bem como tentar eliminar alguns problemas identificados nas técnicas abordadas neste trabalho.

- **evolução gramatical** – trata-se de uma instigante linha de pesquisa, que compreende a evolução da própria gramática juntamente com o decorrer do processo evolutivo padrão. Isto é, adaptação e direcionamento das regras de produção de modo a otimizar a definição da gramática para o problema em questão. Em outros termos, a modificação da gramática implica na alteração (forma e "tamanho") do espaço de busca. Quando o espaço de busca é otimizado concomitantemente com as soluções, espera-se que a evolução torne-se mais eficaz. Whigham [57] introduziu esta noção e obteve resultados interessantes em seus experimentos.

- **exploração dos operadores genéticos gramaticais** – a programação genética gramatical possibilita uma série de outros operadores genéticos, que se aplicam na codificação por gramática. Dentre estes, estão por exemplo os operadores genéticos de encapsulamento, que funcionam como "blocos de construção". Construções que demonstram ser relevantes durante o processo de evolução são encapsuladas (apelidadas) em um único componente e então inseridas na definição da gramática, para que possam ser usadas na manutenção dos indivíduos. De fato, este ponto é uma especialização da "evolução gramatical", porém com objetivo mais direto e definido.

- **estudo do problema do ciclo** – este ponto merece atenção pelo impacto que pode trazer à co-evolução competitiva. Faz-se útil a investigação sistemática das condições de ocorrência, assim como meios de frear seu efeito negativo sem no entanto

[6] É importante salientar que, dada a natureza do problema do ciclo, a alternância mais elaborada de estratégias (como um ciclo mais longo) pode ter ocorrido, silenciosamente e com certa freqüência, também em gerações avançadas.

deturpar a caracterização do processo co-evolucionário. Técnicas como o elitismo temporal e o enriquecimento do ambiente podem ser melhor estudadas e integradas.

- **auto ajuste de parâmetros de entrada** – dada sua importância direta no processo evolucionário, o estudo e aplicação de técnicas desta área tornam-se um trabalho de pesquisa útil e relevante. Alguns pontos de partida presentes na literatura foram mencionados na Seção 5.1.1.

- **comparação fidedigna com a abordagem tradicional** – devido ao modelo de projeto de implementação [7] deste trabalho, não se pôde comparar o ganho real dos instrumentos aqui empregados em relação ao método tradicional de implementação. No entanto, acredita-se que o confronto com algoritmos e técnicas constantes na literatura, como exposto no Capítulo 5, somado ao sucesso relatado pelos autores dos trabalhos base [8] para esta dissertação, forneça uma noção suficientemente convincente da plausibilidade dos conceitos aqui empregados. Ainda assim, traçar o paralelo com a implementação canônica, usando-se a melhor aproximação de parâmetros, cumpre melhor o propósito científico, e portanto perdura como proposta para trabalho futuro.

[7] A integração da co-evolução competitiva resultou em uma implementação dependente (pouco modular), criando-se assim uma barreira prática (em termos de "custo/benefício") para uma manutenção conjunta do modelo tradicional e co-evolucionário.

[8] Destacam-se os trabalhos de Paredis [43], Whigham [57], Hillis [27], Angeline e Pollack [4], e Axelrod [6].

Referências Bibliográficas

[1] A. PAPAGELIS, D. K. GATree: Genetically evolved decision trees.

[2] AHO, A. V., SETHI, R., E ULLMAN, J. D. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, Reading, Mass., 1986.

[3] ANDRE, D., E TELLER, A. A study in program response and the negative effects of introns in genetic programming. In *Genetic Programming 1996: Proceedings of the First Annual Conference* (Stanford University, CA, USA, 28–31 julho de 1996), J. R. Koza, D. E. Goldberg, D. B. Fogel, e R. L. Riolo, Eds., MIT Press, pp. 12–20.

[4] ANGELINE, P. J., E POLLACK, J. B. Competitive environments evolve better solutions for complex tasks. In *Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93* (University of Illinois at Urbana-Champaign, 17-21 julho de 1993), S. Forrest, Ed., Morgan Kaufmann, pp. 264–270.

[5] AXELROD, R. *The Evolution of Cooperation*. Basic Books, New York, 1984.

[6] AXELROD, R. The evolution of strategies in the iterated prisoner's dilemma. In *Genetic Algorithms and Simulated Annealing* (1987), L. Davis, Ed., Morgan Kaufman, pp. 32–41.

[7] BANZHAF, W., NORDIN, P., KELLER, R. E., E FRANCONE, F. D. *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann, dpunkt.verlag, janeiro de 1998.

[8] BARBOSA, H. J. C. Uma introdução aos algoritmos genéticos. *Mini-Curso no XX CNMAC, Congresso Nacional de Matemática Aplicada e Computacional, Gramado, RS* (1997).

[9] BARRETT, W. A., BATES, R. M., GUSTAFSON, D. A., E COUCH, J. D. *Compiler construction: theory and practice (2nd ed.)*. SRA School Group, 1986.

