Alexandre Grings

Regressão Simbólica via Programação Genética: um Estudo de Caso com Modelagem Geofísica

Uberlândia

Fevereiro 2006

Alexandre Grings

Regressão Simbólica via Programação Genética: um Estudo de Caso com Modelagem Geofísica

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Ciência da Computação pelo Programa de Pós-Graduação da Faculdade de Computação da Universidade Federal de Uberlândia.

Orientador:
Prof. Dr. Antônio Eduardo Costa Pereira

Uberlândia

Fevereiro 2006

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO

Os abaixo assinados, por meio deste, certificam que leram e recomendam para a Faculdade de Computação a aceitação da dissertação intitulada "**Regressão Simbólica via Programação Genética: um Estudo de Caso com Modelagem Geofísica**" por **Alexandre Grings** como parte dos requisitos exigidos para a obtenção do título de **Mestre em Ciência da Computação**.

Uberlândia, 24 de fevereiro de 2006

Orientador: _____

Prof. Dr. Antônio Eduardo Costa Pereira
Universidade Federal de Uberlândia UFU/MG

Banca Examinadora: _____

Prof. Dr. João Bosco da Mota Alves
Universidade Federal de Santa Catarina UFSC/SC

_____

Prof^{a}. Dr^{a}. Márcia Aparecida Fernandes
Universidade Federal de Uberlândia UFU/MG

Dedicatória

A minha família.

A essa sutil sensação de que existe um algo mais, de que não somos apenas pó, de que, mesmo não fazendo sentido, a existência possui uma razão além do entendimento.

Agradecimentos

A minha esposa, Simonis, uma mulher que mudou muitos destinos para melhor, o meu inclusive. A minha irmã, Ana Cecília, que acreditou em mim mais do que eu mesmo. A meus pais, Edite e Almiro, pela torcida. Ao meu irmão, Cristiano, por agüentar meus desabafos.

Ao colega e amigo Alexsandro Soares, um grande cientista, sem o qual esse trabalho não chegado tão longe.

Ao professor Costa por toda a liberdade no trabalho e confiança na capacidade da nossa equipe.

Aos colegas que ajudaram com os modelos de teses, listas de publicações e dicas para a apresentação: Renê, Mylene, Juliana e Joslaine. Ao amigo Gustavo, alguém que realmente tem coragem de dizer o que pensa.

À professora Rita, a qual sempre teve uma palavra de incentivo e encorajamento. Ao professor Carlos e à Libéria da secretaria de Pós-graduação, pelo apoio com a parte burocrática e por sugestões práticas para problemas complicados. Ao André e Madalena da secretaria do departamento de Computação, sempre prestativos. Para a Cynthia, pela ajuda na matrícula e outras atitudes descomplicadoras. Para a Márcia e Bosco por uma extensão essencial no prazo.

Ao pessoal da CAPES pela bolsa.

A todos que, direta ou indiretamente, contribuíram para a realização deste trabalho.

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

A regressão simbólica, que consiste na manipulação de expressões matemáticas para descoberta de funções que descrevam um conjunto de dados, foi uma tarefa exclusivamente humana até pouco tempo atrás. Recentemente, foram desenvolvidas várias técnicas computacionais para automatizar a regressão simbólica. Uma dessas técnicas é a programação genética, uma subárea da computação evolutiva que usa analogia à teoria da evolução de Darwin e idéias do campo da Genética para desenvolver um grupo de programas de computador na busca por soluções a tarefas computacionais. O presente trabalho visa a testar as capacidades de regressão simbólica da programação genética com objetivo de verificar sua viabilidade como ferramenta para a pesquisa de um problema geofísico. Esse problema diz respeito a fenômenos que ocorrem na ionosfera, a região da atmosfera ionizada pela ação dos raios solares, que desempenham um papel fundamental para as telecomunicações. No intercurso dessa tentativa, faz-se o uso de duas implementações tradicionais de programação genética e de uma variante, chamada programação da expressão gênica. Problemas como o sistema estudado demandam muito tempo de processamento e memória, desse modo, o trabalho culmina com uma implementação distribuída de programação genética com o intuito de acelerar o processamento da modelagem.

Palavras-chave: regressão simbólica; programação genética; programação da expressão gênica; modelagem geofísica.

Abstract

Symbolic regression, which is in principal the handling of mathematical expressions for finding a function that describes a data set, was until recently carried out exclusively by humans. But now, several computational techniques of symbolic regression automatization have appeared. One of these techniques is genetic programming, a subarea of evolutive computing that uses an analogy to Darwin's evolutionary theory and some ideas from the Genetics field to develop a group of computer programs in a search for solutions to computational tasks. This work aims to test the symbolic regression capabilities of genetic programming with the objective of verifying its viability as a tool for a specific geophysical research. This research concerns phenomena that occurs in the ionosphere, the region of earth's atmosphere ionized by the action of solar rays, that play a fundamental role in telecommunications. In the course of this trial, we used two implementations of traditional genetic programming and one implementation of a variant, named gene expression programming. Problems like the one under study demand a lot of processor time and are memory consuming, therefore, the work culminates with a distributed implementation of genetic programming with the objective of accelerating the modeling process.

Key-words: symbolic regression; genetic programming; gene expression programming; geophisical modeling.

1 Introdução

1.1 Motivação

O Sol não é só fonte de luz e calor, como visto nas aulas de Geografia do primário: ele também é essencial para as telecomunicações. A radiação solar cria gases eletricamente carregados em parte da atmosfera terrestre. A esse processo dá-se o nome de ionização e a parte ionizada da atmosfera é denominada ionosfera. Gás eletricamente carregado, também conhecido como plasma, tem a propriedade de afetar ondas eletromagnéticas que o atravessam. Como quase todas as telecomunicações na Terra são via ondas eletromagnéticas (ondas de rádio), a ionosfera desempenha um papel fundamental nesta área. O plasma também tem outra propriedade que é a de ser afetado por campos magnéticos. Por isso, variações no campo magnético da terra, chamado magnetosfera, também agem sobre o plasma. O comportamento da magnetosfera, por sua vez, é influenciado pelo vento solar, um fluxo ininterrupto de plasma vindo do Sol. Assim, variações abruptas no comportamento do Sol podem acabar com uma partida de futebol da televisão via satélite.

Outros interesses, além de futebol, motivam os geofísicos a pesquisar esse sistema complexo. A área de geofísica espacial, entre outras coisas, estuda fenômenos da atmosfera terrestre. O estudo da influência da dinâmica solar na ionosfera revela-se essencial para se entender como as movimentações de plasma da ionosfera são afetadas pela atividade solar. Essa questão ficou particularmente importante depois do advento das comunicações por satélite. Prever com antecedência anomalias no plasma ionosférico ajudaria, por exemplo, a minimizar os ruídos nas comunicações que afetam a precisão de sinais de GPS. Antenas de radiotelescópios, fotômetros, geosondas, magnetômetros, foguetes, satélites e outros equipamentos espalhados pelo globo têm coletado dados relativos aos fenômenos ionosféricos e magnetosféricos ao longo de décadas. Esses dados têm sido usados na tentativa de descobrir modelos preditivos para as influências do vento solar e do campo magnético do Sol na ionosfera e magnetosfera terrestres.

Um modelo é uma representação abstrata de um sistema que é utilizada para entendê-lo

melhor. Geralmente, a construção de um modelo é orientada pelo uso que queremos fazer deste. Remove-se, então, tudo o que não é essencial para o uso específico do modelo. Assim, um mapa rodoviário, que é um modelo de uma região geográfica, não precisa ter informações sobre todos os detalhes da região (como relevo, tipos de solo e vegetação), descrevendo apenas informações relativas à malha rodoviária.

A questão motivadora do presente trabalho é a procura de um modelo matemático, a partir de um conjunto de dados, que permita prever o comportamento de um sistema (um modelo preditivo). Isso envolve alguns problemas: filtrar o ruído nos dados coletados, ajustar a dimensionalidade do problema, isolar as variáveis representativas e, finalmente, efetuar algum tipo de regressão matemática. Alguns autores dividem esse trabalho em três fases [1]. A primeira consiste em aplicar a navalha de Occam ao sistema do mundo real, gerando, assim, um modelo simplificado, uma abstração que contém apenas o essencial para o problema. A segunda fase envolveria a criação de um modelo matemático que descrevesse o modelo simplificado. Ao final, o modelo matemático criado seria testado e analisado no mundo real.

Um exemplo de um modelo preditivo, encontrado pela equipe de trabalho desta pesquisa, modela a deriva do plasma equatorial na região mais alta da atmosfera (região F) com base em alguns parâmetros sabidamente representativos para esse fenômeno [2]. Esse modelo foi obtido por um método gráfico que utiliza polinômios de Bernstein. Apesar de funcionar como modelo preditivo, essa aproximação gráfica não é suficiente para a pesquisa científica, pois as expressões obtidas não oferecem uma relação entre os parâmetros que seja compreensível do ponto de vista da Física. Ainda não se descobriu uma fórmula ou um conjunto de fórmulas que ajudasse os físicos a entender a dinâmica interna desses fenômenos. Em suma, os geofísicos estão à procura de uma "lei natural" para o sistema em questão.

1.2 Objetivo

O objetivo deste trabalho consiste em testar ferramentas baseadas em programação genética [3] que ajudem na busca de modelos, matemáticos ou físicos, para verificar sua viabilidade para o uso na pesquisa dos fenômenos de deriva de plasma mencionados.

Os geofísicos já têm alguma idéia de quais seriam as variáveis mais representativas no sistema descrito em [2]. Esse estudo foi feito com base em um modelo simplificado de quatro variáveis (fluxo solar, hora do dia, deriva zonal e atividade geomagnética), utilizando-se dados obtidos entre 1970 e 2003 pelo rádio observatório de Jicamarca (Peru).

A idéia principal de uma ferramenta computacional para ajudar na pesquisa do fenômeno

seria automatizar o trabalho na fase de geração do modelo matemático, ou seja, um programa que gerasse expressões matemáticas que servissem como modelos aproximados do fenômeno em questão e que pudessem ser analisadas por especialistas em geofísica espacial. Além disso, seria desejável que essa ferramenta pudesse ser facilmente adaptada para a solução de problemas semelhantes. Uma das formas de se conseguir essa automatização seria pelo uso de regressão simbólica por programação genética.

A regressão simbólica é um dos processos de busca de funções para um modelo matemático, que é descrito no formato de dados de entrada e saída. Ou seja, há informações sobre o comportamento do sistema em termos de parâmetros de entrada e saída, informação esta obtida por meios empíricos, mas se desconhece o seu funcionamento interno (função geradora). A procura por essa função geradora, a partir de dados observados, de um modelo matemático por meio da manipulação de expressões matemáticas (expressões simbólicas) é denominada regressão simbólica.

A programação genética faz uso de idéias tiradas de teoria da evolução de Charles Darwin e da Genética, para descobrir programas de computador que resolvam um dado problema computacional. A programação genética é uma das ferramentas computacionais que consegue automatizar a regressão simbólica. A regressão simbólica tem sido testada desde as primeiras implementações de programação genética e a literatura da área é rica em estudos de caso para problemas de modelagem matemática.

1.3 Organização da dissertação

Este trabalho está organizado em sete capítulos. O objetivo dos capítulos 2 e 3 é versar especificamente sobre modalidades de programação genética com uma revisão da literatura da área. No capítulo 4 descreve-se o problema geofísico e apresenta-se uma explicação a respeito das variáveis envolvidas em sua modelagem. O capítulo 5 descreve as escolhas tomadas na implementação do trabalho. O capítulo 6 descreve os resultados obtidos com as implementações feitas. Discussões sobre os resultados e conclusões são tecidas no capítulo final.

O capítulo 2 faz uma introdução geral à computação evolutiva, descrevendo os seus representantes principais. A seguir, faz uma comparação entre o algoritmo básico de programação genética e o de algoritmos genéticos. Descreve a modalidade tradicional de programação genética e faz uma rápida explanação sobre estruturas de dados utilizadas pelas variantes de programação genética. Fala sobre regressão simbólica e sobre possíveis alternativas ao uso de programação genética para o mesmo objetivo. Por fim, descreve alguns dos modelos de parale-

lização usados com programação genética.

Devido à opção por testar regressão simbólica com programação da expressão gênica, uma variante de programação genética, o capítulo 3 descreve-a com mais detalhes.

No capítulo sobre o problema geofísico, o enfoque está na importância dos fenômenos ionosféricos para a área de telecomunicações, explicando-se a influência do Sol neste sistema, sem grandes pretensões de aprofundamento em geofísica espacial. Considera-se também sobre como as informações são coletadas e pré-trabalhadas para a obtenção dos dados que são usados no estudo de caso com regressão simbólica.

O capítulo que versa sobre a implementação descreve o trabalho em duas fases: a tentativa com a modalidade tradicional de programação genética, com duas implementações distintas, e a segunda fase, em que se constrói um modelo distribuído cujo núcleo é um sistema de regressão simbólica usando programação da expressão gênica. Descreve as linguagens, sistemas e ferramentas utilizadas e o porquê das escolhas. Versa sobre características de cada implementação e descreve o modelo de processamento distribuído construído.

No capítulo 6, são mostrados alguns resultados obtidos com a modalidade tradicional e é feita uma comparação entre duas implementações diferentes desta. São mostrados testes simples que comparam programação da expressão gênica com a modalidade tradicional. São apresentados resultados de testes com a modelagem de um sistema físico mais simples, com o objetivo de testar a capacidade do sistema. São também descritos os resultados de testes com diferentes configurações para o sistema distribuído. Finalmente, são mostrados os resultados obtidos com o sistema implementado na segunda fase para o problema geofísico.

2 Programação genética

2.1 Histórico

Considerada muitas vezes como uma extensão dos algoritmos genéticos, criados por Holland [4], a programação genética possui seu mesmo algoritmo básico. A idéia fundamental é anterior a Holland, tendo sido proposta por Turing no final da década de 40. Em um artigo de 1950 [5], Turing apresenta noções gerais de como deveria funcionar uma "busca genética ou evolutiva" em que um grupo de programas seria modificado por meio de algum tipo de mutação e onde a seleção dos melhores dependeria do julgamento do experimentador. Como outros trabalhos pioneiros no campo, podem-se citar também:

- Friedberg [6] tentou evoluir programas a partir de mutações pontuais no código dos programas, chegando à conclusão de que esse tipo de mutação induzia a um comportamento excessivamente aleatório dos programas mutantes;
- Operação evolutiva ou EVOP (do Inglês *EVolutionary OPeration*), criada por George Box em 1962, consiste em uma técnica evolutiva para otimização de processos para projeto e análise de experimentos industriais;
- Hans-Joachim Bremermann, em 1962, desenvolveu uma algoritmo que unia idéias de teoria da computação e da biologia evolutiva, tentando aplicá-lo a problemas de otimização numérica e resolução de equações não-lineares;
- Um trabalho póstumo de John von Neumann [7], em 1966, propõe autômatos autoreplicantes como modelos implícitos de computação evolutiva.

A questão da criação da programação genética, nos moldes como é conhecida hoje, é polêmica. Apesar de ter sido patenteada por John Koza por volta de 1990 [8], há vários trabalhos anteriores. Nichael Cramer, em 1985, usou as idéias de Holland para evoluir programas de computador tanto com representações em árvore quanto em seqüências lineares (*strings*) [9].

A implementação de Cramer tinha todas as idéias básicas de programação genética. Em 1986, Joseph Hicklin [10] e, em 1987, Cory Fujiko [11] consideraram o uso de algoritmos genéticos para evoluir programas LISP. Na mesma época, um aluno de graduação alemão, Jurgen Schmidhuber, usou algoritmos genéticos sobre uma máquina simbólica LISP para evoluir programas. Em 1987, Schmidhuber traduziu seu sistema para Prolog e, juntamente com outros dois pesquisadores, publicou um artigo [12] sobre esse sistema. John Koza "reinventa" a programação genética em 1989, citando apenas Holland, sem dar crédito a muitos dos trabalhos anteriores.

2.2 Computação evolutiva

A programação genética é uma especialização dentro da área de computação evolutiva, que, de maneira genérica, é um paradigma de otimização e busca que utiliza como metáfora a seleção natural e a genética.

O trabalho de Holland é tido por muitos como a origem de toda a miríade da computação evolutiva. Na verdade, na década de 60, vários algoritmos usando esse paradigma foram criados quase que ao mesmo tempo. O termo "computação evolutiva" surgiu mais tarde, em uma tentativa de classificar conceitualmente a diversidade de variantes. Hoje, as principais vertentes são: algoritmos genéticos, programação evolutiva, estratégias evolutivas, programação genética e sistemas classificadores.

O funcionamento básico é o mesmo para todas as vertentes. Um grupo de soluções candidatas, denominada *população*, passa por ciclos de avaliações, seleções e alterações que fazem, assim, a exploração do espaço de busca de um problema. As soluções candidatas são denominadas indivíduos, e, em algumas variantes, cromossomos. Um indivíduo é avaliado pela sua capacidade de resolver o problema. As alterações são efetuadas por meio de modificações individuais, denominadas *mutações*, ou pela troca de material entre indivíduos, em uma analogia à reprodução sexuada, denominada *recombinação ou cruzamento*. A seleção faz uma prospecção na população pelas melhores soluções a serem modificadas. As operações de seleção, mutação e recombinação são denominadas *operadores genéticos*. Os operadores genéticos criam um novo grupo de indivíduos que irá fazer parte de uma *nova geração*. O ciclo evolutivo de gerações se repete até que um objetivo pré-estabelecido seja atingido.

2.2.1 Programação evolutiva

Esta variante de computação evolutiva foi criada por Fogel [13] em 1962. Consiste, basicamente, na evolução de máquinas de estados. Uma população de indivíduos (conjunto de máquinas de estados) é exposta a uma lista de símbolos de entrada, e cada indivíduo é avaliado pela sua capacidade de prever o próximo símbolo da lista. Quando toda a seqüência tiver sido aplicada, a aptidão de cada máquina é calculada por sua capacidade de predição. Novos indivíduos são criados por intermédio da mutação dos indivíduos da população corrente. A metade mais apta da população atual (μ) é unida à metade mais apta da população mutante (μ) para formar uma nova geração. Esse esquema de seleção é conhecido como $\mu + \mu$.

2.2.2 Estratégias evolutivas

O algoritmo de estratégias evolutivas [14] surgiu da necessidade de otimização aerodinâmica de dispositivos. Em 1964, dois estudantes da Universidade Técnica de Berlin, Rechenberg e Schwefel, tentavam, sem sucesso, usar o método do gradiente para otimizar um conjunto de parâmetros de controle de aerodinâmica em um túnel de vento. Tiveram, então, a idéia de usar uma "estratégia evolutiva" para otimizar esse conjunto de parâmetros. A implementação contou com um terceiro estudante, Bienert.

A técnica original tinha apenas dois operadores (seleção e mutação) atuando sobre apenas um indivíduo: se a mutação resulta em algo melhor, o novo indivíduo toma o lugar do "pai". A abrangência da mutação tem distribuição normal: mutações pequenas têm maior probabilidade de ocorrer do que mutações grandes. O indivíduo leva dois tipos de variáveis: as de objetivo e as estratégicas. As variáveis de objetivo são formadas pelos parâmetros a serem otimizados, definindo a solução. As estratégicas são variância e co-variância para mutação. Como as variáveis estratégicas também são passíveis de evolução, o sistema adapta-se sozinho. Esta primeira abordagem ficou conhecida como estratégia 1+1.

Mais tarde, surge uma nova estratégia, a $\mu + 1$, em que uma população de μ indivíduos sofre recombinação aleatória para formar um único indivíduo que sofre também mutação; o resultante substitui o pior indivíduo de μ, se for melhor do que este. Esta estratégia deu lugar às estratégias $(\mu + \lambda)$ e (μ, λ). Em $(\mu + \lambda)$, μ indivíduos da população ascendente se juntam a λ indivíduos da descendente para seleção da próxima geração. Já na estratégia (μ, λ), apenas os λ descendentes sobrevivem.

2.2.3 Algoritmos genéticos

O trabalho de Holland [4] é considerado um dos precursores da computação evolutiva. Seu trabalho com algoritmos genéticos começa no início da década de 1960, com o intuito original de entender os princípios dos sistemas adaptativos.

Enquanto a programação evolutiva e as estratégias evolutivas foram criadas com um único objetivo (otimização), a abordagem de algoritmos genéticos se difere por ser mais abrangente, com um enfoque em aprendizagem de máquina, o que fica evidente pelo seu uso em sistemas classificadores.

O conceito principal dos algoritmos genéticos é, basicamente, o mesmo: evoluir um conjunto de soluções candidatas (cromossomos), mediante a aplicação repetida de operadores genéticos. Na proposta original, os cromossomos são representados por seqüências de tamanho fixo de valores binários. Existem, ainda, implementações que usam números reais. Os operadores genéticos básicos são: recombinação, mutação e auto-inversão.

A recombinação de dois indivíduos é feita pelo emparelhamento dos cromossomos e pela troca de segmentos de mesmo tamanho e lugar. Os segmentos são definidos pela escolha aleatória de um ou mais pontos de corte. O mais comum é a escolha de recombinação em um ponto (proposta original de Holland) ou dois pontos; neste caso, os segmentos de permuta são os delimitados entre os dois pontos de corte. A mutação é feita pela alteração de um ou mais elementos dos cromossomos; no caso de valores binários, a troca de zero para um e vice-versa. O operador de inversão atua sobre um indivíduo, dividindo-o em um ponto aleatório e invertendo a posição dos segmentos resultantes. A função deste último operador não é muito bem explicada e ele raramente é usado.

A adequação das soluções ao problema é designada por meio de uma função de aptidão geralmente específica ao tipo de problema a ser resolvido. Os operadores genéticos são aplicados à população corrente para formar uma nova população. A escolha de um indivíduo para reprodução pelos operadores é feita aleatoriamente, mas com uma probabilidade proporcional à sua aptidão. A nova população toma o lugar da antiga e o ciclo se repete, até uma condição de parada ser alcançada.