[10] BEASLEY, D., BULL, D. R., E MARTIN, R. R. An overview of genetic algorithms: Part 1, fundamentals. *University Computing 15*, 2 (1993), 58–69.

[11] BLAKE, C., E MERZ, C. UCI repository of machine learning databases, 1998.

[12] BOT, M. C. J. Improving induction of linear classification trees with genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)* (Las Vegas, Nevada, USA, 10–12 2000), D. Whitley, D. Goldberg, E. Cantu-Paz, L. Spector, I. Parmee, e H.-G. Beyer, Eds., Morgan Kaufmann, pp. 403–410.

[13] BÄCK, T. Parallel optimization of evolutionary algorithms.

[14] CHOMSKY, N. Three models for the description of language. *IRA Transactions on Information Theory 2*, 3 (1956), 113–124.

[15] DARWIN, C. *On the Origin of Species by Means of Natural Selection.* Murray, London, 1859. 6th (final) ed'n 1872.

[16] DAWKINS, R. *The Selfish Gene*. Oxford University Press, 1976.

[17] DE BARY, H. A. *Die Erscheinung der Symbiose [The Phenomenon of Symbiosis]*. Velarg von Karl J. Trubner, 1879.

[18] DE JONG, K., E POTTER, M. A. Evolving complex structures via cooperative coevolution. In *Proc. on the Fourth Annual Conf. on Evolutionary Programming* (Cambridge, MA, 1995), MIT Press, pp. 307–317.

[19] DUCH W, ADAMCZAK R, G. K. A new methodology of extraction, optimization and application of crisp and fuzzy logical rules. *IEEE Transactions on Neural Networks (March 2001) 11*, 2 (2000).

[20] E. G. M. LACERDA, A. C. P. L. F. C. Introdução aos algoritmos genéticos. In *XVIII Jornada de Atualização em Informática (II Brazilian Artificial Intelligence Meeting), ENIA 99, XIX Congresso Brasileiro de Computação* (July 1999), pp. 51–126.

[21] FOLINO, G., PIZZUTI, C., E SPEZZANO, G. Genetic programming and simulated annealing: A hybrid method to evolve decision trees. In *Genetic Programming, Proceedings of EuroGP'2000* (Edinburgh, 15-16 abril de 2000), R. Poli, W. Banzhaf,

W. B. Langdon, J. F. Miller, P. Nordin, e T. C. Fogarty, Eds., vol. 1802 of *LNCS*, Springer-Verlag, pp. 294–303.

[22] FOLINO, G., PIZZUTI, C., E SPEZZANO, G. Parallel genetic programming for decision tree induction. In *Proceedings of the 13th International Conference on Tools with Artificial Intelligence* (Dallas, TX USA, 7-9 novembro de 2001), IEEE, pp. 129–135.

[23] FRANCISCO HERRERA, MANUEL LOZANO, J.-L. V. Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis. *Artificial Intelligence Review 12*, 4 (1998), 265–319.

[24] GALAV'IZ, J. A self-adaptive genetic algorithm for function optimization. In *Proceedings of IX International Symposium on Artificial Intelligence* (Cancun, Mexico, 7-9 1996).

[25] GOLDBERG, D. E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, Mass., 1989.
An introductory textbook and guide to current research. Contains examples, exercises, and sample programs as well as proofs of most of the major GA theorems on crossover and implicit parallelism.

[26] HAYNES, T., WAINWRIGHT, R., SEN, S., E SCHOENEFELD, D. Strongly typed genetic programming in evolving cooperation strategies. In *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)* (Pittsburgh, PA, USA, 15-19 1995), L. Eshelman, Ed., Morgan Kaufmann, pp. 271–278.

[27] HILLIS, W. D. Co-evolving parasites improve simulated evolution as an optimization procedure. *Phys. D 42*, 1-3 (1990), 228–234.

[28] HOLLAND, J. H. *Adpatation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975.

[29] HUELSBERGEN, L. Finding general solutions to the parity problem by evolving machine-language representations. In *Genetic Programming 1998: Proceedings of the Third Annual Conference* (University of Wisconsin, Madison, Wisconsin, USA, 22-25 julho de 1998), J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, e R. Riolo, Eds., Morgan Kaufmann, pp. 158–166.

[30] IORIO, A., E LI, X. Parameter control within a co-operative co-evolutionary genetic algorithm. In *Proceedings of The Seventh International Conference on Parallel Problem Solving from Nature - PPSN VII*, J. Merelo Guervos, P. Adamidis, H. Beyer, J. Fernandez-Villacanas, e H. Schwefel, Eds., Lecture Notes in Computer Science (LNCS 2439), pp. 247–256.

[31] KNUTH, D. E. *The Art of Computer Programming: Sorting and Searching*, vol. 3. Addison-Wesley, New York, 1973.

[32] KOZA, J. R. Genetic programming: a paradigm for genetically breeding populations of computer programs to solve problems. Technical Report CS-TR-90-1314, Stanford University, Department of Computer Science, junho de 1990.

[33] KOZA, J. R. *Genetic programming: on the programming of computers by means of natural selection*. MIT Press, Cambridge, MA, 1992.