2.2.4 Sistemas classificadores

Os sistemas classificadores [4] usam algoritmos genéticos para descobrir regras para a solução de um problema. Nessa vertente, a população é formada por um conjunto de regras no

formato:

"Se *condição* então *ação*"

Essas regras, também chamadas de regras de produção ou classificadores, são codificadas por meio de seqüências binárias, que passam pelo mesmo processo evolutivo de algoritmos genéticos. A função de aptidão define o desempenho dos classificadores. Há duas abordagens quanto à maneira de fazer a avaliação: a primeira avalia o desempenho da população como um todo, isto é, todo o conjunto de regras é a solução para resolver o problema e a segunda define indivíduos como sendo subconjuntos de regras e aplica a avaliação individualmente.

2.3 Formalização da computação evolutiva

Schwefel e Bäck [15] propõem uma formalização do algoritmo geral da computação evolutiva. Esta formalização envolve apenas computação evolutiva para otimização de parâmetros, isto é, programação evolutiva, estratégias evolutivas e algoritmos genéticos. Apesar disso, a formalização é genérica o bastante para englobar programação genética com pequenas alterações.

Dadas as seguintes definições:

- I representa o espaço de busca (espaço dos indivíduos);
- $\alpha(t) \in I$ representa um indivíduo na geração t;
- $\vec{x} \in R_n$ representa um ponto no espaço de soluções;
- μ e λ são respectivamente os tamanhos da população ancestral e da população descendente,

podem ser definidas as seguintes funções:

- $f : R^n \rightarrow R$: função objetivo a ser otimizada;
- $\phi : I \rightarrow R$: função de avaliação.

Na maioria das vezes, a função objetivo é uma função a ser maximizada (ou, de forma equivalente, minimizada). O vetor $\vec{x}$ é conjunto de valores para os parâmetros livres da função f e o objetivo da otimização é encontrar um vetor $\vec{x}^*$ tal que

$\forall \vec{x} \in R_n, f(\vec{x}) \leq f(\vec{x}^*) = f^*$.

A função f e a função ϕ são mapeamentos de domínios diferentes, pois um indivíduo (genótipo) representa uma solução (fenótipo) indiretamente através de uma função de codificação. Por exemplo, em algoritmos genéticos, o cromossomo pode ter uma codificação binária e ser tratado pelos operadores de busca como uma seqüência indistinta de zeros e uns, mas ser mapeado para um vetor de parâmetros em um subdomínio dos números reais para fins de avaliação da solução (prospecção). Portanto, necessariamente, a função f é componente de ϕ pois cada indivíduo $\alpha(t)$ representa uma solução $\vec{x}(t)$ através de um mapeamento Γ:

$$\vec{x}(t) = \Gamma(\alpha(t))$$

e para a população $P(t) = \{\alpha_1(t), ..., \alpha_\mu(t)\}$, cada indivíduo *i* otimiza a função f com qualidade dada por $\phi(\alpha_i(t))$. Por exemplo, no caso de maximização ou minimização de uma função f em algoritmos genéticos, temos:

$$\phi(\alpha_i(t)) = \delta(f(\Gamma(\alpha_i(t)))) = \delta(f(\vec{x}_i(t))),$$

onde δ assegura que valores de aptidão maiores denotem indivíduos melhores — em um problema de minimização, a aptidão, na maior parte das vezes, é inversamente proporcional ao valor de $f(\vec{x})$.

Sendo θ_s, θ_r e θ_m os parâmetros para controle dos operadores genéticos, estes poderiam ser definidos como:

- Seleção: $s_{\theta_s} : (I^\lambda \cup I^{\lambda+\mu}) \rightarrow I^\mu$;
- Recombinação: $r_{\theta_r} : I^\mu \rightarrow I^\lambda$;
- Mutação: $m_{\theta_m} : I^\lambda \rightarrow I^\lambda$.

Os operadores genéticos são funções que operam sobre indivíduos, mas, sem perda de generalidade, pode-se defini-los como macro-operadores, isto é, como funções sobre populações. Portanto, a seleção escolhe indivíduos na população ascendente (I^λ), ou nas populações ascendente e descendente ($I^{\lambda+\mu}$), reduzindo a população novamente ao tamanho μ. A recombinação cria uma população de tamanho λ (I^λ) a partir de uma população de tamanho μ (I^μ). A mutação altera uma população I^μ gerada pela recombinação sem alterar seu tamanho.

O critério de parada é uma função booleana definida por:

$h : I^\mu \rightarrow \{V, F\}$.

onde V e F são valores verdade.

Finalmente, o algoritmo geral de computação evolutiva poderia ser dado por:

1. $t \leftarrow 0$
2. $P(0) \leftarrow \{\alpha_1(0), ..., \alpha_\mu(0)\} \in I^\mu$
3. Avaliar $P(0)$: $\{\phi(\alpha_1(0)), ..., \phi(\alpha_\mu(0))\}$
4. Enquanto $h(P(t)) \neq V$ faça

   $P'(t) \leftarrow r_{\theta_r}(P(t))$ (Recombinação)

   $P''(t) \leftarrow m_{\theta_m}(P'(t))$ (Mutação)

   Avaliar $P''(t) : \{\phi(\alpha_1(t)), ..., \phi(\alpha_\lambda(t))\}$

   $P(t+1) \leftarrow s_{\theta_s}(P''(t) \cup Q)$ (Seleção)

   $t \leftarrow t+1$

onde a população Q representa os indivíduos da população ascendente escolhidos para passar pelo operador de seleção. Se Q for vazio, a população ascendente não sobrevive à passagem de geração. Alguns autores [16] dividem a seleção em dois tipos: para sobrevivência e para recombinação. O primeiro tipo escolhe os indivíduos que passarão de uma geração a outra, o segundo seleciona os pares de casamento para a operação de recombinação. Na definição original de Schwefel e Bäck não há essa distinção, mas em [17], um trabalho posterior em que Bäck é um dos autores, a divisão aparece na forma de um operador “parent_selection” que escolhe os pares para casamento. Essa distinção raramente acontece na prática, pois na maioria dos programas de computação evolutiva os dois tipos de seleção são agregados em uma única operação: ao escolher indivíduos para sobrevivência, o operador de seleção cria, implicitamente, uma seqüência aleatória de pares para a atuação do operador de recombinação (seleção de pais).

2.4 Comparação com algoritmos genéticos

O algoritmo fundamental de algoritmos genéticos é, basicamente, o mesmo da proposta original de programação genética. Isso fica claro ao se compararem os fluxogramas das figuras 2.1 e 2.2 retirados do livro do próprio autor do nome e proprietário da marca “programação genética” [3]. Os fluxogramas são mostrados aqui menos como ferramenta de engenharia de

software ultrapassada do que como modo de evidenciar essas semelhanças. Além do mais, o funcionamento dos dois não se diferencia muito do que é mostrado na seção 2.3. Ainda assim, serão utilizados aqui para explicar a execução tanto de algoritmos genéticos quanto de programação genética.

A figura 2.1 mostra um fluxograma convencional de algoritmos genéticos. O sistema permite que se escolha um número N de execuções. A adição desse laço externo de repetição é necessária, porque, freqüentemente, é preciso efetuar várias execuções independentes do algoritmo base para se obterem resultados satisfatórios.

Cada execução cria uma população inicial e a evolui em gerações até que um critério de parada seja atingido. O parâmetro M no fluxograma representa o tamanho escolhido para a população (número máximo de indivíduos). Os operadores de reprodução[1], recombinação e mutação são escolhidos de acordo com probabilidades P_r, P_c e P_m, respectivamente. É interessante frisar que essas probabilidades somam 1 no algoritmo, isto é, essas operações não ocorrem em série. Há implementações, tanto de algoritmos genéticos quanto de programação genética, em que a mutação ocorre em série com as outras duas — o algoritmo na seção 2.3 ilustra este fato.

O operador de reprodução simplesmente faz uma cópia do indivíduo ascendente. O operador de seleção fica em série com os demais e escolhe um ou dois indivíduos dependendo do operador genético escolhido. O operador de recombinação precisa de dois ascendentes (pais) e gera dois descendentes (filhos) — por isso, a variável que controla o tamanho da nova população (i) é incrementada duas vezes nessa operação. Os outros dois operadores (mutação e reprodução) necessitam de apenas um ascendente.

A figura 2.2 mostra o fluxograma de programação genética. Na proposta original de Koza [18], a diferença consistia em não utilizar o operador de mutação. A partir do seu terceiro livro [3], o autor inclui a mutação como operador padrão e o algoritmo ficaria igual ao de algoritmos genéticos, se não fosse a inclusão de uma nova classe: os operadores de alteração de arquitetura.

2.5 O algoritmo de programação genética

Seria um desrespeito à inteligência média tentar explicar o algoritmo de programação genética, pois ele é igual ao de algoritmos genéticos. A diferença de programação genética não reside

[1] A escolha deste operador em vez do operador de inversão é uma questão menor, dependente do tipo de implementação

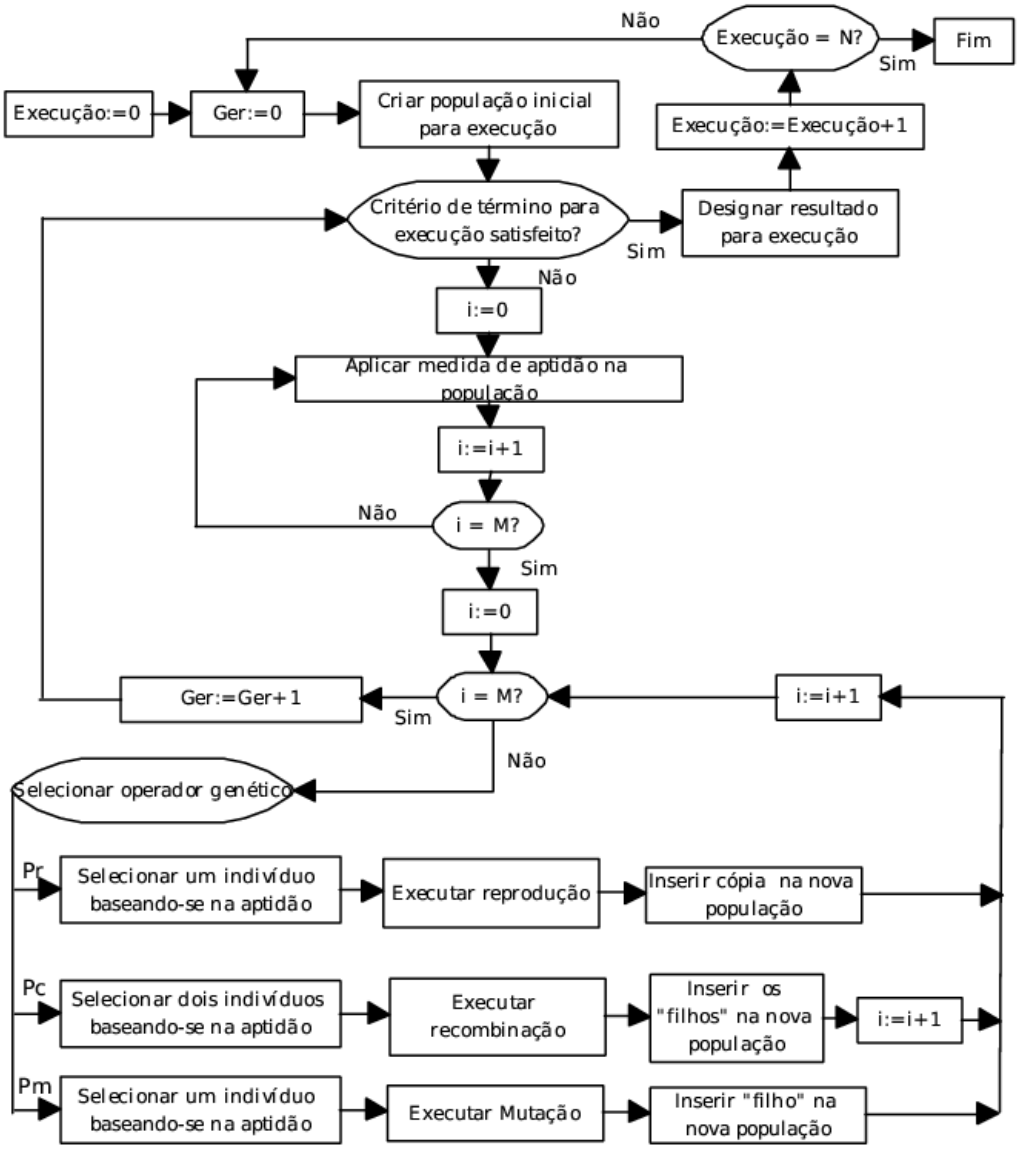


Figura 2.1: Fluxograma de algoritmos genéticos.

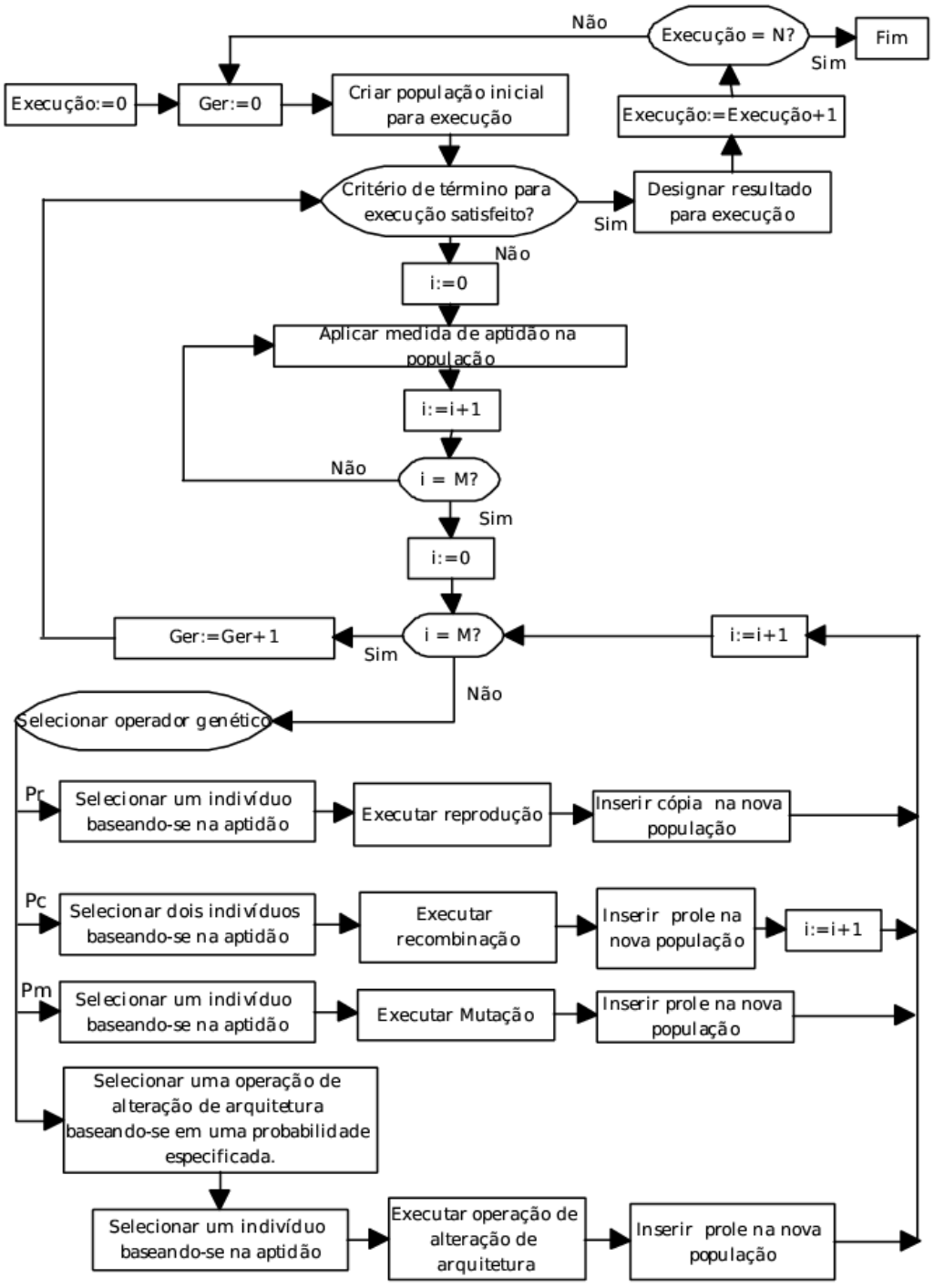


Figura 2.2: Fluxograma de programação genética.

propriamente no algoritmo, mas nos indivíduos. Aqui, os indivíduos não são mais seqüências de valores numéricos a otimizar, mas programas de computador. Esses programas são avaliados pela sua capacidade de desempenhar uma tarefa ou resolver um problema de maneira computacional.

Todavia, a simples mudança no tipo do indivíduo complica bastante a implementação no que diz respeito à representação dos indivíduos e do funcionamento dos operadores genéticos. Formas eficientes de representação dos programas são importantes para uma avaliação rápida, pois avaliar significa executar o programa para comparar resultados e executar significa interpretar ou compilar grupos de instruções desses programas, uma tarefa complicada, que torna o sistema lento. Também aqui, os operadores genéticos servem para efetuar alterações na população, porém estas alterações têm que tomar os devidos cuidados para não gerar indivíduos sintaticamente incorretos. Um desses cuidados diz respeito ao número de argumentos ao se trocar uma instrução por outra; outro é que instruções que recebem resultados de outras instruções devem possuir tipos de dados compatíveis. Koza resolve este último exigindo que o conjunto de operações (instruções) usadas para construção dos indivíduos tenha a propriedade de fechamento.

Além disso, a proposta original de programação genética é a de gerar programas com certa complexidade. A proposta sugere que os programas gerados tenham estruturas de controle, laços de repetição, chamadas recursivas e façam reaproveitamento de código com o uso de sub-rotinas.

As sub-rotinas, na proposta de Koza, são implementadas por meio do que é chamado de *funções definidas automaticamente*. Essas funções são criadas, assim como em qualquer programa comum, em uma região à parte do programa principal no indivíduo. Para facilitar a compatibilidade na manipulação pelos operadores, os indivíduos carregam informações sobre a quantidade de argumentos de cada uma das suas sub-rotinas. Essas informações correspondem à arquitetura do programa. Aí é que entra a necessidade das operações de alteração de arquitetura.

Quando se tenta descobrir o programa que melhor resolva um problema, é bastante vantajoso saber com antecedência quais os operadores e sub-rotinas mais adequados para a montagem da solução. Isso pode ser feito com base na experiência do usuário no domínio do problema. Por exemplo, um problema de seis variáveis pode ter um comportamento mais ou menos conhecido como dois subsistemas de três variáveis. Informar ao sistema que é interessante criar funções de três variáveis pode ajudar na diminuição da complexidade do problema. No entanto, isso corresponde a informar ao sistema *como* resolver o problema e não *o que* resolver. Esse tipo de

forma de trabalhar, segundo Koza, iria contra o que se esperaria de um sistema de programação genética, no que ele entende por sistema "inteligente". O sistema deveria ser capaz de induzir por si mesmo o formato adequado das subestruturas a serem utilizadas para um problema. Os operadores de alteração de arquitetura têm, então, a função de encontrar de forma autônoma arquiteturas mais adequadas para a resolução de problemas. A implementação desses operadores e da estrutura para funções automaticamente definidas é complicada para a representação tradicional de indivíduos e sobrecarrega bastante o processamento. Por isso, o uso dessas duas características foi descartado nas implementações feitas aqui. Uma implementação alternativa para a idéia será discutida na seção 3.8.

2.6 Representação dos indivíduos

Existem várias formas de representação para os indivíduos em programação genética, que serão discutidas ao longo do texto, mas, para efeito de explanação, a representação tradicional é, com certeza, a mais didática — além de ser a implementação mais usual. A implementação de Koza representa os indivíduos na forma de árvores sintáticas. Essa representação é bastante facilitada pela escolha que fez da linguagem de implementação: LISP. As expressões simbólicas do LISP (*S-Expressions*) são árvores sintáticas prontas.

Uma sub-rotina que em linguagem Pascal seria escrita como:

```
function DivP(var arg1, arg2: Real ):Real;
begin
  if (arg2 <> 0)
    then result:= arg1/arg2
    else result:= 0;
end
```

ficaria em LISP como :

```
(defun DivP
  (ARG1 ARG2)
  (values
    (if (/= ARG2 0)
        (/ ARG1 ARG2)
        0
```

```
    )
  )
)
```

A representação em árvore é mostrada na figura 2.3. Os nós terminais da árvore são compostos por variáveis, constantes ou funções com aridade zero. Os nós não-terminais são, para linguagens do paradigma funcional, simplesmente funções, que podem ser operadores matemáticos ou booleanos, funções matemáticas, instruções de controle de fluxo, instruções de iteração, funções de recursão ou sub-rotinas predefinidas.

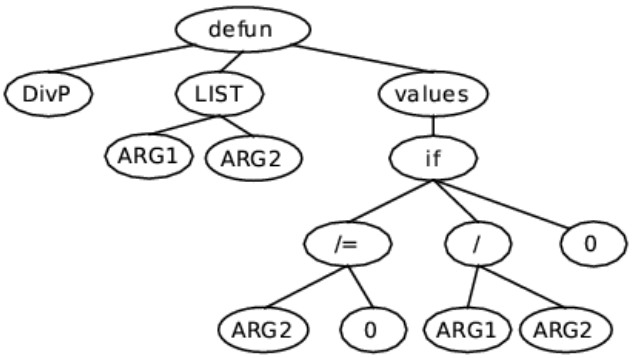


Figura 2.3: Árvore sintática de função LISP.

No início do processamento, os indivíduos são criados por meio da escolha aleatória de terminais ou não-terminais tomados a partir de conjuntos previamente especificados e compatíveis com o tipo do problema. O processo de criação é explicado a seguir.

2.7 Criação da população inicial

Bons algoritmos para criação de indivíduos são essenciais para gerar diversidade genética no início do processamento. Dentre os métodos mais usuais para geração de árvores sintáticas em programação genética, estão *grow*, *full* e *ramped half-and-half*.

O método *grow* cresce árvores de maneira a escolher aleatoriamente entre terminais e não-terminais até uma profundidade máxima especificada, onde escolhe apenas do conjunto de terminais fechando o ramo. Portanto, um determinado ramo termina quando atinge uma profundidade máxima D ou quando um terminal é escolhido. Esse método cria árvores irregulares.

O método *full* escolhe aleatoriamente apenas do conjunto de não-terminais até uma profundidade "$D-1$" e então escolhe entre os terminais. Isso força todos os ramos da árvore criada a

terem uma profundidade *D*, gerando árvores uniformes.

O método mais utilizado pela comunidade de programação genética é o *ramped half-and-half*. Esse método procura criar diversidade mediante dois artifícios:

- Gera subpopulações com altura variável, incrementando o parâmetro de profundidade *D* de 2 a um valor máximo D_{max};
- Para cada uma dessas subpopulações, metade é gerada usando o método *full* e a outra metade o método *grow*.

Um parâmetro que também pode ser especificado para o algoritmo de criação é o número máximo de nós. Outra possibilidade em *ramped half-and-half* é permitir que se especifique a porcentagem de árvores *full* e *grow*.

Sean Luke [19] propõe métodos alternativos, introduzindo dois novos algoritmos (PCT1 e PCT2), que dão maior controle sobre o tamanho e formato das árvores e também sobre a probabilidade da ocorrência de terminais e não-terminais. Um estudo sobre a evolução de tamanho e formato dos indivíduos durante a evolução foi feito por Langdom [20] e uma visão geral sobre os diferentes métodos de criação de árvores pode ser encontrada em [21].

2.8 Operadores

A implementação dos operadores genéticos é bastante dependente do tipo de representação dos indivíduos da população. Mesmo em algoritmos genéticos, nos quais a implementação mais básica usa seqüências de valores binários como cromossomos, os operadores podem estar sujeitos a restrições de acordo com o tipo de problema e a necessidade de representação decorrente. Na programação genética tradicional, a representação dos programas em formato de árvores e a necessidade de fechamento entre as operações fazem com que os operadores tenham uma complexidade maior em relação às outras modalidades de computação evolutiva.

Podem-se dividir os operadores em duas categorias: operadores de exploração (ou busca) e de prospecção [2]. Na primeira categoria estão os operadores de mutação e recombinação. Estes são responsáveis por imprimir diversidade à população, explorando o espaço de busca. Na segunda categoria está o operador de seleção. Ele tem o encargo de extrair da população o melhor aproveitamento possível, de forma que se mantenha tanto a qualidade das soluções (indivíduos)

[2] o termo em Inglês é *exploitation*, significando exploração no sentido de aproveitamento máximo de um recurso.

quanto a diversidade genética do grupo. A monopolização da seleção por um pequeno número de indivíduos altamente aptos deixaria pouca escolha para manipulação pelas operações de busca, levando rapidamente a um máximo local. O máximo local é a melhor aproximação dentro de uma região restrita do espaço de busca. Por ser o ponto mais alto nessa região de busca, algoritmos com tendência a privilegiar o maior valor podem ficar presos nessa região. O correto equilíbrio entre as duas categorias de operadores é essencial para uma convergência adequada.

2.8.1 Mutação

A operação de mutação em programação genética pode ter duas modalidades: mutação pontual e macro mutação.

No caso da mutação pontual, um nó da árvore é escolhido aleatoriamente para mutação e o elemento é, então, trocado por outro disponível nos conjuntos de terminais e não-terminais permitidos. Dependendo do tipo de representação dos indivíduos, como na representação em árvore, essa operação deve levar em conta as aridades dos elementos escolhidos para mutação. Trocar o elemento mutante por outro com aridade diferente acarretaria em um erro de sintaxe. A figura 2.4 mostra esse tipo de mutação.

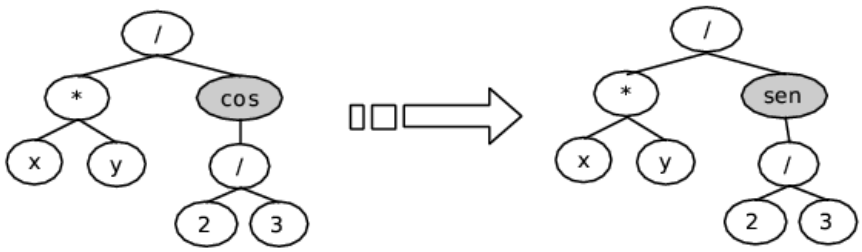


Figura 2.4: Mutação pontual de uma árvore cuja função é $f(x,y) = xy/cos(2/3)$: o ponto escolhido para mutação na árvore à esquerda está em destaque (*co-seno*); na árvore à direita, essa função é trocada pela *seno*.

A macro mutação começa, como no caso anterior, com a escolha de um nó da árvore do programa. Este nó, e todo ramo subjacente (se existir) é removido e, em seu lugar, é inserido um novo ramo, criado pela mesma função responsável pela geração dos indivíduos da população inicial. Um fenômeno indesejado e freqüente durante o intercurso de evolução de programação genética é um crescimento excessivo dos indivíduos sem melhora na aptidão conhecido como

"inchaço" [3]. Segundo Luke [21], algoritmos de criação de indivíduos com maior controle sobre o tamanho e o formato das árvores poderiam combater o fenômeno de inchaço durante a macro mutação. A figura 2.5 mostra um caso de macro mutação.

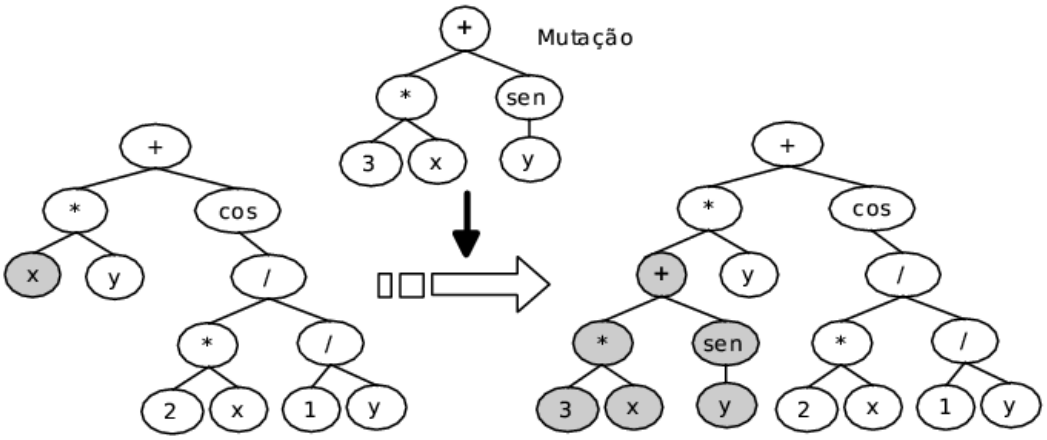


Figura 2.5: Macro mutação de uma árvore cuja expressão correspondente é $xy + cos(2x/(1/y))$: o ponto escolhido para mutação na árvore à esquerda está em destaque; na árvore resultante, à direita, esse ramo é substituído por uma árvore criada aleatoriamente $(3x + sen(y))$.

O papel da mutação na programação genética sempre foi motivo para discussões. A proposta original não fazia uso desse operador [22]. A orientação comumente aceita é que a mutação em programação genética deve ser usada de maneira limitada. Mas a probabilidade com que é aplicada depende bastante da aplicação.

2.8.2 Recombinação ou cruzamento

Recombinação é a troca de material genético entre dois cromossomos. Em programação genética isso significa a troca de partes de código entre dois programas, resultando em dois indivíduos descendentes provavelmente distintos dos pais. Na representação em árvore, escolhem-se aleatoriamente nós para permuta nos dois pais e os ramos subjacentes aos nós escolhidos são permutados. A figura 2.6 ilustra um exemplo de recombinação.

A recombinação é, ao mesmo tempo, promotora de grandes saltos no espaço de busca, e, por isso mesmo, também uma operação destrutiva. Dependendo dos pontos escolhidos para recombinação em cada pai, a recombinação pode destruir totalmente bons candidatos, quebrando os chamados "blocos construtores". Segundo Langdon [23]:

[3] Em inglês, *bloat*.

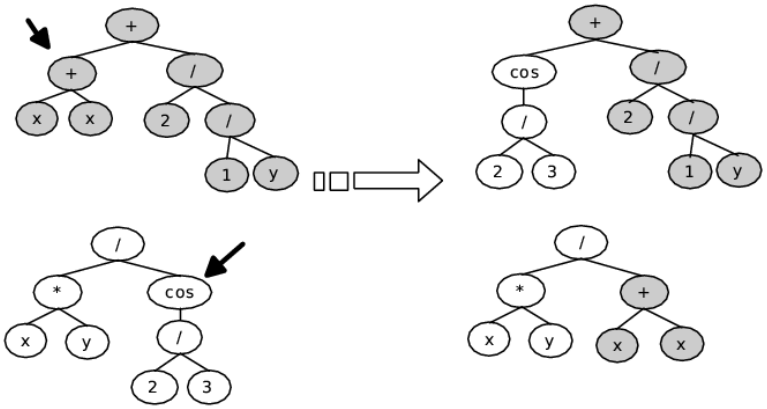


Figura 2.6: Recombinação: a árvore cinza e a árvore branca, ambas à esquerda, permutam os ramos cujos nós raiz são indicados pelas setas.

> "Bloco construtor é um padrão de genes em uma seção contígua de um cromossomo que, se presente, confere uma alta aptidão ao indivíduo. De acordo com a hipótese de blocos construtores, uma solução completa pode ser construída através da operação de recombinação pela junção de vários desses blocos espalhados pela população em um único indivíduo."

Alega-se que essa característica destrutiva da recombinação seja a causa do fenômeno de inchaço pela proliferação de íntrons. Em Genética, íntrons são partes do cromossomo que não carregam informação genética válida, isto é, não geram proteínas. Em programação genética, íntrons são partes do código redundantes para efeito da avaliação do programa, como somas com zero ou multiplicações por um. Exemplos de íntrons são indicados na figura 2.7.

Há propostas bastante conflitantes quanto ao tratamento dado a íntrons. Ekart [24] parte do pressuposto de que os íntrons e código ineficiente significam processamento desnecessário durante a fase de avaliação dos programas e propõe um operador de simplificação, que, se aplicado na proporção certa durante o ciclo de gerações, diminuiria a quantidade de código redundante. Ekart propõe dois parâmetros para a simplificação: a freqüência, que define de quantas em quantas gerações deve ser executada, e a probabilidade, que define sua probabilidade de aplicação quando executada. Nordin [25] e Angeline [26] utilizam uma abordagem preventiva. Partindo-se do fato de que ramos não codificantes são bons pontos para operar a poda para recombinação, propõem a inserção proposital de íntrons no código de programas, com o objetivo de diminuir a possibilidade de destruição de blocos construtores durante a recombinação. Sugere-se, ainda, um meio de se demarcar bons pontos de poda pela atribuição de pesos aos nós. Os pesos evoluem juntamente com os indivíduos, de forma que acabam por identificar implicitamente íntrons e blocos construtores. Esses pesos influenciam as escolhas

para poda pelo operador de recombinação.

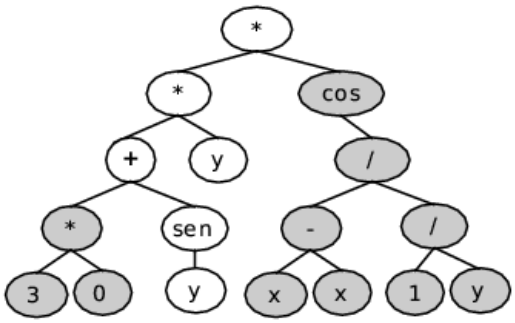


Figura 2.7: Íntrons: os ramos em destaque são íntrons. O ramo da esquerda por ser componente em uma soma e ter valor zero, o ramo da direita por fazer parte de uma multiplicação e ter valor um. É fácil perceber que a função pode ser simplificada para $sen(y) * y$.

O operador de recombinação é considerado o de maior importância para a programação genética tradicional. Nas primeiras propostas do algoritmo, esse operador era a única fonte de variabilidade. Apesar disso, há muita controvérsia a respeito do seu desempenho frente ao operador de mutação.

2.8.3 Outros operadores genéticos

Os operadores de mutação e recombinação são os mais comumente utilizados em programação genética. Existe, porém, uma série de pequenas variações para adequar-se a representações distintas e problemas específicos.

O operador de *reprodução* efetua a simples cópia do indivíduo para a próxima geração. A nomenclatura de Koza pode gerar um pouco de confusão pois *reproduction*, em Inglês, também significa reprodução sexuada. Outros autores denominam-no como operador de *replicação*.

O operador de *edição* ou *simplificação* procura eliminar código ineficiente, como expressões do tipo "$x+x+x+\ldots$", e íntrons. Como simplificação eficiente pode ser uma operação bastante complicada, exigindo capacidades de processamento simbólico, e com alto custo de processamento, esses operadores acabam por não serem muito elaborados. E mesmo simplificados, continuam dispendiosos, devendo ser aplicados a intervalos espaçados entre as gerações e com probabilidade baixa [24];

O operador de permutação escolhe um nó não terminal da árvore-programa e permuta seus ramos. Já o operador de destruição elimina os indivíduos menos aptos, geralmente usado nas

primeiras gerações para efetuar uma “limpeza”.

2.9 Avaliação e aptidão

Na natureza, a medida para a sobrevivência de um indivíduo e a perpetuação dos seus genes é sua capacidade de adaptação ao ambiente. Em programação genética, essa medida é dada pela capacidade do programa simular um comportamento. Geralmente, o comportamento a ser seguido é representado por um conjunto de valores de entrada e saída, ou dados de treinamento. Esses dados são conhecidos como casos de aptidão (*fitness cases*). Para se obter a aptidão de um programa são necessários dois passos:

- executar o programa para os valores de entrada informados nos casos de aptidão, obtendo uma série de valores de saída;
- comparar os valores de saída do programa com as saídas esperadas, informadas pelos casos de aptidão. Quanto menor a diferença absoluta entre os valores, maior a aptidão do indivíduo.

A função responsável pelo último passo é chamada função de aptidão ou função de *fitness*. O termo “avaliar o programa” é freqüentemente usado para descrever o primeiro passo (de execução), o que pode gerar alguma confusão.

A forma mais direta de se estabelecer o valor de aptidão é pela quantificação de erro ou acerto do indivíduo. Esse método para cálculo de aptidão é chamado *aptidão bruta* (*raw fitness*). Um exemplo de função de aptidão bruta é a equação de erro absoluto, geralmente usada para regressão simbólica:

$$f_i = \sum_{j=1}^{N_c} |p_{ij} - s_j|, \tag{2.1}$$

onde f_i é o valor de aptidão do indivíduo i, N_c é o número de casos de avaliação, p_{ij} é a saída do programa i para o caso j e s_j é a saída do caso de avaliação j.

Outra medida de avaliação freqüentemente usada é o erro quadrático total. O erro quadrático tem a propriedade de enfatizar as diferenças:

$$f_i = \sum_{j=1}^{N_c} (p_{ij} - s_j)^2 \tag{2.2}$$

A função de aptidão bruta é totalmente dependente do problema. Por exemplo, em um problema que envolve equações de lógica booleana, uma função de aptidão bruta a considerar seria o número de acertos na tabela verdade. No caso anterior, usando erro absoluto (equação 2.1), quanto menor o valor de aptidão, melhor o indivíduo; neste caso, acontece o contrário. Geralmente o valor fornecido pela função de aptidão bruta é trabalhado posteriormente para obtenção de algum comportamento específico na fase de seleção. Por isso, é interessante adotar um padrão único para a aptidão de forma a tornar o sistema independente do problema. *Aptidão padronizada* é o nome que se dá ao valor adaptado. Usualmente isso é feito por meio de uma função que transforma os valores de aptidão bruta em valores entre zero e infinito, em que a melhor aptidão é zero.

A *aptidão ajustada* para o indivíduo i, a_i, é obtida a partir do valor de aptidão padronizado s_i pela fórmula:

$$a_i = \frac{1}{1+s_i} \tag{2.3}$$

Os valores assim obtidos variam de zero a um, onde um é o melhor valor de aptidão e zero, o pior. Essa equação tem a propriedade de enfatizar as diferenças conforme os valores de s_i se aproximam de zero.

A *aptidão normalizada* para um indivíduo i, n_i, é calculada a partir da aptidão ajustada, a_i, pela equação:

$$n_i = \frac{a_i}{\sum_{j=1}^{N_p} a_j}, \tag{2.4}$$

onde N_p é o número de indivíduos na população.

2.10 Seleção

Não existe uma forma de seleção específica para programação genética. As mais usadas são as seleções em roleta e em torneio. O método da roleta consiste em uma analogia com sorteio em uma roleta de cassino. A roleta é dividida em fatias proporcionais à aptidão de cada indivíduo da população. Indivíduos com maior aptidão recebem "fatias maiores" do círculo e têm maior probabilidade de serem sorteados. Analogamente, indivíduos menos aptos têm menor chance de ganhar o sorteio. Os valores da roleta somam um e são obtidos pela função de aptidão normalizada dada pela equação 2.4. O método da roleta também é conhecido como *método de seleção proporcional*.

O método do torneio vem de outra analogia, desta vez com torneios de competição. Grupos de n indivíduos são sorteados na população e o mais apto vence o torneio. O método do torneio apresenta algumas vantagens em relação ao da roleta. Para "montar" a roleta, é preciso avaliar todos os indivíduos, ao passo que no método do torneio essa avaliação pode ser feita sob demanda.

Há um fator importante na seleção em computação evolutiva, chamado *pressão da escolha*. Em uma seleção com alta pressão de escolha, os indivíduos mais aptos têm maior probabilidade de serem escolhidos, e com baixa pressão de escolha, a competição é menos desigual. Maior pressão de seleção significa mais prospecção e menor pressão de seleção, mais exploração. Na ausência absoluta de pressão, a busca se torna um "passeio ao acaso". Controlar a pressão da escolha usando a roleta pode ser uma questão trabalhosa, enquanto, no torneio, isso é feito mudando-se o valor de n.

Em populações muito grandes é interessante usar a chamada *seleção por truncamento* [27]. É estabelecido um limite de aptidão e indivíduos abaixo desse limite são descartados, então a seleção é feita apenas entre os indivíduos restantes.

A seleção proporcional tem a desvantagem de supervalorizar indivíduos muito aptos ou descartar prontamente aqueles com baixo valor de aptidão. Isso pode acarretar diminuição na diversidade genética e convergência precoce a um máximo local. Para contornar o problema, Baker [28] propôs um método chamado seleção por nivelamento linear. Neste método, os indivíduos são ordenados pelo valor de aptidão, formando uma seqüência "$p_1, \ldots, p_i, \ldots, p_n$". A posição 1 contém o pior indivíduo e a posição n o melhor. O valor de aptidão é, então, ponderado como uma função linear de sua posição i na seqüência.

Outro método de seleção proposto por Baker é a seleção por nivelamento exponencial [28]. A única diferença do método anterior é que se utiliza uma função exponencial em vez da função linear para adaptar os valores de aptidão.

2.11 Algoritmo *steady state* versus geracional

O algoritmo geracional foi apresentado nas seções 2.3 e 2.4. Neste algoritmo, há uma passagem de gerações bem definida com a nova população substituindo totalmente a antiga. Uma alternativa é o *steady state*, que se difere do anterior por seu método de substituição dos indivíduos a cada ciclo de busca.

No *steady state* tradicional, N indivíduos da população são escolhidos aleatoriamente e

passam por uma seleção do tipo torneio; os selecionados passam aos operadores de busca e a prole resultante substitui alguns dos perdedores do torneio.

Outra modalidade de *steady state* consiste em selecionar N indivíduos da população, aplicar as operações genéticas de busca a estes e substituir os piores indivíduos da população pela prole resultante.

O nome vem do fato de que a população não se move de uma geração a outra, os indivíduos é que morrem ou nascem (substituem outros ou são substituídos por outros). Não existe a passagem de geração, os indivíduos são *mantidos* durante os ciclos de seleção e busca até que outros melhores sejam encontrados. Isso garante que a aptidão entre os ciclos seja sempre crescente.

2.12 Elitismo

O fato de um indivíduo ter aptidão alta não significa que será escolhido pelo passo de seleção. Mesmo depois de ser escolhido, os operadores genéticos de busca podem piorar sua condição, ao invés de melhorá-la. Em muitos tipos de problema, é interessante que os melhores indivíduos sobrevivam de uma geração à outra. O elitismo é uma forma de preservar os melhores indivíduos durante o intercurso das gerações, de forma que a evolução não "regrida" de uma geração a outra. O mecanismo é simples: em uma população de tamanho M, depois da operação de avaliação, os N melhores indivíduos são automaticamente copiados para a próxima geração, deixando ao encargo da operação de seleção escolher os outros $M - N$ indivíduos. O elitismo deve ser usado com parcimônia, pois pode induzir a uma convergência precoce a um máximo local.

2.13 Critérios de término

A escolha do critério de término é outro fator dependente do problema. O mais comum é fazer o programa executar por um *número máximo de gerações* e por certo *número de execuções independentes* [18]. Mas esse tipo de abordagem pode interromper uma evolução promissora. Registros sobre a evolução de aptidão do melhor indivíduo e mecanismos para salvar a população final são artifícios necessários para contornar esse problema.

Uma alternativa é prosseguir até se *encontrar uma solução satisfatória*. Obviamente esse critério traz a desvantagem de ficar preso a máximos locais e geralmente é usado juntamente com o primeiro.

Com o fim específico de contornar máximos locais, Kramer [29] propõe que a evolução continue enquanto houver melhora na população. A detecção de uma estagnação no processo evolutivo pode ser feita pelo acompanhamento da convergência de aptidão do melhor indivíduo. Outra forma é pelo acompanhamento de convergência da aptidão média da população: aptidão média com valores muito próximos aos da melhor aptidão geralmente significam baixa diversidade genética. Os critérios acima podem ser usados em conjunto.