[34] KOZA, J. R. *Genetic programming: On the programming of computers by natural selection*. MIT Press, Cambridge, Mass., 1992.

[35] MANACHER, G. K. An improved version of the Cocke-Younger-Kasami algorithm. *j-COMP-LANGS 3*, 2 (1978), 127–133.

[36] MANGASARIAN, O. L., STREET, W. N., E WOLBERG, W. H. Breast cancer diagnosis and prognosis via linear programming. Relatório Técnico MP-TR-1994-10, 1994.

[37] MONTANA, D. J. Strongly typed genetic programming. Relatório Técnico #7866, Bolt Beranek and Newman, Inc., 10 Moulton Street, Cambridge, MA 02138, USA, 7 1994.

[38] MORRISON, J. Co-evolution and genetic algorithms. Tese de Mestrado, University of Carleton, Ottawa, Ontario, 1998.

[39] MYUNG WON KIM, JOONG GEUN LEE, C. M. Efficient fuzzy rule generation based on fuzzy decision tree for data mining. In *Fuzzy Systems Conference Proceedings, 1999. FUZZ-IEEE '99.* (1999), vol. 3, IEEE International, pp. 1223–1228.

[40] NOLFI, S., E FLOREANO, D. How co-evolution can enhance the adaptive power of artificial evolution: Implications for evolutionary robotics. In *EvoRobots* (1998), pp. 22–38.

[41] O'NEILL, M., E RYAN, C. Grammatical evolution. *IEEE Transactions on Evolutionary Computation 5*, 4 (2001), 349–358.

[42] PAREDIS, J. The evolution of behavior: some experiments. In *Proceedings of the first international conference on simulation of adaptive behavior on From animals to animats* (1991), MIT Press, pp. 419–426.

[43] PAREDIS, J. Steps towards co-evolutionary classification neural networks. In *Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems* (1994), pp. 102–108.

[44] QUINLAN, J. Improved use of continuous attributes in C4.5. *Journal of Artificial Intelligence Research 4* (1996), 77–90.

[45] RICARTE, I. L. M. Programação de sistemas: Uma introdução, 2001.

[46] ROSIN, C. D., E BELEW, R. K. New methods for competitive coevolution. *Evolutionary Computation 5*, 1 (1997), 1–29.

[47] ROUWHORST, S. E., E ENGELBRECHT, A. P. Searching the forest: Using decision trees as building blocks for evolutionary search in classification databases. In *Proc. of the 2000 Congress on Evolutionary Computation* (Piscataway, NJ, 2000), IEEE Service Center, pp. 633–638.

[48] RYAN, C., COLLINS, J. J., E O NEILL, M. Grammatical evolution: Evolving programs for an arbitrary language. In *Proceedings of the First European Workshop on Genetic Programming* (Paris, 14-15 1998), W. Banzhaf, R. Poli, M. Schoenauer, e T. C. Fogarty, Eds., vol. 1391, Springer-Verlag, pp. 83–95.

[49] SMITH, J., E FOGARTY, T. C. Self adaptation of mutation rates in a steady state genetic algorithm. In *International Conference on Evolutionary Computation* (1996), pp. 318–323.

[50] SURRY, P. D., E RADCLIFFE, N. J. Real representations. In *Foundations of Genetic Algorithms 4*, R. K. Belew e M. D. Vose, Eds. Morgan Kaufmann, San Francisco, CA, 1997, pp. 343–363.

[51] THRUN, S. B., BALA, J., BLOEDORN, E., BRATKO, I., CESTNIK, B., CHENG, J., JONG, K. D., DŽEROSKI, S., FAHLMAN, S. E., FISHER, D., HAMANN, R.,

KAUFMAN, K., KELLER, S., KONONENKO, I., KREUZIGER, J., MICHALSKI, R. S., MITCHELL, T., PACHOWICZ, P., REICH, Y., VAFAIE, H., DE WELDE, W. V., WENZEL, W., WNEK, J., E ZHANG, J. The MONK's problems: A performance comparison of different learning algorithms. Relatório Técnico CS-91-197, Pittsburgh, PA, 1991.

[52] TUSON, A., E ROSS, P. Adapting operator settings in genetic algorithms. *Evolutionary Computation 6*, 2 (1998), 161–184.

[53] VAN BENEDEN, P. J. *Les commensaux et les parasites dans le regne animal [Commensalists and Parasites of the Animal Kingdom]*. Paris: G. Bailliere, 1875, 1875.

[54] WALL, M. GAlib: A C++ library of genetic algorithm components, 1996.

[55] WANG, G., DEXTER, T., GOODMAN, E., E PUNCH, W. Optimization of a GA and within the GA for a 2-dimensional layout problem, 1996.

[56] WHIGHAM, P. A. Grammatically-based genetic programming. In *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications* (Tahoe City, California, USA, 9 1995), J. P. Rosca, Ed., pp. 33–41.

[57] WHIGHAM, P. A. *Grammatical Bias for Evolutionary Learning*. PhD thesis, School of Computer Science, University College, University of New South Wales, Australian Defence Force Academy, Canberra, Australia, 14 outubro de 1996.