2.14 Preparação do sistema

A preparação de um sistema de programação genética é bastante voltada para o tipo de problema. Os pré-requisitos necessários para o sistema funcionar adequadamente são os seguintes:

- Casos de aptidão: geralmente no formato de uma tabela de entrada e saída, os dados devem estar em quantidade suficiente e qualidade adequada para uma boa generalização por parte do algoritmo;
- função de aptidão (também chamada função de custo): a escolha da função certa é vital para nortear a direção da busca e evitar convergência muito rápida ou muito lenta;
- conjunto de terminais e não-terminais: as funções e constantes necessárias para a construção da solução devem ser escolhidas cuidadosamente de acordo com o problema a ser resolvido;
- parâmetros: o ajuste certo de parâmetros é um trabalho de longo prazo, como se mostra a seguir.

A resolução de problemas por programação genética é bastante dependente de parametrização específica, isto é, cada situação se desenvolve melhor com determinadas configurações que devem ser descobertas ao longo de várias tentativas. Por esse motivo, os experimentos devem, necessariamente, fazer o registro dos parâmetros usados para tornar possível sua análise e reprodução.

Alguns sistemas evoluem os parâmetros juntamente com a população. Um exemplo simples disso é o das variáveis estratégicas em estratégias evolutivas (seção 2.2.2). Um exemplo em programação genética é do sistema comercial *Discipulus* [30], que usa esse tipo de abordagem e alega ter “auto-configuração inteligente, sem nenhuma necessidade de entrada de parâmetros”. Para a maioria dos sistemas, porém, a configuração dos parâmetros é feita manualmente, e alguns deles controlam:

- tamanho da população;
- critério de término: número máximo de gerações, aproximação mínima desejada, constantes para detecção de estagnação;
- taxa de aplicação de cada operador genético: recombinação, mutação, reprodução, etc.
- criação de indivíduos e mutações: no caso de representação por árvores, a profundidade máxima, número máximo de nós, e, as vezes, taxa de criação de cada terminal e não-terminal;
- criação da população inicial: algoritmo de criação (*full*, *grow*, ou *ramped*) e porcentagem de cada modo de criação (no caso de *ramped*);
- pressão seletiva: o número de indivíduos no torneio, constantes para ajuste das funções de nivelamento, tamanho da seleção (no caso do algoritmo *steady state*), limite de truncamento;
- tamanho do grupo de elite;
- simplificação: freqüência e probabilidade com que é aplicada;
- condições de chamada de destruição;

Outro parâmetro que não consta na literatura, mas é bastante usado na prática, é a semente de aleatoriedade. Para reproduzir adequadamente os resultados, é necessário que o sistema ofereça a opção de se escolher uma semente para a geração de números aleatórios.

2.15 Estruturas de dados e variantes

As principais modalidades de programação genética diferem-se mais pela estrutura de representação dos indivíduos do que propriamente pelo algoritmo. De fato, o algoritmo base é o mesmo, com idiossincrasias forçadas pela representação usada. A simples mudança de representação, porém, afeta consideravelmente o tempo de processamento e a eficiência da busca.

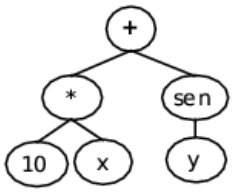
O uso de LISP como plataforma de desenvolvimento da maioria dos clássicos da área de programação genética deu lugar à linguagens como C, C++ e Java por motivo de eficiência computacional [3][4]. Para que os programas em formato de árvore, representados naturalmente no

[4]Embora a literatura da área faça esse tipo de afirmação, é bom lembrar que compiladores LISP de última geração como o *BigLoo* são muito rápidos e chegam a suplantar *Fortran* em velocidade de processamento para problemas numéricos.

LISP por expressões simbólicas, sejam implementados em outras linguagens, faz-se necessário o uso de novas estruturas de dados.

2.15.1 Estruturas em árvore

A construção de árvores de programas usando ponteiros é considerada dispendiosa em se tratando de uso de espaço, exigindo um trabalho pesado de alocação de memória [3]. A alternativa proposta é a que usa as representações pós-fixa (notação polonesa) ou pré-fixa. Um programa em notação polonesa pode ser visto na figura 2.8.



1	+
2	*
3	10
4	x
5	sen
6	y

Figura 2.8: Notação Polonesa para a função $10x + sen(x)$: à esquerda a representação em árvore correspondente.

Para esse tipo de notação, é preciso tratar o caso de funções de múltiplas aridades, criando um novo nome de função para cada aridade. Por se tratar de uma estrutura implementada em um vetor, a escolha do ponto de poda é feito em tempo constante. Uma implementação alternativa dessa representação foi implementada neste trabalho e é discutida com maiores detalhes na seção 5.5.2.

As alternativas mais usuais à estrutura de árvore são as estruturas de grafos e as estruturas de código linear.

2.15.2 Estruturas em grafos

Na estrutura em grafo, o programa é representado como um grafo direcionado. Os argumentos dos não-terminais são referenciados por ponteiros. A figura 2.9 mostra este tipo de estrutura.

A estrutura em grafo tem a propriedade de representar indivíduos complexos de maneira compacta. Exemplos de usos dessa estrutura podem ser encontrados no sistema de processamento de sinais PADO [31], cujo núcleo é um motor de programação genética, na imple-

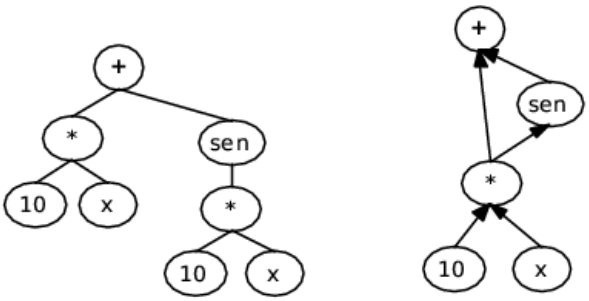


Figura 2.9: Estrutura em grafo para a função $10x + sen(10x)$: à esquerda a representação em árvore correspondente.

mentação da programação genética paralela distribuída [32] e na programação genética cartesiana [33].

2.15.3 Estruturas de código linear

As implementações fora do LISP têm a execução mais rápida e com menos gasto de memória, porém há uma contrapartida séria quanto à flexibilidade do projeto. A maioria das linguagens não possui uma maneira simples de executar compilação em tempo de execução de programas, que no LISP é efetuada pela função “eval”, o que força a construção de um interpretador que suporte todos os terminais e não-terminais requeridos. Esse interpretador é, ainda, um gargalo razoável de processamento. Por esse motivo algumas variantes de programação genética adotam a idéia de manipular de forma direta código executável. Essa é a motivação por trás dos sistemas de estrutura de código linear.

Estruturas de código linear representam os programas como um conjunto linear de instruções que são executadas seqüencialmente. Essas instruções podem ser funções ou variáveis. As funções são operações com até dois argumentos que operam apenas sobre variáveis, de maneira similar ao modo como instruções de código de máquina manipulam registradores. Um exemplo desse tipo de estrutura pode ser visto na figura 2.10. Sistemas de programação genética linear são orientados à programação imperativa, enquanto os baseados em estrutura de árvore fundamentam-se no paradigma funcional.

Algumas implementações de sistemas de programação genética linear são as mais rápidas quando se leva em conta tempo de processamento. Isso ocorre porque, quando a implementação manipula programas em linguagem de máquina, não há necessidade do uso de um interpretador interno e as instruções são passadas diretamente ao processador. Há também algumas implementações sobre código de máquinas virtuais como Java.

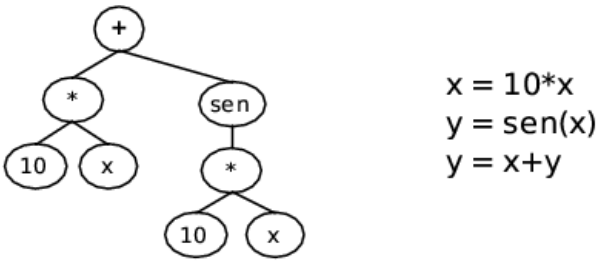


Figura 2.10: Estrutura linear para a função $10x + sen(10x)$: à esquerda a representação em árvore correspondente.

Os sistemas de codificação linear são estudados principalmente por Nordin, Banzhaf e Francone [30]. Segundo Nordin, os sistemas implementados assim têm também melhor desempenho relativo ao se comparar evolução *versus* gerações.

O uso de estruturas de dados para programação genética vai muito além e a nomenclatura para classificação de variantes é ainda bastante nova e diversa. Por exemplo, alguns autores consideram como tendo representação linear toda variante que manipula os cromossomos como segmentos lineares (de maneira semelhante a algoritmos genéticos). Além disso, existe uma série de implementações que usam estruturas híbridas, reunindo conceitos, e a classificação rígida é difícil.

2.15.4 Variantes

A programação genética é muito prolífica em se tratando de variantes. Entre as implementações variantes mais populares estão: a programação genética linear, a programação da expressão gênica, a programação por multiexpressão, a programação genética cartesiana, a programação genética sem rastro[5], a evolução gramatical e os algoritmos genéticos para derivação de *software* . Uma explicação superficial sobre cada uma pode ser vista em [34].

Além da programação genética tradicional, as variantes consideradas para implementação por suas promessas de rapidez e eficiência foram três: programação genética linear, programação por multiexpressão e programação da expressão gênica. Esta última variante foi escolhida por fatores que envolvem facilidade de implementação, boa documentação e disponibilidade de uma versão para testes. Ela será explicada com maiores detalhes no capítulo 3.

A programação genética linear é a principal representante das variantes que usam estruturas

[5]O nome em inglês é Traceless Genetic Programming

de código linear. Dos casos da literatura, parece ser a mais rápida e eficiente das variantes. Sua implementação é mais trabalhosa do que a da maioria das vertentes porque uma implementação realmente rápida envolve o uso de código de máquina.

A programação por multiexpressão usa conceitos híbridos de representação linear e grafos. Carrega promessas de ser muito rápida e eficiente. Sua implementação foi descartada apenas por uma questão de tempo.

2.16 Regressão simbólica

A regressão simbólica consiste em induzir expressões matemáticas a partir de dados de um sistema matemático através da manipulação de expressões. Esse processo também é chamado procura de funções (*function finding*). O sistema matemático é descrito por dados de entrada e saída (os casos de aptidão), que consistem em valores de uma função desconhecida f, tal que $f : R_n \rightarrow R$, ou seja, os casos de aptidão geralmente têm um formato:

$$x_1, x_2, \ldots, x_n, y$$

onde x_1 a x_n representam variáveis independentes no sistema e y, a variável dependente. Problemas de regressão simbólica são usados freqüentemente para testes de desempenho em programação genética e a literatura da área é rica em estudos de caso de busca de funções.

2.16.1 Alternativas para regressão simbólica

Existem vários outros métodos para efetuar regressão simbólica. Redes neuronais [35] são uma alternativa para modelar dados no formato entrada e saída. O problema é que, por sua natureza, é impossível extrair a função modelada da malha neuronal. Um trabalho pioneiro sobre a indução de equações sobre dados é o programa BACON [36]. As primeiras versões do programa usam heurísticas simples para construir recursivamente equações de duas variáveis: dependendo do tipo de variação que um termo tem em relação a outro, adota-se uma hipótese correspondente: a relação entre os termos é linear ($y = ax + b$), um produto ou uma divisão. Para equações de múltiplas variáveis, o artifício usado é comparar uma variável independente de cada vez com a variável dependente. Uma explicação mais detalhada pode ser vista em [37]. Essa mesma idéia é compartilhada por todos os *sistemas de descoberta de equações*, uma subárea do campo de aprendizado de máquina, e uma explanação bastante completa sobre o assunto é encontrada no trabalho de Todorovski [38, 39]. A primeira desvantagem desses métodos é

que a maioria exige heurísticas desenvolvidas especificamente para o problema, enquanto a abordagem da programação genética é completamente genérica. Outro problema é que para funcionarem adequadamente é preciso reduzir o universo de soluções provendo um conjunto de estruturas prováveis para o modelo. Essas estruturas têm a mesma função que as arquiteturas de programas em programação genética (explicadas na seção 2.4). Como não há conhecimento prévio sobre estruturas prováveis para o modelo geofísico, os sistemas de descoberta de equações não pareceram ser viáveis para o problema.

2.16.2 Criação de constantes

A criação de constantes parece ser um problema para regressão simbólica em programação genética [18]. A abordagem original de programação genética para resolver esse problema é pelo uso de um terminal especial chamado constante aleatória efêmera. Para cada constante usada nas árvores da população inicial, é gerado um número aleatório em uma faixa previamente definida. Ao longo do processo evolutivo, essas constantes são movidas entre os indivíduos pelo operador de recombinação. Outras abordagens incluem a melhora periódica de constantes por métodos de otimização como algoritmos genéticos [40]. Outros propõem métodos matemáticos como busca local gradiente para fazer a otimização [41].

Ferreira [42] afirma que a variante de programação de expressão gênica não precisa da manipulação explícita de constantes e ainda assim tem desempenho melhor do que a abordagem original. A geração de constantes para esta variante será discutida no capítulo 3.7.

2.17 Processamento paralelo e distribuído

Sabe-se que a programação genética demanda uma grande capacidade de processamento e muitas aplicações necessitam de grandes populações para convergirem satisfatoriamente. Os problemas de regressão simbólica em programação genética fazem parte dessa categoria de problemas que consomem tempo de máquina e memória. Em um exemplo “simples” tirado de [3], para verificar a capacidade do sistema, tenta-se regressão simbólica para a expressão:

$$f(x) = 2,718x^2 + 3,1416x$$

Esse problema executa em uma máquina paralela de 64 processadores, com 1500 indivíduos por processador, durante 20 gerações e atinge um resultado aproximado (não se acha especificamente a função em questão). E esse é um problema simples (*toy problem*). Em contrapartida,

computação evolutiva é bastante fácil de paralelizar. A principal tarefa consumidora de tempo de processamento é a avaliação de indivíduos. Mas essa tarefa de avaliação, dependente apenas de características do indivíduo, é algo que pode ser feito de maneira localizada.

Por esses dois motivos (alta demanda de recursos e facilidade de paralelização) programação genética e processamento paralelo têm andado juntos nas últimas décadas. Mas computação com máquinas paralelas é uma opção cara e mesmo *clusters* de processamento não são muito baratos. Existe uma solução mais recente, chamada computação em rede (*network computing*), que tem revolucionado a forma de ver processamento distribuído.

A computação em rede, também chamada *grid computing*, consiste em uma forma de processamento distribuído em que se usa o poder de processamento de computadores conectados à *Internet* para processar trabalho computacional pesado. Felizmente, certos modelos de paralelização de programação genética são bons candidatos à computação em rede.

2.17.1 Modelos de paralelização

Há diferentes modelos de paralelização para computação evolutiva e grande parte pode ser usada para programação genética. De fato, a maioria dos modelos usados vem da pesquisa em algoritmos genéticos. Entre alguns desses modelos podem-se citar [16, 43]:

Modelo de migração ou de ilhas: as populações evoluem separadamente e ocorrem migrações esporádicas entre populações. A separação entre as populações assegura diversidade genética, a migração permite saltos evolutivos.

Modelo mestre-escravo ou fazendeiro-trabalhador: o mestre é o computador responsável por fazer a seleção dos indivíduos da população e atribuir a aptidão, os computadores escravos ficam com o serviço pesado: recombinação, mutação e avaliação de funções. Um exemplo desse tipo de modelo é usado em [44].

Modelo de difusão: a população é espalhada por unidades de processamento em que cada unidade tem uma vizinhança baseada em uma topologia pré-definida. Cada unidade evolui separadamente, mas somente troca material genético com sua vizinhança. Ao longo do processamento, surgem regiões de aptidão semelhantes. A analogia é a de que o material genético se espalha pela população global como moléculas em um processo de difusão. Esse modelo é mais usado com *arrays* de processadores. Um caso de uso pode ser visto em [45].

O modelo de migração tem sido o mais usado em programação genética e, por motivos que serão explicados melhor na seção 5.7.2, também é o que melhor se encaixa ao uso de computação em rede. Alguns dos aspectos que envolvem a migração são:

- A topologia de comunicação, que define as interconexões entre subpopulações;
- o esquema de migração, que controla quais indivíduos devem migrar de uma subpopulação a outra e quais devem ser substituídos;
- a taxa de migração, que define a quantidade de indivíduos que migram;
- o intervalo de migração, o qual determina a freqüência das migrações;
- a sincronicidade: se as migrações ocorrem ao mesmo tempo, ou de forma assíncrona.

2.18 Topologias de migração

Dentro do modelo de migração, existem diferentes opções de topologias de conexão entre os nós [16], sendo que as mais comuns incluem grades bidimensionais e tridimensionais, hipercubos e toróides.

O modelo mais simples seria o totalmente conectado, em que não há restrições de localidade para migração. Um exemplo pode ser visto na figura 2.11.

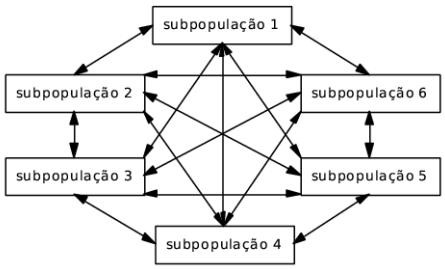


Figura 2.11: Migração em topologia totalmente conectada.

Um esquema de migração específico utilizando esse tipo de topologia é mostrado na figura 2.12. Cada subpopulação sorteia um indivíduo entre os melhores das outras subpopulações, e o escolhido substitui seu pior indivíduo. Isto é repetido para cada subpopulação. O esquema garante que a subpopulação não escolherá um de seus próprios indivíduos.

Um exemplo de topologia bidimensional é a conexão em anel, na qual os indivíduos migram apenas em uma direção em um anel de populações. A figura 2.13 exemplifica este tipo de topologia.

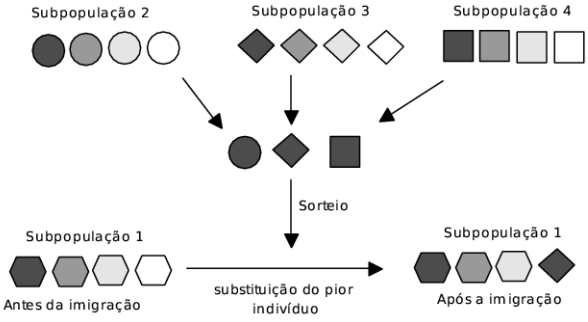


Figura 2.12: Migração em topologia totalmente conectada com esquema de migração irrestrita. Os indivíduos mais claros têm aptidão menor.

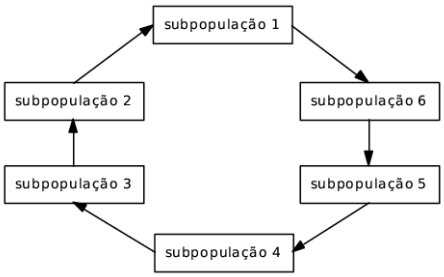


Figura 2.13: Migração em topologia em anel.

Por fim, uma topologia tridimensional, baseada em vizinhança, é mostrada na figura 2.14. Nessa disposição, é definida uma vizinhança na qual uma subpopulação pode migrar indivíduos. Diferentemente da distribuição em anel, a migração pode ocorrer nos dois sentidos entre vizinhos.

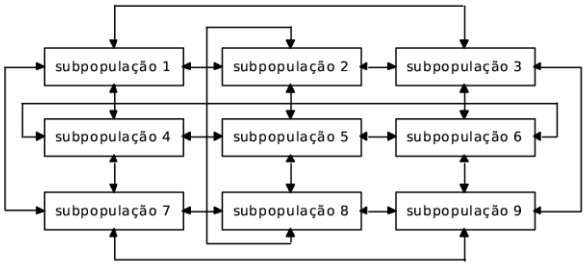


Figura 2.14: Migração em topologia de vizinhança. Este tipo de vizinhança é conhecida como toroidal.

Na seção 5.7 esses conceitos serão usados para explicar as opções tomadas no projeto de processamento distribuído.

3 Programação da expressão gênica

A programação da expressão gênica não foi muito bem aceita inicialmente na comunidade de programação genética. Os artigos da doutora Cândida Ferreira [46], graduada em Bioquímica e com um Ph.D em Biologia, reportavam resultados tão significativamente melhores do que os resultados comuns que os avaliadores dos congressos chegavam a rejeitá-los, alegando serem muito bons para serem verdadeiros. Na página da autora na *Internet* [1] existe o registro de experimentos que chegam a ultrapassar em 374 vezes o algoritmo tradicional em eficiência[2].

O interessante é que Ferreira optou por uma estrutura bastante simples, se for levada em conta a implementação tradicional de programação genética. E, a despeito da simplicidade, a estrutura parece realmente levar a resultados de eficiência indiscutível em relação à abordagem tradicional.

3.1 Representação dos indivíduos

A grande descoberta da doutora Ferreira fundamenta-se, basicamente, em uma representação bastante inteligente dos indivíduos. Os indivíduos, chamados aqui de cromossomos, são representados como uma seqüência linear de símbolos. Esses cromossomos têm tamanho fixo, e são formados por um ou mais genes, também de tamanho fixo. Os genes são decodificados da esquerda para a direita na seqüência de símbolos. A decodificação do gene em uma árvore se dá pela construção nível a nível (em largura) desta. A árvore assim construída é chamada árvore de expressão. Um exemplo de decodificação pode ser visto na figura 3.1.

Nesse processo, como os elementos de um nível são os argumentos dos elementos do nível superior, um nível está completo quando todos os argumentos do nível anterior tiverem sido preenchidos. A codificação da árvore acaba quando não houver mais funções com argumentos em aberto.

[1] www.gene-expression-programming.com
[2] Conforme o método exposto na seção 5.4.2

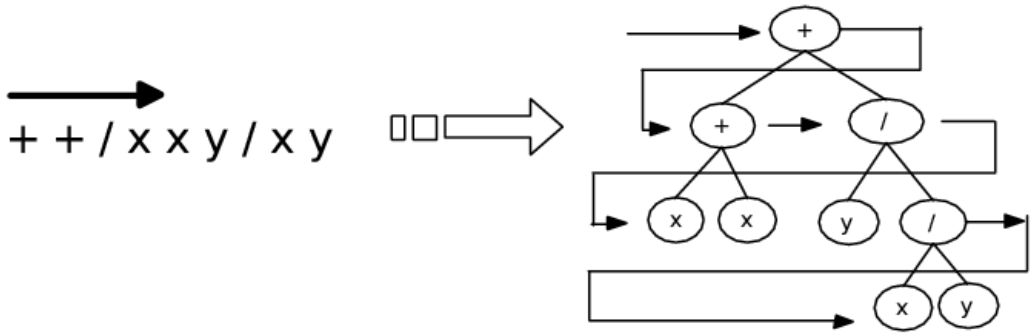


Figura 3.1: Um gene, à esquerda, composto por uma série de símbolos, é lido da esquerda para a direita que vão sendo depositados, nível a nível, na árvore de expressão.

Esse tipo de decodificação permite que existam seqüências não codificantes ao final dos genes. De fato, na maioria dos casos, nem toda a extensão do gene é codificável, e isso representa a essência do funcionamento da programação da expressão gênica, permitindo a modificação do genoma pelos operadores genéticos, sem que haja a ocorrência da criação de estruturas sintaticamente incorretas.

O nome "expressão gênica" vem do fato de existir uma separação entre genótipo (grupo de cromossomos) e um fenótipo (indivíduos formados depois da expressão do genótipo). Os operadores genéticos atuam sobre o genótipo e a avaliação, sobre o fenótipo, assim como ocorre na natureza. A programação genética tradicional não faz distinção entre fenótipo e genótipo. O conceito é um tanto quanto dúbio, dado que o cromossomo pode ser visto apenas como uma forma diferente de se representar uma árvore de programa e que, como se verá na seção 5.6.2, a expressão e execução dos indivíduos podem ser feitas simultaneamente.

3.2 Estrutura dos genes

Os genes têm uma estrutura e certas regras de manipulação que garantem sua integridade. A estrutura divide um gene em cabeça e cauda. A regra de manipulação define que, na cabeça do gene, podem existir elementos terminais e não-terminais, e que na cauda, somente podem existir elementos terminais. Para garantir que o gene tenha um tamanho certo de maneira a assegurar integridade na decodificação, a cauda deve ter um tamanho mínimo. Esse tamanho mínimo é calculado pela fórmula:

$$t = h(n-1)+1, \tag{3.1}$$

onde *t* é o tamanho da cauda, *h* é o tamanho da cabeça, e *n* o número de argumentos do não-terminal de maior aridade no conjunto de não-terminais usado. Isso garante que o pior caso não extravase a estrutura. Um exemplo do funcionamento dessa estrutura é mostrado na figura 3.2. Nesse exemplo, o tamanho da cabeça é 4. Portanto, pela fórmula dada em 3.1, o tamanho da cauda deve ser 5. Na figura, as caudas estão em cinza. A cabeça do gene à esquerda possui dois terminais e dois não terminais. Seguindo as regras de codificação, nem toda a seqüência deste gene é codificante, e a árvore resultante tem apenas três nós. Mas, modificando-se apenas o segundo elemento da cabeça, um *x*, por um operador de adição, tem-se uma árvore de expressão completamente diferente da anterior.

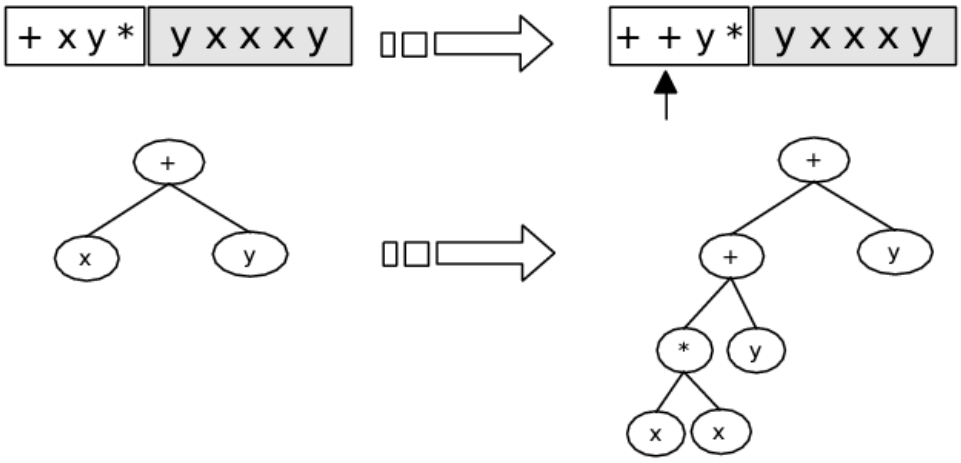


Figura 3.2: Estrutura do gene. O gene à esquerda dá origem à árvore de expressão abaixo deste; somente os três primeiros elementos são codificantes. Mudando se apenas um elemento (no lugar indicado pela seta), a árvore de expressão codificada muda completamente.

Manter a organização estrutural cabeça-cauda é a chave para a manutenção da correção sintática. Portanto, os operadores genéticos devem seguir as regras de organização: ao se modificar um elemento da cabeça, não há restrições quanto à geração de terminais ou não terminais, mas modificações na cauda somente devem gerar elementos terminais.

3.3 Estrutura dos cromossomos

Cada problema amolda-se a uma determinada estrutura de cromossomo. A solução de uma regressão para uma função simples, na maior parte das vezes, exige árvores menores do que as de uma função muito complexa. Portanto, entre os parâmetros importantes para iniciar a resolução de um problema, estão o número de genes por cromossomo e o tamanho da cabeça dos genes. Estes dois parâmetros definem a estrutura do cromossomo.

Os cromossomos em programação da expressão gênica são multigênicos, isto é, podem ser formados por um ou mais genes. A figura 3.3 mostra um cromossomo com três genes. Como o tamanho da cabeça é 4, o tamanho da cauda, definido pela fórmula 3.1, é 5. Temos então genes de tamanho 9 e o cromossomo de tamanho 27.

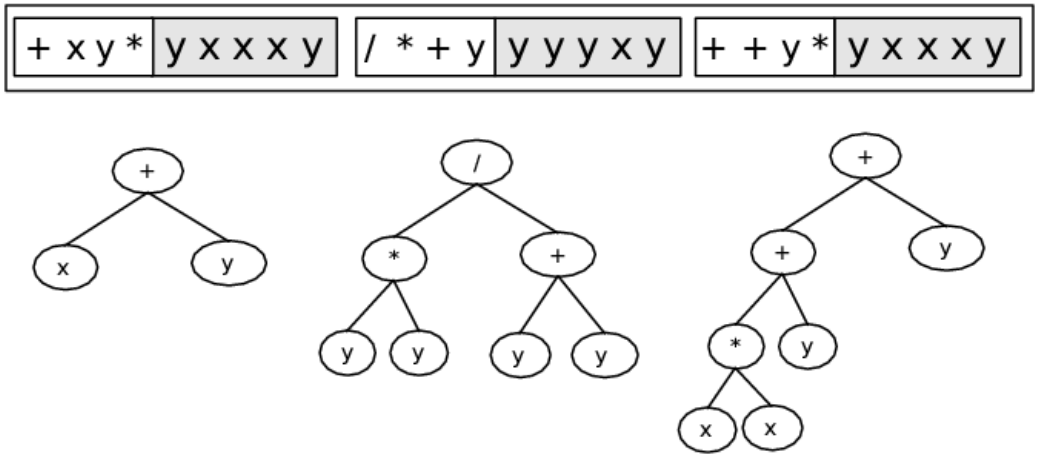


Figura 3.3: Estrutura de cromossomo com três genes, com cabeças de tamanho 4. Abaixo as três subárvores de expressão codificadas nos genes.

Os genes em um cromossomo geram árvores de expressão distintas, chamadas subárvores de expressão. Essas subárvores podem trabalhar em conjunto ou separadas para formar a solução. Geralmente, elas são unidas em uma única árvore por meio de um operador de ligação definido como parâmetro do programa. Ferreira faz analogia à Genética, em que subcadeias de proteínas se unem para formar proteínas maiores.

Funções de ligação podem ser operadores booleanos (como “if”, “and” ou “or”), para problemas envolvendo lógica de primeira ordem, ou os operadores de adição e multiplicação, para o caso de expressões matemáticas. A aptidão do indivíduo nesse caso é definida pela árvore formada pelos genes conectados pela função de ligação. Há casos em que se trabalha com aptidão separada para cada subárvore [46].

A função de ligação opera da esquerda para a direita na cadeia de genes. A figura 3.4 mostra a árvore resultante da conexão das três subárvores do exemplo anterior, usando-se a adição como operador de ligação.

3.4 Algoritmo

O funcionamento geral de programação da expressão gênica é delineado no algoritmo da figura 3.5. Um fator notável no algoritmo é a enorme variedade de operadores genéticos. De

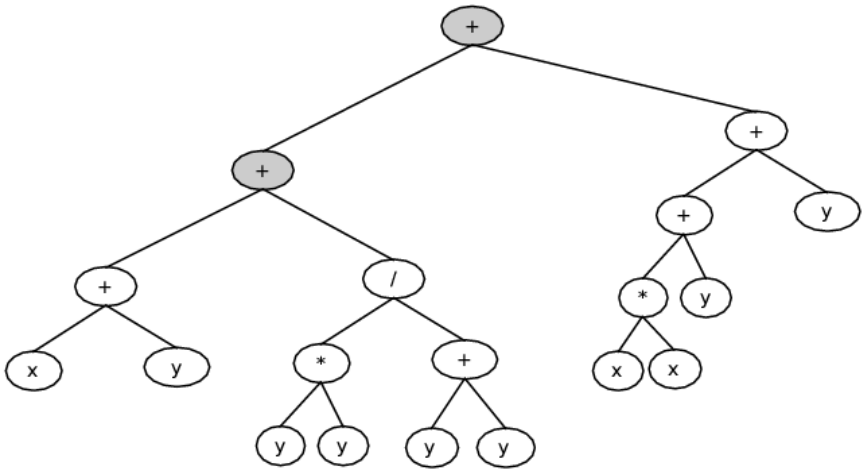


Figura 3.4: Conexão de subárvores de expressão através do operador de ligação. Neste caso o operador escolhido é a adição (nós destacados em cinza).

fato, Ferreira diz que a grande flexibilidade gerada pela estrutura dos cromossomos permite a aplicação de uma série de novos operadores. Outro detalhe que não existe no algoritmo tradicional de programação genética é o passo "expressar cromossomos". Esse passo representa a decodificação dos cromossomos em árvores de expressão (programas). Outra diferença é que os operadores genéticos, com exceção do operador de replicação, são aplicados em série, com probabilidades independentes.

3.5 Operadores genéticos

A aplicação em série dos operadores implica que um mesmo cromossomo pode ser modificado por vários operadores genéticos de busca. Isto significa que as modificações sobre um cromossomo são cumulativas. Um determinado operador é aplicado apenas uma vez para cada cromossomo.

Ferreira usa uma nomenclatura diferente de Koza para o operador de cópia. Koza o chama de operador de reprodução e Ferreira, de operador de replicação. O operador de reprodução, para Ferreira, compõe uma classe composta por todos os operadores genéticos de busca.

3.5.1 Seleção

O algoritmo padrão de programação da expressão gênica usa seleção por roleta. Segundo Ferreira, este método de seleção apresenta a melhor relação tempo de processamento *versus* taxa de sucesso. Mas também afirma que há pouca diferença entre métodos de seleção, e que

Criar população inicial;
Expressar os cromossomos;
Executar os programas resultantes;
Avaliar aptidão;
enquanto *critério de término não atingido* **faça**
  Manter melhor programa (elitismo);
  Selecionar programas;
  Aplicar replicação , com p_{rep};
  Aplicar **mutação**, com p_{mut};
  Aplicar **transposição de inserção**, com p_{tr};
  Aplicar **transposição de inserção na raiz**, com p_{tr_raiz};
  Aplicar **transposição de gene**, com p_{tr_gene};
  Aplicar **recombinação em um ponto**, com p_{rec_1pt};
  Aplicar **recombinação em dois pontos**, com p_{rec_2pt};
  Aplicar **recombinação de genes**, com p_{rec_gen};
  Expressar os cromossomos;
  Executar os programas resultantes;
  Avaliar aptidão;
fim

Figura 3.5: Algoritmo de programação da expressão gênica.

a "essência do algoritmo está no poder dos operadores genéticos" [46]. O elitismo simples (de apenas um indivíduo) é sempre aplicado. Não há parâmetros para aumentar o número de elementos da elite, ou mesmo para evitar a aplicação de elitismo.

3.5.2 Replicação, mutação e inversão

O operador de replicação funciona da maneira tradicional: o indivíduo, escolhido com uma probabilidade p_{rep} previamente estabelecida para o operador, é copiado para a próxima geração. O indivíduo assim escolhido não sofre alteração por nenhum outro operador genético.

O operador de mutação aplicado é o pontual. A mutação ocorre com uma probabilidade previamente estabelecida p_{mut} e, quando aplicada, escolhe um ponto do cromossomo para simples troca do símbolo. O símbolo para substituição é escolhido aleatoriamente do conjunto de terminais e não terminais. Tipicamente, Ferreira aplica uma taxa de mutação equivalente a dois pontos por cromossomo. A única restrição a esse operador é que, para a substituição de símbolos da cauda, deve-se usar somente símbolos não-terminais. Diferentemente da programação genética usual, não é necessário haver compatibilidade quanto ao número de argumentos entre o elemento a ser substituído e o novo elemento; a própria organização estrutural-funcional mantém a correção sintática. Podem ocorrer também mutações na parte não codificante. Essas

mutações podem tornar-se ativas posteriormente no caso de se transformarem em codificantes por uma alteração futura. A figura 3.6 ilustra um exemplo da aplicação da operação de mutação sobre um cromossomo.

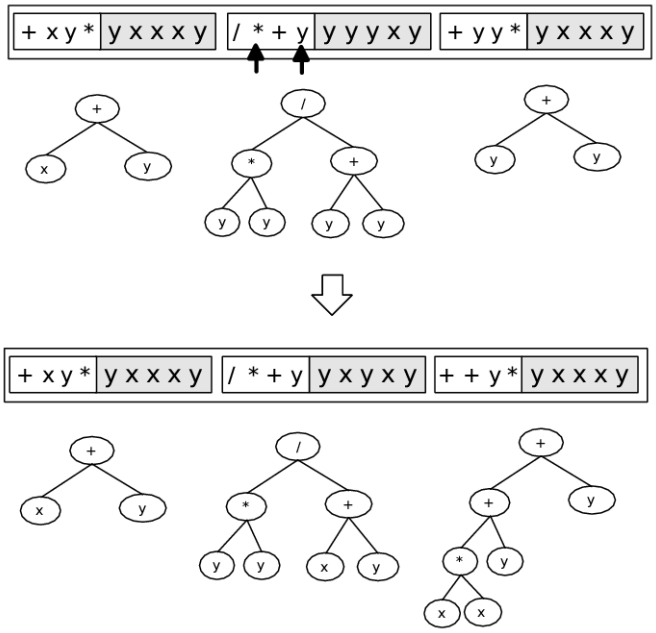


Figura 3.6: Operador de mutação. Os pontos escolhidos para mutação estão indicados pelas setas. O indivíduo original é mostrado na parte de cima na figura, o indivíduo resultante da operação mostrado na parte de baixo. As respectivas subárvores de expressão são representadas logo abaixo dos cromossomos.

O operador de inversão, criado pela própria Ferreira, faz a simples inversão de uma seqüência de símbolos na cabeça do gene. O cromossomo é escolhido com uma probabilidade p_{inv} estabelecida para o operador. O gene em que será aplicado e os pontos inicial e final para inversão dentro da cabeça deste são escolhidos aleatoriamente. A figura 3.7 mostra um exemplo da ação do operador de inversão.

3.5.3 Recombinação

Há três tipos de recombinação: recombinação em um ponto, recombinação em dois pontos e recombinação de genes. De forma semelhante aos outros operadores, a aplicação de cada modalidade de recombinação está associada a probabilidades independentes e pré-determinadas como parâmetros do sistema, mostradas no algoritmo como: p_{rec_1pt}, p_{rec_2pt} e p_{rec_gene}, respectivamente.

Na recombinação em um ponto, os cromossomos pais são emparelhados e é escolhido um ponto ao acaso para corte. As metades a partir desse ponto são permutadas, dando origem a dois

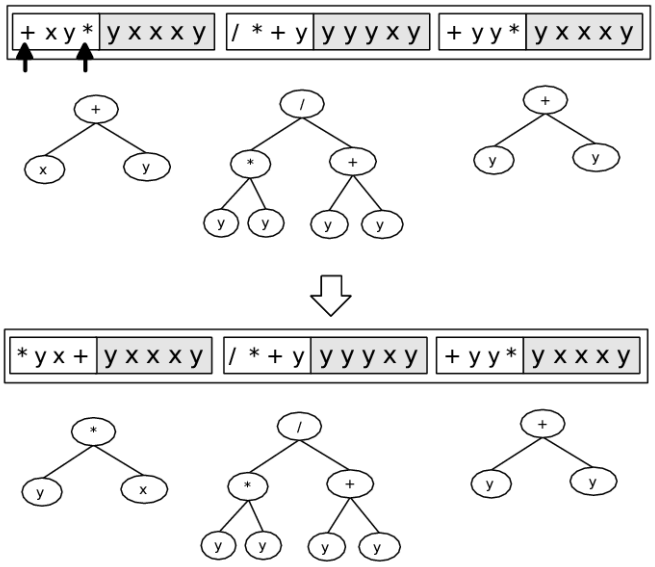


Figura 3.7: Operador de inversão. Os pontos inicial e final da seqüência de inversão no cromossomo original são mostrados pelas setas. As subárvores de expressão são representadas logo abaixo dos cromossomos. O cromossomo resultante é mostrado na parte inferior da figura.

cromossomos prole. Ferreira alega que a natureza destrutiva da recombinação é balanceada pela presença de partes não codificantes e pela existência de múltiplos genes em um cromossomo. Essa característica diminuiria, então, a possibilidade de quebra dos blocos construtores. Um exemplo da aplicação do operador de recombinação é mostrado na figura 3.8.

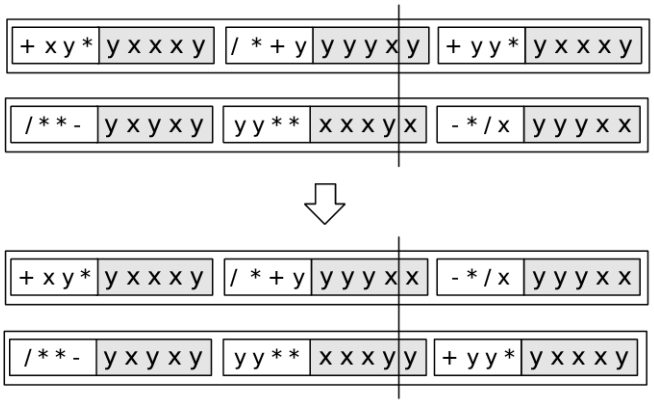


Figura 3.8: Operador de recombinação em um ponto. Os dois cromossomos pais são mostrados na parte de cima da figura. O ponto escolhido para corte é demarcado pelo segmento de reta. Os cromossomos resultantes são mostrados na parte inferior da figura.

Para a recombinação em dois pontos, o processo é semelhante, com a diferença de que se escolhem dois pontos ao acaso para corte. As seqüências entre esses dois pontos é trocada entre os pais, dando origem a dois cromossomos prole. A recombinação em dois pontos é mais

destrutiva que a anterior. A figura 3.9 ilustra o funcionamento deste operador.

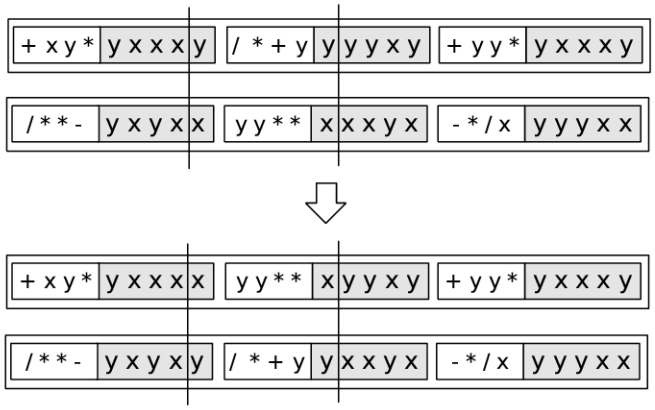


Figura 3.9: Operador de recombinação em dois pontos. Os dois cromossomos pais são mostrados na parte de cima da figura. As seqüências escolhidas para troca são as que estão delimitadas entre os segmentos de reta. Os cromossomos resultantes são mostrados na parte inferior da figura.

Na recombinação de genes, há a troca de dois genes de mesma posição (escolhida aleatoriamente) entre os dois pais. Portanto, a mudança estabelecida por esse operador ocorre em um nível superior, trocando subárvores completas entre indivíduos. Essa operação não cria novos genes, mas tem a função de "embaralhá-los" entre os indivíduos da população. Um exemplo é mostrado na figura 3.10.

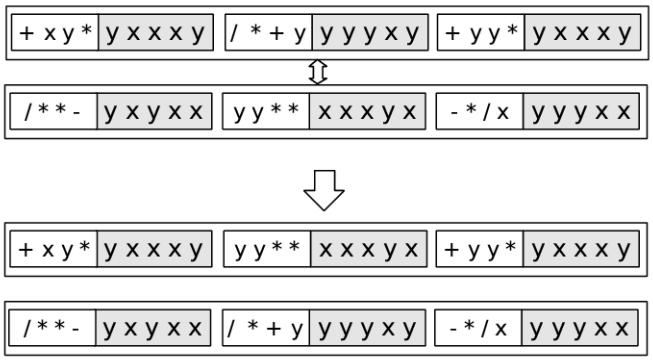


Figura 3.10: Operador de recombinação de genes. Os dois cromossomos pais são mostrados na parte de cima da figura e os genes escolhidos para permuta indicados pelo sinal de duas setas. Os cromossomos resultantes são mostrados na parte inferior da figura.

3.5.4 Recombinação e outras variantes

É interessante notar que, no caso da recombinação na programação genética tradicional, quando dois indivíduos idênticos são escolhidos para cruzamento, na maioria das vezes, isso dá

origem a proles totalmente distintas entre si e também diferentes dos pais. Isso ocorre porque, na representação em árvore, as subárvores escolhidas para permuta têm grande probabilidade de serem diferentes, mesmo com pais iguais. Essa mesma situação na recombinação em programação da expressão gênica não é capaz de alterar os cromossomos iguais, pois a operação troca seqüências idênticas no mesmo lugar.

Nordin [30] alega que o tipo de estrutura que usa em programação genética linear permite um tipo de recombinação menos destrutivo. Seus genes são compostos por blocos de instruções de tamanho fixo e a recombinação somente pode atuar trocando blocos entre os pais. Na natureza, durante a recombinação, os cromossomos são alinhados, e as permutações de genes ocorrem apenas entre regiões iguais. Regiões iguais significam mesma funcionalidade e isso impede que uma região que dá características para o nariz acabe trocando genes com a que dá características ao pé. Esse tipo de recombinação é chamado homólogo. Na programação genética tradicional regiões diferentes são trocadas e pé acaba cruzando com nariz. Isto é, blocos construtores com funcionalidades definidas acabam sendo quebrados e combinados de maneira errada. Nordin afirma que experimentos com recombinação homóloga, trocando blocos apenas na mesma região dos cromossomos, apresentaram diminuição do fenômeno de inchaço.

Em um estudo da dinâmica dos operadores na programação da expressão gênica [47], Ferreira mostra que os operadores nessa variante comportam-se de maneira totalmente diferente dos estudos anteriores em programação genética. Mostra, ainda, que a recombinação tem poder muito limitado frente à mutação e, se aplicada sozinha, tem a tendência de homogeneizar as populações. Em uma comparação entre os tipos de recombinação da variante, mostra-se que quanto mais conservadora a operação, mais rapidamente ocorre a homogeneização. Com esses resultados, conclui que a recombinação homóloga, assim como na natureza, não serve para outra coisa além de manter o *status quo* em períodos de *stasis*(parada na evolução).

3.5.5 Transposição

A operação de transposição consiste na cópia de uma seqüência contígua de símbolos de uma parte do cromossomo e sua inserção em outra parte do mesmo cromossomo. Há três tipos de transposição: transposição de seqüência de inserção, transposição de seqüência de inserção para a raiz e transposição de genes. Essas operações são aplicadas com probabilidades pré-definidas e independentes: p_{tr}, p_{tr_raiz} e p_{tr_gene}, respectivamente.

No caso da *transposição de seqüência de inserção*, sorteiam-se aleatoriamente os pontos inicial e final da seqüência a ser copiada e a posição, na cabeça de um dos genes, em que será inserida a cópia. Essa posição a ser escolhida não pode ser a primeira da cabeça (raiz

da árvore de expressão). A seqüência de símbolos à direita da posição de inserção é afastada para a acomodação da parte inserida e os símbolos que transbordam o tamanho da cabeça são eliminados. A figura 3.11 mostra um exemplo desse tipo de transposição.

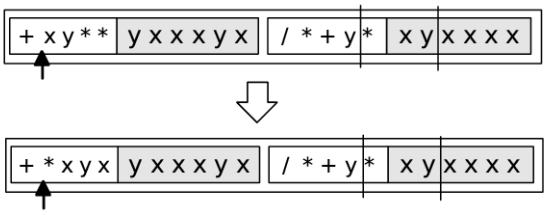


Figura 3.11: Operador de transposição de seqüência de inserção. O cromossomo original é mostrado na parte de cima da figura com a seqüência de transposição escolhida delimitada pelos segmentos de reta. O ponto escolhido para inserção é indicado pela seta. Os três últimos elementos da cabeça são eliminados para acomodação da inserção. O cromossomo resultante é mostrado na parte inferior da figura.

A *transposição de seqüência de inserção para a raiz* exige que o primeiro elemento da cópia a ser inserida seja uma função. Um gene do cromossomo a sofrer a operação é escolhido aleatoriamente. Sorteia-se, então, uma posição na cabeça do gene e percorre-se a seqüência da esquerda para a direita, a partir desse ponto, até que se ache uma função. A primeira função encontrada passa a ser o início da seqüência. O tamanho é também escolhido aleatoriamente e pode pegar parte da cauda. A seqüência copiada é *necessariamente* inserida na primeira posição da cabeça (raiz) do gene escolhido. Quanto ao afastamento dos símbolos à direita da inserção e à eliminação dos símbolos que ultrapassam o limite da cabeça, o tratamento é o mesmo que o usado na modalidade anterior. A figura 3.12 ilustra um caso de aplicação do operador.

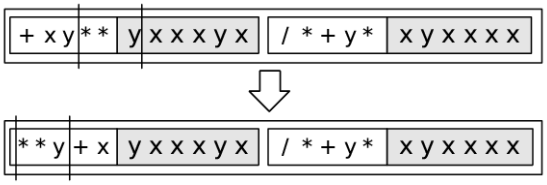


Figura 3.12: Operador de transposição de seqüência de inserção. O cromossomo original é mostrado na parte de cima da figura com a seqüência de transposição escolhida delimitada pelos segmentos de reta. A seqüência é inserida na cabeça do gene escolhido, resultando no indivíduo mostrado na parte de baixo. Os três últimos elementos da cabeça são eliminados para acomodação da inserção.

A transposição de genes não faz mais do que embaralhar os genes do cromossomo que passa por esta operação. Ele escolhe um dos genes do cromossomo ao acaso e o *move* para o início deste. Esse mover significa que ele é removido da posição em que estava e inserido antes do primeiro gene afastando os genes à direita. Essa operação, apesar de não criar genes

novos, cria diversidade indiretamente. No caso de operadores de ligação não comutativos, a posição dos genes faz diferença. Além disso, mesmo com operadores comutativos, o embaralhamento dos genes pode modificar a forma como os cromossomos são afetados pelo operador de recombinação. No caso de pais iguais, por exemplo, depois de uma transposição de genes, apesar de a árvore de expressão dos pais ser a mesma, uma recombinação destes resulta em filhos diferentes. Uma demonstração da aplicação desse operador é vista na figura 3.13

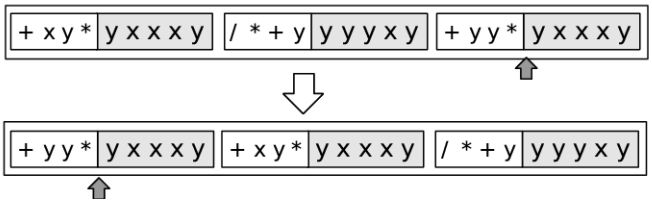


Figura 3.13: Operador de transposição de genes. O cromossomo original é mostrado na parte de cima da figura com o gene escolhido para transposição indicado pela seta cinza. O gene é movido para a primeira posição do cromossomo afastando os outros genes para a direita.

3.6 Funções de aptidão

Ferreira define duas funções de aptidão básicas para problemas de regressão simbólica. A primeira equação é praticamente a mesma equação de erro absoluto dada em 2.1.

$$f_i = \sum_{j=1}^{C_t} \left[M - \left| C_{i,j} - T_j \right| \right], \tag{3.2}$$

onde f_i é a função de aptidão para o indivíduo i , C_t é o número de casos de avaliação, $C_{i,j}$ é o valor avaliado do indivíduo i para o caso j e T_j é o valor esperado para o caso j.

A diferença para a equação de erro absoluto é o uso de uma constante M, chamada faixa de seleção, que define um valor máximo para aptidão dado pela equação:

$$f_{max} = C_t * M \tag{3.3}$$

A equação 3.3 é obtida da equação 3.2 fazendo $|C_{i,j} - T_j| = 0$ para todo caso j, isto é, no caso de diferença zero para todos os pontos de avaliação entre a função alvo e a função estimada.

A segunda função de erro é a função de erro relativo — também bastante usada em compu-

tação evolutiva no geral — acrescida do mesmo artifício com a constante M.

$$f_i = \sum_{j=1}^{C_t} \left[M - \left| \frac{C_{i,j} - T_j}{T_j} 100 \right| \right] \tag{3.4}$$

Como de praxe, a função de aptidão é completamente dependente do domínio do problema em foco. Em seu sistema comercial, chamado *Automatic Problem Solver*, Ferreira disponibiliza também outras funções de aptidão.

3.7 Geração de constantes

Como visto na seção 2.16, a geração de constantes é considerada um fator deficiente em programação genética. O algoritmo padrão de programação da expressão gênica não faz uso explícito de constantes. A geração de constantes é deixada a cargo da própria evolução surgindo a partir de expressões matemáticas do tipo:

$$\frac{x+x}{x} = 2$$

Ferreira também apresenta uma forma alternativa com geração explícita de constantes. A conexão das constantes com as árvores de expressão se dá de forma indireta. É definido um vetor global de constantes (A) que serve de domínio para terminais deste tipo. A estrutura do gene é modificada com a adição de uma seqüência de t elementos[3] após a cauda, representando domínio das constantes (D_C). Esses elementos representam índices do vetor global de constantes. Um novo símbolo, geralmente representado com o sinal de interrogação, é adicionado ao conjunto de terminais para atuar como demarcador de lugar para constantes. Quando a população inicial é criada, o algoritmo de criação de indivíduos tem a função adicional de gerar índices aleatórios na seção que contém o domínio das constantes nos genes.

No processo de evolução, os índices gerados no início são permutados entre os cromossomos por meio dos operadores de recombinação e modificados por mutação e transposição. Na expressão dos cromossomos, após a geração das árvores de expressão, os símbolos "?" são substituídos de cima para baixo e da esquerda para a direita com os valores indicados pelos índices em D_C. Um exemplo passo a passo do processo de expressão é mostrado na figura 3.14.

Ferreira afirma, no entanto, que o uso de constantes aleatórias em programação da expressão gênica é desnecessário para a maioria dos problemas de regressão simbólica. Em um

[3] t é o tamanho da cauda.

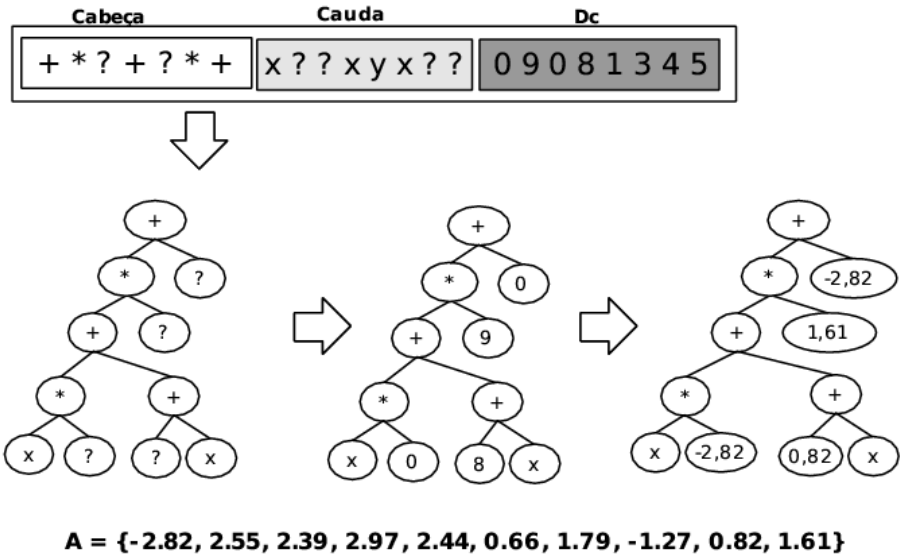


Figura 3.14: Geração explícita de constantes. O processo de expressão é mostrado em três fases para melhor compreensão: a subárvore é expressa, os nós demarcados com o sinal de interrogação são preenchidos com os índices de D_C e, finalmente, com os valores do vetor global A apontados pelos índices.

artigo sobre o tema [42], demonstra isto pela comparação de três problemas deste tipo tratados com e sem o uso de constantes. Mas há vozes discordantes. Xin Li [48] faz duas afirmações que mostram opiniões contrárias às afirmações de Ferreira. Primeiro que os operadores desta modalidade são mais destrutivos que na programação genética tradicional e que isso impede o algoritmo de encontrar funções com estruturas adequadas. Segundo, propõe o uso de algoritmos de geração explícita de constantes para amenizar essa característica destrutiva.

3.8 Genes homeóticos

É possível deixar a programação da conexão entre subárvores de expressão a cargo da evolução dos indivíduos, deixando que estes encontrem a melhor forma de combinar essas subestruturas.

Na natureza existem genes “mestres” que controlam a expressão de outros genes. Estes genes são denominados homeóticos. Em programação da expressão gênica, os genes homeóticos têm o mesmo papel. São implementados no final da cadeia de genes comuns dos cromossomos e seus elementos são operadores de ligação e índices que apontam para os genes não homeóticos do mesmo cromossomo. Apesar do domínio dos terminais e não-terminais ser diferente dos outros genes, a estrutura é a mesma, composta de cabeça e cauda, e as regras para alteração são iguais, também. O tamanho desses genes pode ser diferente dos demais. A estrutura

derivada da expressão de um gene homeótico é denominada célula. Um cromossomo pode ter mais de um gene homeótico, dando origem a um indivíduo multicelular. A figura 3.15 mostra a composição e expressão de um cromossomo contendo um gene homeótico. A expressão das subárvores um, dois e três depende da referência pelos índices da árvore de expressão gerada pelo gene homeótico ao final do cromossomo. Como a parte codificante do gene homeótico faz referência apenas às subárvores dois e três, a subárvore um não é expressa.

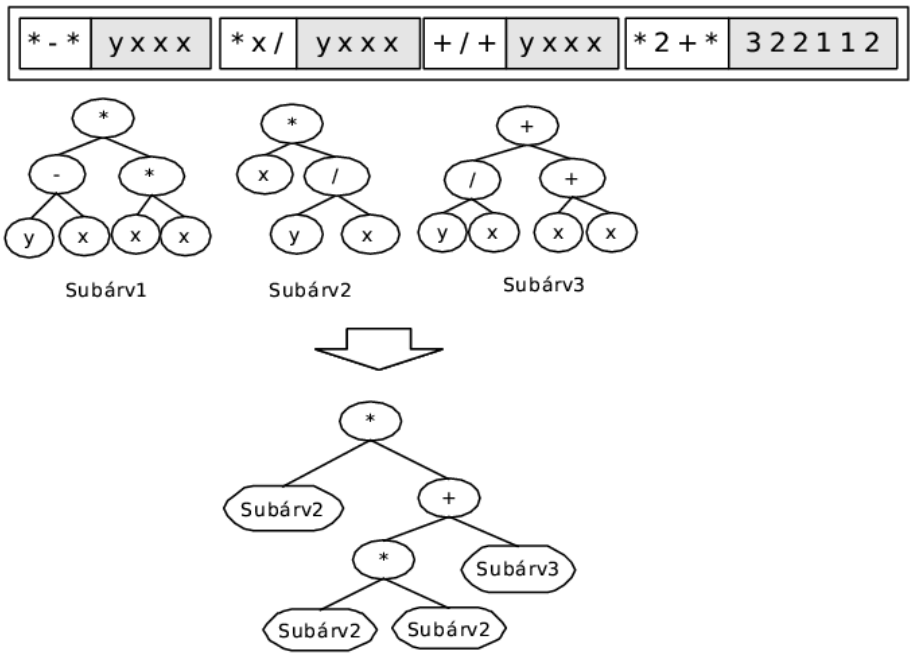


Figura 3.15: Expressão de cromossomo contendo gene homeótico. O gene homeótico controla a expressão dos demais genes e a forma como estes são conectados. Apenas os genes 2 e 3 são expressos, dando origem à célula mostrada na parte inferior da figura.

Apesar de guardar vaga semelhança com o papel das *funções definidas automaticamente*, pois não possuem, como na implementação de Koza, o conceito de parâmetros, Ferreira afirma que esta é um forma elegante de implementar esse tipo de conceito.

4 O problema geofísico

A vida no nosso planeta depende do Sol. Todo o ciclo natural tem sido orientado pelo ciclo solar desde os primórdios. O conhecimento do ciclo solar sempre foi importante para a humanidade, pois disso dependia sua sobrevivência: a agricultura e a caça apresentam características sazonais. Culturas antigas já faziam registros dos períodos de solstício e equinócio, muitos monumentos foram construídos especialmente para demarcar dias específicos do ciclo solar.

O interesse sobre o ciclo das estações gerou muito conhecimento empírico sobre nossa estrela. Alguns povos antigos chegaram a tal conhecimento de astronomia que eram capazes de prever eclipses solares e lunares com bastante precisão e antecedência. No ocidente, o conhecimento científico sobre o Sol apenas começou a tomar lugar no final da Renascença, com o surgimento da nova física com Kepler, Copérnico e Galileu. A teoria heliocêntrica tornou muito mais fácil entender certos fenômenos e formou as bases sobre as quais seria construída, mais tarde, a teoria da gravitação universal de Newton.

Um fenômeno solar conhecido desde a mais remota antiguidade é o das manchas solares, tendo sido registrado na China já no ano de 28 a.C. Essas manchas consistem em áreas no Sol com luminosidade menor do que a da vizinhança, podendo ser observadas a olho nu. Seu estudo científico começou na época de Galileu, no início do século XVII, sendo este um dos que iniciou o registro sistemático a partir de observações com telescópios por projeção da imagem do Sol. Foi Samuel Heinrich Schwabe, em 1843, que descobriu que as manchas seguem um ciclo de onze anos, variando em número entre máximos e mínimos. Mas foi só recentemente que se descobriu uma relação entre a atividade magnética solar e o ciclo das manchas solares. Hoje, o estudo das manchas solares é extremamente importante para se entender a dinâmica dos processos eletromagnéticos e sua influência na projeção de gases do Sol.

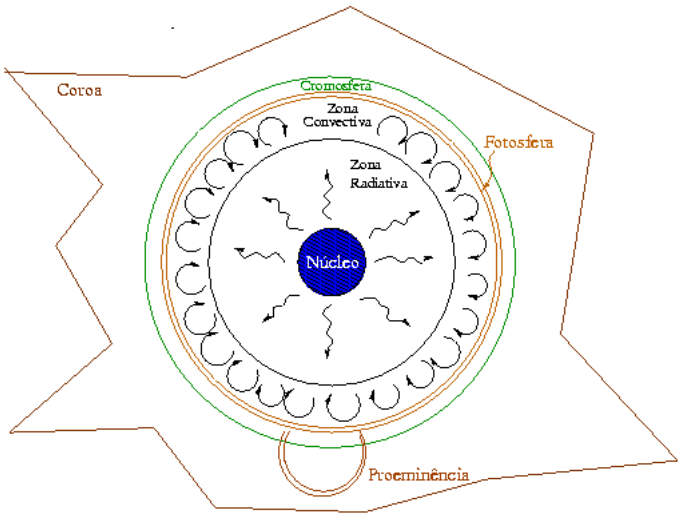


Figura 4.1: Estrutura do Sol em corte.

4.1 Interação Sol–Terra

A influência de nossa estrela sobre a Terra não se dá somente por meio da troca de calor, mas também através de seu campo eletromagnético e pela emissão de gases eletricamente carregados. O Sol possui um comportamento dinâmico e essa dinâmica afeta o clima terrestre.

O Sol é constituído principalmente por hidrogênio e hélio. Dentro de estrelas como o Sol, o hidrogênio se transforma gradualmente em hélio por meio da fusão nuclear: quatro átomos de hidrogênio se juntam para formar um átomo de hélio. A massa de um átomo de hélio é menor que a de quatro átomos de hidrogênio, e a reação libera energia de acordo com a equação $E = mc^2$ de Einstein. A figura 4.1 mostra, esquematicamente, a estrutura do Sol. A atmosfera solar possui duas partes: a fotosfera e a cromosfera. A fotosfera é a parte interna da atmosfera solar e forma a parte visível da superfície, onde aparecem as manchas solares. A parte mais externa, visível durante eclipses solares formando um anel em volta do Sol, é chamada cromosfera. Além da atmosfera solar, situa-se a coroa solar, vista como um halo luminoso durante os eclipses. A coroa solar é composta por gases ionizados em temperaturas altíssimas. Esse gás ionizado, em Química e Física, é denominado plasma.

Estrelas como o Sol expelem grande quantidade de plasma. Esse plasma expelido, viajando através do sistema solar, é composto por nuvens de partículas como elétrons, prótons e até mesmo subpartículas, como neutrinos. Esse fenômeno é conhecido como vento solar. Um efeito visível do vento solar é a cauda dos cometas: ao se chocar com o cometa, a nuvem de íons espalha partículas de gelo e gás dos cometas.

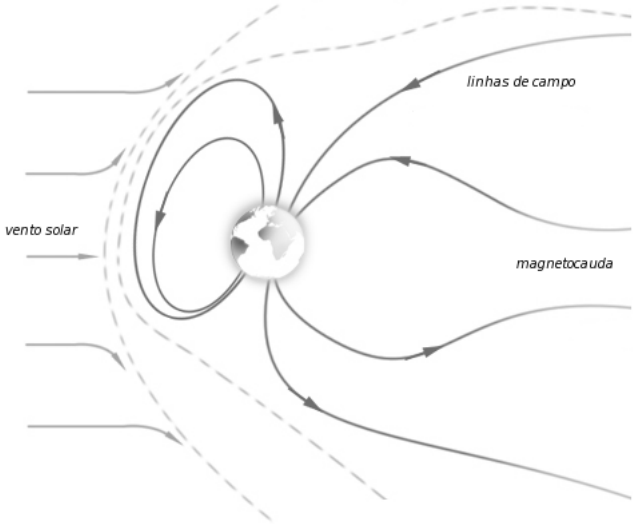


Figura 4.2: A magnetosfera terrestre. O vento solar é defletido pela magnetosfera, mas, ao mesmo tempo, causa a deformação das linhas de campo magnético.

Variações no vento solar estão associadas à atividade das manchas solares. Períodos com maior quantidade de manchas solares correspondem a épocas com maior fluxo de vento solar. Esses períodos também estão relacionados à maior ocorrência de um fenômeno conhecido como ejeção de massa coronal, que consiste em explosões solares de grande magnitude seguidas do lançamento de grandes quantidades de material da coroa. Essas ejeções seguem grandes arcos magnéticos e viajam a velocidades maiores do que o vento solar. Quando atingem a Terra, chegam a causar queda de energia em regiões localizadas em altas latitudes devido à sobrecarga das linhas de tensão.

4.2 Magnetosfera terrestre

Magnetosfera é o campo eletromagnético formado em torno de certos corpos espaciais. A Terra possui uma magnetosfera cujos pólos magnéticos quase se alinham com os pólos geográficos. Os corpos espaciais que possuem magnetosfera tendem a defletir o vento solar, impedindo que grande parte das partículas atinja diretamente sua superfície. Em contrapartida, esse vento solar, sendo ionizado, exerce influência sobre o campo magnético do corpo atingido, comprimindo e deformando o campo. A figura 4.2 mostra a magnetosfera terrestre com a deformação característica causada pelo vento solar. A ação do vento solar comprime as linhas de campo na região de impacto frontal com a magnetosfera e causa um arrasto destas na parte de trás formando uma cauda magnética (a magnetocauda).

A velocidade do vento solar nas proximidades da Terra é de 400 a 800 km/s e a densidade normalmente fica em torno de dez partículas por centímetro cúbico. As mudanças na densidade e velocidade do vento solar causam compressões e expansões na magnetosfera, conforme o fluxo aumenta ou diminui, respectivamente.

O Sol gera um campo eletromagnético que afeta todo o nosso sistema solar. Esse campo eletromagnético recebe o nome de campo magnético interplanetário. O comportamento do campo interplanetário é de suma importância para entender os efeitos do vento solar na magnetosfera. A orientação local (nas cercanias de nosso planeta) do campo eletromagnético do Sol muda com o tempo e altera os efeitos do vento solar na magnetosfera.

4.3 Ionosfera

A ionosfera é a parte da atmosfera terrestre formada por gases ionizados pela ação dos raios solares. A ionização dos gases se dá pela perda dos elétrons na camada externa dos átomos, libertos pela energia irradiada pelo Sol. A ionosfera é composta, portanto, por um plasma fracamente ionizado. O plasma, dotado de carga elétrica, recebe influência de campos magnéticos. Por esse motivo, mudanças na magnetosfera regidas pela ação do vento solar afetam a ionosfera. Por ser composto de gases, também sofre a ação dos ventos e, por ter peso, sofre ação da gravidade terrestre e da maré lunar. Quanto ao seu nível de ionização, composição de gases e freqüência característica, a ionosfera pode ser dividida em três camadas: D, E e F.

A camada D forma a base da ionosfera, situando-se de 50 a 90 km acima da superfície terrestre. Por estar em uma zona de maior pressão, prevalece nessa camada a ionização molecular, formada na combinação e quebra de moléculas como o dióxido de nitrogênio, o ozônio e vapor de água. A camada D desaparece no período noturno. Camada intermediária da ionosfera, a região E situa-se de 90 a 140 km de altitude. Possui a maior condutividade elétrica das três camadas. Por esse motivo, correntes elétricas formam-se nessa área e acabam por influenciar o comportamento das camadas vizinhas. A camada E também desaparece durante a noite, deixando apenas alguns rastros esporádicos. A camada F é o topo da ionosfera (acima de 140 km de altitude). Nela a pressão atmosférica é muito baixa, tornando difícil a formação de moléculas, por isso há a prevalência de átomos livres. Portanto a ionização não é mais molecular, mas atômica. Pela baixa pressão e proximidade com a magnetosfera, essa camada é muito mais sujeita à ação das correntes magnéticas do que aos chamados ventos neutros (correntes de ar). Nas camadas D e E, onde a pressão é maior, acontece o contrário. A camada F pode ser dividida em duas subcamadas: a camada F1 e a camada F2. A camada F1 se esvanece durante a noite,

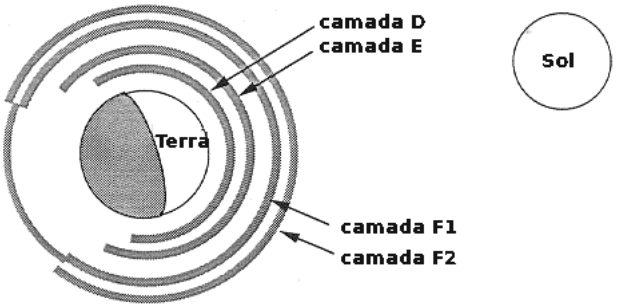


Figura 4.3: Camadas da ionosfera terrestre e seu comportamento diurno-noturno.

deixando apenas a camada F2. A figura 4.3 ilustra a posição e comportamento das camadas da ionosfera.

4.4 Ionosfera e telecomunicações

As camadas da ionosfera apresentam diferentes comportamentos quanto a freqüências de sinais de rádio. A camada D, por exemplo, tem tendência a absorver ou amenizar freqüências usadas nas comunicações por rádio como as ondas AM. Esse é o motivo das rádios AM ajustarem seus sinais antes do anoitecer e do nascer do Sol: o desaparecimento noturno da camada D permite economia de potência de transmissão.

As camadas E e F são bastante usadas para comunicações a longa distância por ondas de rádio. Isso porque essas camadas têm a propriedade de refletir freqüências específicas. Essa característica permite que lugares isolados entre si pela curvatura da Terra, isto é, sem contato em "linha reta", consigam comunicar-se. O método consiste em transmitir o sinal indiretamente, em direção à ionosfera, de forma que esta age como retransmissor da comunicação. A camada F2, por estar sempre presente, é a mais importante para as telecomunicações. A figura 4.4 ilustra o funcionamento desse tipo de uso da ionosfera. Na ilustração, transmissor e receptor não conseguiriam comunicação devido à curvatura da Terra, mas, com o uso da ionosfera como meio de reflexão do sinal, a comunicação pode ser estabelecida.

Períodos com grande quantidade de manchas solares correspondem a excelentes períodos de transmissão para rádio. Muitas manchas solares significam mais atividade solar e, conseqüentemente, mais fluxo solar. Quanto maior o fluxo solar, mais forte a ionização da atmosfera. Se for mais forte a ionização, melhor será a capacidade de reflexão da ionosfera. Ionização fraca causa perdas de sinal.

Além disso, comunicações por satélite dependem de certa regularidade no comportamento

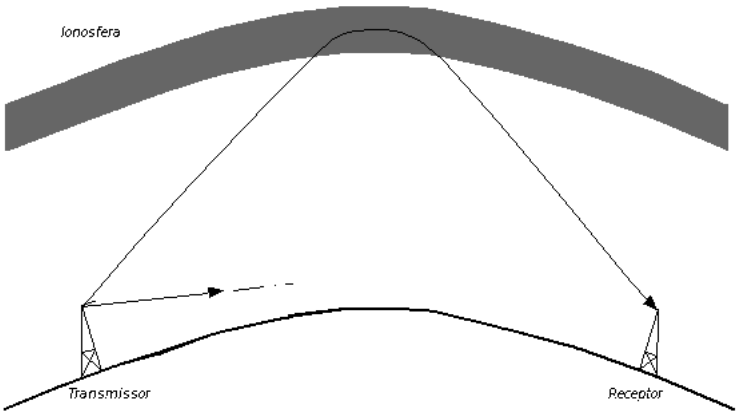


Figura 4.4: Propagação de sinais com a ajuda da ionosfera.

das camadas, para evitar flutuações nos sinais. Mas a ionosfera tem um comportamento bastante instável e isso causa erros nas transmissões de satélite também. A precisão dos sinais de posicionamento global (GPS) é limitada exatamente por isso. Erros causados por flutuações do sinal passando pela ionosfera limitam a precisão do sinal a quarenta metros. Se, por um lado, isso impede os militares de acertarem seus mísseis com a famosa precisão cirúrgica, por outro, invalida o uso do sinal para fins mais pacíficos, como na navegação para aproximação de aeronaves nos aeroportos.

4.5 Deriva iônica

A ionosfera é instável, porque o plasma se move constantemente. Essa movimentação é chamada deriva iônica (*ion drift*, em inglês). A deriva iônica pode ser vertical ou horizontal (nesse caso chamada deriva zonal). Correntes de plasma têm comportamento específico de acordo com a região do planeta, pois fluxo solar, forças eletromagnéticas e vento neutro têm influências diferentes de região para região. A figura 4.5 mostra as correntes magnéticas no período diurno sobre as Américas. Em baixas latitudes, as correntes apresentam movimentação paralela à linha do equador magnético (leste-oeste).

Duas das anomalias que aparecem na atmosfera são as chamadas bolhas de plasma e o *spread-F*. As bolhas são formadas por bolsões localizados onde o plasma é mais rarefeito do que o da ionosfera circundante. O *spread-F* consiste na difusão do plasma em uma região, o que causa a dispersão de sinais de rádio.

A deriva do plasma pode ser medida de diversas formas, entre elas: por satélites, por son-

Figura 4.5: Correntes de plasma na ionosfera. Em baixas latitudes as correntes adotam movimentação paralela ao equador magnético.

das espaciais, por fotômetros em terra, e por variações em sinais de rádio. O uso de satélites e sondas espaciais é mais recente e caro. A maioria dos dados de longo prazo vem então dos dois outros métodos. O método com fotômetros usa a movimentação das bolhas de plasma, visíveis em certas freqüências de onda, como meio de descobrir a movimentação do plasma circundante. O segundo método consiste em usar radio-observatórios para emitir sinais em freqüências específicas para a ionosfera que, quando refletidas, permitem detecção de movimento horizontal ou vertical no plasma.

Entender a forma como o plasma se movimenta, o relacionamento dos fatores que agem sobre a deriva e, finalmente, descobrir um modelo para esses fenômenos de deriva são pontos-chave para prever e corrigir interferências nas transmissões que atravessam a ionosfera [2].

4.6 Dados do problema geofísico

Os dados usados para o problema geofísico em questão foram obtidos entre os anos de 1970 e 2003 pelo rádio-observatório de Jicamarca, no Peru [2]. As informações coletadas consistem em velocidades das derivas iônicas zonais daquela região. O objetivo dos geofísicos é descobrir um modelo para a deriva zonal do plasma na região, relacionando-o com a ocorrência de anomalias específicas. As variáveis que se acreditam estarem envolvidas no problema são: a hora local, o fluxo solar, a atividade geomagnética e a sazonalidade.

Para determinação do fluxo solar diário, usou-se o índice de fluxo solar 10,7 cm (ou sfi, de *solar flux index*). Esse número corresponde ao comprimento de onda para medição do chamado "ruído solar" que indica valores aproximados para o fluxo.

Para determinação horária da atividade geomagnética, usou-se o índice Kp. O nome vem do alemão *planetarische Kennziffer* ou índice planetário. O índice Kp é gerado em Gottingen na

Alemanha, a partir de valores médios de distúrbios magnéticos coletados por treze observatórios de atividade magnética espalhados ao redor do mundo. Índices Kp menores que quatro indicam baixa atividade geomagnética, maiores ou iguais, alta atividade geomagnética (geralmente após a ocorrência de alguma tempestade solar).

A deriva zonal na região de Jicamarca tem movimento oeste durante o dia e leste durante a noite. De acordo com a convenção adotada, velocidades leste têm sinal positivo, velocidades oeste têm sinal negativo.

Às vezes, há falta de informações nos dados coletados para algumas horas no dia, e mesmo dias ou meses inteiros faltantes. Além disso, as informações estão sujeitas a certo erro de medição. Por esses motivos, os dados brutos não fornecem valores muito precisos da realidade do fenômeno observado. A figura 4.6 mostra uma seção de um arquivo de dados bruto.

```
day 860311     flux   79.
%  LT   Z.DRIFT(m/s)  Kp
15.25   -3.49     1.0
15.75    4.70     1.0
16.25    0.16     1.3
16.75    1.15     1.3
17.25    2.17     1.3
17.75   10.91     1.3
18.25   24.64     1.3
18.75   39.78     1.3
19.25   58.62     0.7
19.75   85.07     0.7
20.25  145.03     0.7
20.75  108.22     0.7
22.25  143.58     0.3
22.75  138.61     0.3
```

Figura 4.6: Seção de um arquivo de dados coletados. O dia é informado como seqüências de seis dígitos (dois para o ano, dois para o mês e dois para o dia). As colunas são: hora local (LT), velocidade de deriva zonal em metros por segundo (Z.DRIFT) e índice Kp. Aqui, para o dia onze de março de 1986, não existem dados antes das 15:25.

Para o estudo anterior envolvendo o grupo de pesquisa [2], os dados foram filtrados pelo cálculo de valores médios dentro de determinadas janelas. Quanto à atividade geomagnética, os dados foram agrupados em períodos de atividade geomagnética calma (Kp médio de 1.8 nas nove horas precedentes) e em períodos com distúrbios geomagnéticos. O objetivo disso é estudar os efeitos dos distúrbios no modelo pela comparação com o comportamento “normal”. Quanto à sazonalidade, os dados foram agrupados em três períodos: solstício de junho (maio a agosto), solstício de dezembro (novembro a fevereiro) e equinócios (março a abril e setembro a outubro). Os estudos foram realizados para três valores de fluxo solar: 90, 100 e 110 sfi. Dentro desses grupamentos, filtraram-se as médias diárias de hora em hora, de meia-noite a meia-noite.

Para os testes de modelagem com programação genética, usou-se o período de equinócios com atividade geomagnética calma. O gráfico resultante, que o modelo deve aproximar, é dado

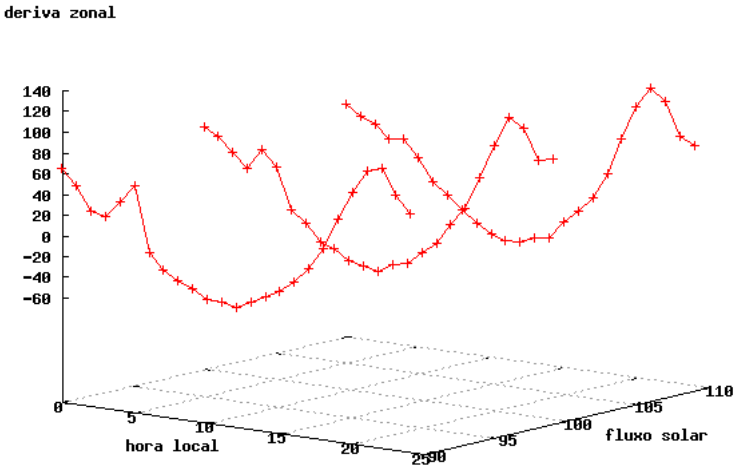


Figura 4.7: Modelo geofísico a aproximar. As três curvas representam os valores de velocidade de deriva zonal para os fluxos 90, 100 e 110 sfi durante o decorrer de dias de atividade geomagnética calma nos períodos de equinócio. Valores negativos de velocidade de deriva são direção oeste, valores positivos, direção leste.

na figura 4.7.

5 Implementação

No caminho de construção da solução, a escolha dos elementos que se adequassem às necessidades da solução e contornassem as limitações de recursos foi uma parte importante do processo. Esses elementos consistiram na linguagem de programação, ferramentas de análise auxiliar, sistemas operacionais e seus recursos e equipamento disponível. A metodologia escolhida foi bastante norteada pelo que se conseguiu extrair desses recursos.

A primeira abordagem foi a tentativa de resolver o problema com diferentes implementações do algoritmo tradicional. Mas os primeiros experimentos de regressão simbólica com os dados do problema geofísico expuseram a natureza não trivial do problema. A segunda abordagem foi a procura por uma variante de programação genética mais eficiente que a tradicional e a implementação de um sistema distribuído para tornar o processamento mais rápido.

5.1 Escolha da linguagem de programação

As linguagens de programação são como maquinário especializado, geralmente adaptam-se a tarefas ou áreas muito particulares. Implementações clássicas de programação genética foram feitas em LISP ([12], [18], [11]). Suas características únicas para tratamento de expressões tornaram seu uso comum na área de computação evolutiva. Mas, por razões já citadas em 2.15, LISP não satisfazia às necessidades do projeto.

Uma característica do LISP que seria desejável em uma possível linguagem substituta é a expressividade inerente às linguagens declarativas (paradigmas funcional e lógico). Portanto, algumas das opções cogitadas para linguagem de implementação foram dentro desses paradigmas de programação: Prolog [49], Pliant [50] e OCaml [51].

5.1.1 Pliant

Pliant [50] é uma linguagem *sui generis*. O nome, em Francês ou Inglês, significa "flexível". E o nome define plenamente sua mais forte característica: a flexibilidade. Pliant possui mais do que um compilador, ela carrega todo um conjunto de serviços na forma de aplicações *Internet* (servidor de páginas, servidor de transferência de arquivos, servidor de fórum, servidor de *e-mail* e de impressão), de um servidor de banco de dados simples e outras aplicações utilitárias. Mas todo esse pacote é extremamente compacto.

Essa característica sintética é conferida pela filosofia de sua arquitetura. Todo o conjunto de aplicações é montado em torno de um núcleo pequeno. A linguagem que compõe o núcleo tem a característica de poder ser moldada às necessidades do usuário por meio de meta-programação. o que significa que o programa consegue controle total sobre sua própria compilação. A linguagem possui um conjunto de instruções específicas para análise e compilação de expressões. Funções que usam meta-programação são chamadas meta-funções. Uma meta-função tem a capacidade de receber uma expressão em formato texto "$x+1$" e gerar sua árvore de expressão (processo denominado "parsing"), reconhecer seus componentes (operadores e argumentos), manipulá-la e, num segundo passo, gerar código executável 5.1.

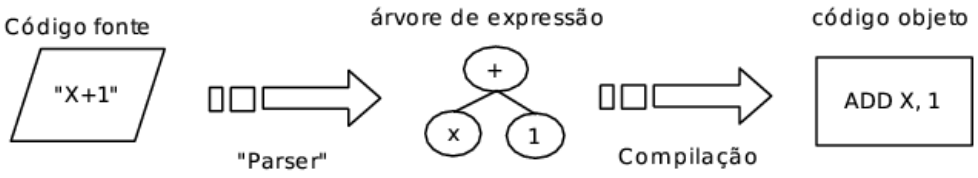


Figura 5.1: Ilustração do processo de compilação de uma expressão por uma meta-função em Pliant.

Esse tipo de compilação também pode ser feita em tempo de execução (compilação dinâmica). Essa característica, em particular, torná-la-ia interessante para o uso da linguagem em programação genética. Porém, a técnica para compilação dinâmica não é direta como no LISP, exigindo que o código seja bastante trabalhado. A compilação otimizada do Pliant normalmente gera programas mais rápidos do que LISP. Porém, compilar em tempo de execução é algo extremamente lento e cai-se novamente no problema enfrentado pelo LISP. Uma análise sobre o problema pode ser encontrada no fórum de discussão do Pliant [52].

Apesar disso, construiu-se uma implementação para testes de programação da expressão gênica em Pliant. Essa implementação é mais simples que a padrão, possuindo apenas os ope-

radores genéticos básicos e cromossomos unigênicos (compostos de apenas um gene). Uma cópia desse programa pode ser encontrada em [53].

5.1.2 Ocaml

A linguagem Ocaml possui muitas das características desejáveis à implementação da programação genética, dentre as quais se podem destacar:

- multiparadigma: pode-se usá-la como linguagem funcional, imperativa e/ou orientada a objetos.
- bom gerenciamento automático de memória;
- excelente portabilidade;
- geração de código nativo bastante eficiente (rápido);
- contém uma vasta biblioteca de recursos;
- agrega uma comunidade de usuários bastante ativa;
- possui farta documentação;
- possui recursos de compilação dinâmica.

A característica multiparadigma da linguagem permite que se testem tanto abordagens imperativas quanto abordagens funcionais para o problema. Como a comunidade de programação genética divide-se entre esses dois mundos, essa característica foi bastante utilizada.

Um bom gerenciador de memória é imprescindível para a programação genética. Durante o processo de evolução desta, a quantidade de estruturas criadas, movidas e removidas é imensa. Um gerenciador de memória ineficiente afetaria o tempo de processamento. Além disso, sem um gerenciador automático de memória, o trabalho de alocação e liberação dos recursos fica a cargo do desenvolvedor, o que representaria um contratempo considerável no processo de construção do sistema. A falta de recursos de gerência de memória é um dos pontos fracos de linguagens como C e C++, por exemplo.

A portabilidade é um ponto alto em Ocaml. O compilador está disponível para uma grande quantidade de plataformas. Mas, o mais interessante é que seus compiladores geram *código nativo* sem nenhuma necessidade de artifícios de marcação no código fonte (as chamadas diretivas de compilação) e sem grandes perdas de funcionalidade. Durante o processo de desenvolvimento, houve necessidade de se trabalhar em duas plataformas diferentes (Linux e Windows)

e, na segunda fase do projeto, a exigência de que os programas gerados executassem eficientemente em diversas plataformas.

Ao se trabalhar com estruturas de dados complexas, é importante ter um bom conjunto de recursos prontos na linguagem. Dependendo da implementação, a programação genética faz uso intensivo de listas, vetores, matrizes, *hashes* (tabelas de espalhamento) e outras estruturas de dados não triviais. Implementações eficientes desses elementos e dos mecanismos de acesso, ordenação, criação e remoção são importantes para evitar trabalho de "reinvenção de roda". Para trabalho com expressões matemáticas, é preciso um bom analisador léxico. No trabalho de carregar configurações e dados, é preciso interpretar um grande número de formatos distintos. O Ocaml possui um bom *parser* e esse recurso foi utilizado intensamente.

Somente uma documentação eficiente dos recursos da linguagem não é suficiente para um bom apoio em caso de dúvidas. Muitas vezes aparecem problemas que dizem respeito a características internas de implementação da linguagem ou seus recursos. As linguagens de programação modernas vêm acompanhadas de um grande conjunto de módulos o que as torna difíceis de serem administradas por uma única pessoa. Nesse contexto, uma base de usuários ativa, com desenvolvedores empenhados e que conheçam a fundo o funcionamento das peculiaridades de cada parte do pacote, tem a capacidade de resolver muito mais rapidamente um problema.

Existe uma extensão da linguagem, chamada Meta-Ocaml [54], para meta-programação e compilação dinâmica. As características de compilação dinâmica da linguagem foram estudadas, mas não se realizaram trabalhos mais aprofundados do uso desse recurso com programação genética.

5.1.3 Visual Prolog

Uma interface visual com capacidade de configuração autônoma é uma necessidade para a administração do sistema distribuído construído na segunda fase do trabalho. A correta configuração de parâmetros é um desafio, conforme visto na seção 2.14, mesmo em sistemas simples. Em um sistema distribuído este problema é ainda maior pois, conforme o número de máquinas aumenta, o controle sobre o processamento distribuído torna-se algo complexo.

O sistema de programação genética distribuída gera informações estatísticas sobre o processamento das ilhas de migração. Usar essas informações para guiar corretamente a busca de maneira eficiente é a proposta para uso de Visual Prolog neste trabalho.

O ambiente Visual Prolog tem mostrado sua robustez na construção de sistemas de controle de usinas nucleares, controle de tráfego aéreo de aeroportos e sistemas de escalonamento de

tarefas para a indústria. A experiência adquirida no grupo de pesquisa na implementação de exemplos para o livro Visual Prolog for Tyros [49] (documentação oficial do ambiente Visual Prolog) foi de grande valia para o presente trabalho. A proposta para documentação do sistema visual é fazer uso das idéias desenvolvidas no grupo de pesquisa e expostas no artigo disponível em [55].

5.2 Ferramentas auxiliares

5.2.1 Gnuplot

As soluções encontradas pela programação genética para problemas de regressão simbólica são difíceis de entender à primeira vista, mesmo quando o problema é simples. A programação genética é bastante profícua em gerar soluções criativas para uma aproximação [3]. Por exemplo, uma simples função quadrática, quando avaliada em uma determinada faixa do eixo das abscissas (eixo x) pode ser aproximada tanto por meio de outra função quadrática, quanto por um polinômio do oitavo grau invertido, em um processo envoltório (figura 5.2). Essa "criatividade" exige que mesmo uma boa aproximação seja analisada com maiores detalhes para saber se vale a pena evoluir no caminho indicado por esta.

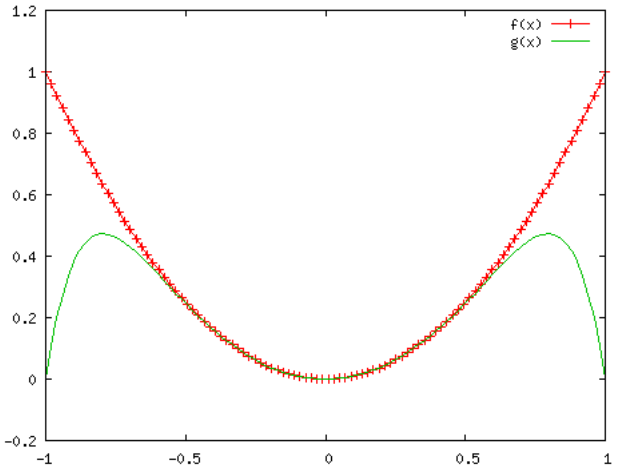


Figura 5.2: Invólucro: o polinômio do oitavo grau $f(x) = -x^8 + x^2$ é uma aproximação razoável para a função quadrática $f(x) = x^2$ no intervalo $(-0,5;0,5)$ do eixo das abscissas.

Como ferramenta para análise visual dos dados usou-se principalmente o Gnuplot [56]. O Gnuplot permite a plotagem de gráficos em três dimensões e, nas versões mais recentes, a rotação completa do gráfico de forma que é possível ter uma visão de todos os ângulos das superfícies geradas pelas funções. As figuras podem ser exportadas para diversos formatos

gráficos (inclusive os aceitos pelo processador de textos LaTeX). A maioria das figuras deste documento foi gerada pelo Gnuplot.

5.2.2 Maxima

Outros fatores importantes para tratamento das expressões encontradas são a questão da divisão protegida, a simplificação da expressão e diferentes níveis de fatoração.

A divisão protegida, usada em alguns sistemas de programação genética como forma de contornar a divisão por zero, representa um problema quando se tenta plotar diretamente a expressão[1]. Quando o programa gráfico tenta avaliar a função em pontos onde a divisão por zero aparece, o resultado é um valor indefinido.

Para resolver isso, poder-se-ia simplesmente montar um programa que avaliasse a função em diferentes pontos, que usasse a divisão protegida, gerando os dados necessários para a plotagem. Mas a questão de entender o aspecto das expressões do ponto de vista matemático, de seus componentes e repetições de padrão, induz à necessidade de simplificar as funções.

Simplificar uma expressão matemática qualquer já é algo árduo de se programar e a simplificação da maioria das soluções saídas de um sistema de programação genética é uma tarefa ainda mais complexa. As expressões são geralmente enormes, principalmente em programação da expressão gênica, em que as constantes são geradas de maneira indireta. É comum haver uma grande quantidade de trechos redundantes, freqüentemente de simplificação nada trivial.

Níveis de fatoração diferentes ajudam a entender melhor uma expressão nos seus elementos constituintes. Por exemplo, uma expressão fatorada como $(x+1)+(x+1)$ pode ser representada como x^2+2x+1. Para esse exemplo simples, o nível de fatoração não faz muita diferença, mas em expressões mais complexas isso passa a ser importante. O Maxima permite expandir de diferentes maneiras uma expressão fatorada, com escolha das variáveis ou operações sobre as quais se deva efetuar a expansão.

O Maxima é um sistema de álgebra computacional escrito em LISP [57]. Ele permite manipular expressões simbólicas e numéricas, possuindo, além de outras funcionalidades, um poderoso simplificador simbólico. A figura 5.3 mostra o uso do Maxima para simplificação de uma expressão. O Maxima pode gerar gráficos em duas e três dimensões por intermédio de uma interface com o Gnuplot.

[1] A divisão protegida retorna o valor um se o denominador é zero, e o valor da divisão caso contrário.

```
(%i38) ((((x-((x@(x@((x+x)-x)))*(((x+x)+(x*x))*((x-x)@(x-x)))))*(x@((((x+x)-
(x-x))+((x+x)@(x@x)))*(((x+x)@x)+x))))+((((x@x)*(x+(x*(x@x))))+((x-((x-x)-
(x+x)))*x))*(x@((((x@x)*(x-x))+((x-x)@x))*(((x@x)-x)@(x-x)))))+x);

                                2
                           x - x (x  + 2 x)       2
(%o38)                     ---------------- + 3 x  + 3 x
                              4 (x + 2)
(%i39) ratexpand(%o38);
                                3                    2
                             11 x      25 x       17 x
(%o39)                       ------- + ------- + -------
                             4 x + 8   4 x + 8   2 x + 4
(%i40) expand(%o38);
                         3          2
                        x        2 x          x          2
(%o40)            - ------- - ------- + ------- + 3 x  + 3 x
                    4 x + 8   4 x + 8   4 x + 8
```

Figura 5.3: Exemplo de simplificações no Maxima: na primeira linha (i38), a expressão saída do sistema de programação genética é inserida; o símbolo @ foi previamente configurado como divisão protegida; as linhas i39 e i40 mostram diferentes comandos para expansão da expressão.

5.2.3 Linux e *shell scripts*

Fazer a análise dos dados gravados nos arquivos de registro de execução dos programas é uma tarefa trabalhosa. Esses arquivos de registro (*logs* no jargão técnico) contém informações sobre as gerações como data e hora, número da geração, melhor indivíduo da geração e sua aptidão. Filtrar esses dados em um determinado grupo de gerações, selecionar as colunas adequadas e formatá-los para as aplicações gráficas e para o simplificador simbólico é um processo demorado. Felizmente, os sistemas Unix provêem uma grande flexibilidade para automatizar esse tipo de tarefa por meio de programação por *shell scripts* [58]. O *shell* é interpretador de comandos do Unix. Os *scripts* são arquivos contendo comandos do *shell* que podem ser executados como um programa. O conjunto de comandos disponível é tão elaborado que é possível construir pequenos sistemas usando-se *scripts*. Esse tipo de programação é muito usado para administrar sistemas Unix de grande porte.

Na segunda fase do projeto, havia necessidade de iniciar e parar remotamente um conjunto de unidades de processamento em diferentes máquinas espalhadas em três laboratórios diferentes. Todo o controle sobre execução de clientes para a computação distribuída através da rede dos laboratórios foi feita por *shell scripts*.

5.3 Equipamento

O equipamento dos laboratórios de Pós-Graduação é bastante heterogêneo quanto à capacidade de processamento. Há um laboratório de uso geral, onde as máquinas, exceto o servidor, não suportariam de maneira adequada, operando sozinhas, as altas demandas de processamento exigidas por experimentos mais complexos de programação genética. As máquinas dos laboratórios de uso específico para banco de dados e inteligência artificial são bem melhores, mas parte dessas máquinas, por razões justas, fica alocada a determinados projetos.

Todos esses laboratórios estão interconectados através de uma rede local. O controle de acesso às máquinas é centralizado no servidor do laboratório de uso geral. O diretório *home* dos usuários, localizado no servidor, é compartilhado remotamente através da rede de forma a ser visível por todas as máquinas. Essa configuração facilita bastante o esquema de controle remoto das máquinas para as experiências de programação distribuída.

Grande parte do poder de processamento desses laboratórios é desperdiçado. O uso das máquinas nos dois laboratórios de uso específico é bastante disputado durante o dia. Durante a noite, a maior parte das máquinas é deixada ligada, sem nenhuma atividade. A capacidade de vazão dessa rede é baixa (de apenas 10 megabits por segundo), de forma que inviabilizaria o aproveitamento para sistemas distribuídos como *clusters* de processamento. O uso de computação em rede para essas máquinas durante a noite, e mesmo durante inatividade diurna, é um bom modo de aproveitar esse período ocioso. Foi exatamente essa a idéia implementada para viabilizar maior poder de processamento para o projeto.

É possível fazer acesso a qualquer máquina dos laboratórios via *Internet*. A rede está conectada à *Internet* através da rede da Universidade. O servidor do laboratório geral permite acessos vindos da *Internet* através de conexão de terminal remoto seguro. Este recurso é de grande valia para controlar remotamente o processamento de aplicações executando durante a noite, finais de semana e feriados.

5.4 Metodologia

5.4.1 Testes comparativos

Nesse estudo, partiu-se de uma implementação mais convencional de programação genética, para estruturas menos convencionais e, em seguida, para uma variante mais recente (programação da expressão gênica).

Testes comparativos entre variantes são difíceis de serem feitos. Primeiro, porque como as diferenças na implementação das diversas variantes as tornam mais eficientes com parâmetros específicos, portanto, é inútil tentar igualar os parâmetros para comparação. Por exemplo, a programação da expressão gênica funciona melhor com populações pequenas (até 500 indivíduos), a variante de Koza exige populações maiores (mais de 1500 indivíduos). Além disso, muitos parâmetros não podem ser comparados por não se aplicar a determinadas variantes, como no caso do tamanho e profundidade das árvores geradas entre a variante de Koza e determinadas representações lineares. Em segundo lugar, porque resoluções para diferentes problemas se adaptam a configurações específicas que incluem uma grande gama de elementos: o tipo de algoritmo, os parâmetros de programação genética, o modelo de população, o modelo de computação distribuída e os parâmetros de migração. Esse ajuste de configuração, por si só é um problema complexo.

Com essa dependência de particularidades, o uso de sementes para geração de números aleatórios é útil somente em experiências de comparações com variações de parâmetros de um algoritmo em específico.

Dadas as limitações impostas pelos recursos computacionais disponíveis, pela natureza complexa da questão de ajuste de parâmetros e pelo prazo para a implementação dos testes, testar extensivamente diferentes variantes para resolução de regressões simbólicas não triviais ficou fora de questão. Por esse motivo, os testes entre as diferentes variantes foram feitos com o uso de problemas de regressão simbólica mais simples (*toy problems*). A suposição, nesse caso, é que os algoritmos mantenham um desempenho proporcional com o aumento da escala dos problemas.

5.4.2 Cálculo de esforço computacional

Para algumas dessas comparações de regressões simbólicas mais simples, será utilizado o método de cálculo para esforço computacional proposto por Koza (em [22], seção 4.11). Esse método consiste em coletar dados estatísticos de sucesso em N execuções independentes com populações de M indivíduos cada, por G gerações.

Com base nesses dados, são calculadas as probabilidades de sucesso acumulado para M indivíduos até a geração i, ou $P(M,i)$.

A probabilidade z de obtenção de sucesso até a geração i com M indivíduos em até R execuções independentes é dada por:

$$z = 1 - [1 - P(M,i)]^R \tag{5.1}$$

Koza considera $z = 99\%$ em seus experimentos. Aplicando-se logaritmos à equação, obtém-se:

$$R = R(M,i,z) = \left\lceil \frac{log(1-z)}{log(1-P(M,i))} \right\rceil, \tag{5.2}$$

que representa o número estimado de execuções independentes para sucesso com probabilidade z até a geração i com o uso de M indivíduos. O esforço computacional é medido pela quantidade de avaliações de aptidão (número de indivíduos processados) necessárias para alcançar sucesso. O número de indivíduos que devem ser processados para se obter sucesso com probabilidade z até a geração i é dado por:

$$I(M,i,z) = M(i+1)R(M,i,z), \tag{5.3}$$

onde $M(i+1)$ é o número de indivíduos processados até a geração $i+1$. É definido, então, um valor E, dado por:

$$E = I(M,i^*,z) = M(i^*+1)R(M,i,z), \tag{5.4}$$

onde i^* representa o valor de geração que minimiza a função da equação 5.3, e E é o esforço computacional estimado, em número de indivíduos a serem processados, para a obtenção de uma solução com probabilidade z.

Segundo o próprio autor do método: este valor é uma estimativa pós-processamento, dependente de todo o contexto (M, G e outros parâmetros), baseada nas estatísticas coletadas, e não representa, portanto, o mínimo esforço computacional para o problema ([22], seção 4.11).

5.5 Implementações do algoritmo

A primeira abordagem consistiu na realização de testes com os dados do sistema geofísico com o uso do algoritmo tradicional de programação genética. Para isso, construiu-se uma versão para Ocaml do programa original em LISP, chamado *littleGP*, disponível em [59]. Testou-se também uma estrutura de dados alternativa para a implementação das árvores. Alguns algoritmos rápidos de criação de árvores (PCT1 e PCT2 [19]) foram adaptados para essa nova estrutura

e fizeram-se comparações de desempenho entre as duas versões.

5.5.1 "Little Ocaml GP"

Praticamente todas as características do programa em LISP foram levadas para OCaml. Com exceção de algumas substituições, como a da função "eval" por um interpretador de expressões, o código é uma cópia do programa LISP. A idéia aqui não era criar nada novo, mas testar o desempenho do "estado da arte" em relação ao problema e familiarizar-se com os conceitos de programação genética. Outra diferença foi quanto à implementação das árvores. A estrutura foi definida por meio de um tipo algébrico recursivo [51]. A figura 5.4 mostra a definição do individuo. O indivíduo (linha seis) é formado por um programa (árvore sintática) e um conjunto de valores que definem sua aptidão (nas linhas sete a dez). O programa é uma expressão que pode ser uma variável, uma constante ou uma função (linhas dois, três e quatro, respectivamente). Uma função tem um identificador ("string") e uma série de argumentos, os quais, por sua vez, são definidos como uma lista de expressões. A árvore de programa é definida, portanto, em termos de si mesma.

```
01 type expression =
02     Var of string
03   | Const of float
04   | Fn of string * (expression list);;
05
06 type individual = { mutable program               : expression
07                   ; mutable standardized_fitness  : float
08                   ; mutable adjusted_fitness      : float
09                   ; mutable normalized_fitness    : float
10                   ; mutable hits                  : int
11                   }
```

Figura 5.4: Implementação do indivíduo para o "Little Ocaml GP"

Entre os principais elementos dessa implementação pode-se citar:

- opção para três métodos de criação de árvores citados na seção 2.7: *full*, *grow* e *ramped half-and-half*;
- opção de três tipos de métodos de seleção: dois tipos de roleta (comum e com truncamento) e torneio binário (2 indivíduos);
- operadores genéticos de reprodução (replicação), macro mutação e recombinação;
- escolha do número de indivíduos em elite;

- divisão protegida e um simplificador de expressões;

5.5.2 Código linear de Read

O formato pré-fixo é usado para representar árvores na programação genética desde as primeiras versões dos sistemas construídos em linguagens imperativas. Na representação pré-fixa tradicional, para identificar completamente o ramo a ser podado, este deve ser percorrido por inteiro (uma operação com tempo linear no número de nós). O agravante nesse caso é que, para cada nó, é preciso fazer duas leituras: a do nome da função e a da sua aridade.

O código de Read é uma notação especial para representação de árvores no formato pré-fixo. Nessa representação, armazena-se para cada nó, além do identificador da função, também sua aridade. A estrutura da árvore é representada por um vetor de aridades de suas funções no formato pré-fixo (figura 5.5). Isso diminui o tempo de poda dos ramos, pois, como a estrutura é representada por um vetor de aridades, não é necessário verificar uma tabela global para descobrir a aridade das funções e fazer o percurso adequado nos ramos. Em contrapartida, por conta da armazenagem local da aridade, a complexidade espacial aumenta.

A idéia não é nova, tendo sido apresentada já em 1997. Estranhamente, esse melhoramento trivial na estrutura e no algoritmo não parece ter sido adotado nas variantes de árvores codificadas em vetores, que parecem ter optado pela complexidade espacial menor. Praticamente não existem referências a esse tipo de codificação fora dos trabalhos de Pelikan, Kvasnicka e Pospichal [60]. As implementações em vetores pesquisadas usam uma tabela global de funções que consultam para saber a aridade da função do nó visitado e, assim, percorrer a árvore. Dessa forma funciona até hoje a versão do sistema DGPC (*Dave's Genetic Programming in C*), usada por Koza [3].

A estrutura da árvore é representada por uma seqüência da aridade de seus nós. Para montagem de uma árvore a partir do código, lê-se a seqüência de aridades da esquerda para a direita, cria-se um nó, abrem-se tantos ramos quanto for o valor da aridade e percorrem-se os ramos em pré-ordem. A figura 5.5 exemplifica uma decodificação. O código tem uma relação biunívoca com a estrutura de árvore, de forma que uma árvore pode ser representada por apenas um código e vice-versa.

Outra decisão adotada foi a de tornar o algoritmo tradicional para criação de árvores mais flexível, de forma a permitir uma melhor escolha da probabilidade de ocorrência de cada terminal e não-terminal nos programas com a aplicação das idéias expostas em [19]. Programou-se, também, um simplificador de expressões a ser aplicado de forma semelhante às idéias de

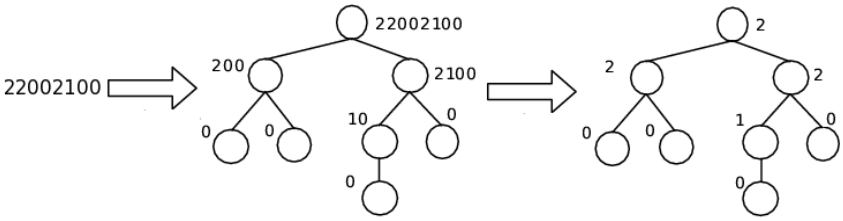


Figura 5.5: Exemplo de decodificação do código de Read.

Ekart [24], com freqüência e probabilidade de aplicação definidas como parâmetros.

5.6 Programação da expressão gênica

Dadas as interessantes características da variante de programação da expressão gênica, implementaram-se duas versões para este algoritmo: uma em Pliant e outra em Ocaml. A versão em Pliant mostrou-se bem mais lenta do que a variante em Ocaml. Todos os testes efetuados envolvendo programação da expressão gênica foram feitos em um sistema construído em Ocaml ou em sua versão de processamento distribuído.

5.6.1 Características da implementação

Todos os operadores genéticos dados na seção 3.5 foram implementados.

Apesar de terem sido implementadas outras funções de aptidão, a função usada na maioria dos testes foi a fórmula de erro absoluto dada na equação 3.2. Os testes preliminares mostraram que a velocidade do programa é bastante reduzida com o uso da equação de erro relativo, mais complexa, dada na equação 3.4.

Todos os parâmetros podem ser especificados em um arquivo texto e, no caso da versão distribuída, enviada pelo servidor aos clientes. Os parâmetros do arquivo são: método de parada, método de seleção, tempo de isolamento (para a versão distribuída), número máximo de gerações, número máximo de execuções, tamanho da população, tamanho da cabeça dos genes, número de genes por cromossomo, função de ligação, taxa de aplicação de cada operador genético, faixa de seleção e número da porta de conexão do servidor.

Implementaram-se os métodos da roleta e de torneio. Como critério de parada, usaram-se duas condições. O algoritmo encerra na descoberta de um indivíduo com aptidão máxima (dada pela fórmula em 3.3), ou no número máximo de execuções estabelecido como parâmetro.

A estrutura de dados usada para representar os cromossomos foi a de vetores de strings. A geração de constantes foi deixada a cargo do próprio algoritmo, sem geração explícita. Em comparações iniciais frente ao algoritmo tradicional usando problemas simples confirmou-se que o algoritmo era eficiente em gerar constantes de forma implícita. Não se implementou o conceito de genes homeóticos.

O sistema pode carregar casos de aptidão de arquivos texto com linhas no formato:

$$x_1, x_2, \ldots, x_n, y$$

onde os n primeiros valores representam as variáveis independentes (entradas) e a última, y, é a variável dependente (saída). Os dados podem ser enviados para os clientes pela rede ou lidos de um arquivo local.

O programa gera registros de execução em arquivo (*logs*) para cada geração, com informações sobre o número da execução, o número da geração, a listagem do melhor indivíduo em formato infixo, sua aptidão, o número de acertos com menos de 0,01 de erro e a aptidão média da população.

5.6.2 Necessidade da fase de expressão

Apesar de explicitamente delineada no algoritmo de Ferreira (seção 3.4), não parece ser necessária uma fase separada de expressão dos cromossomos para uma posterior avaliação do programa. Os dois passos poderiam ser feitos em conjunto por meio de um algoritmo simples.

A chave do procedimento para expressão e avaliação conjunta, visto no algoritmo da figura 5.6, é varrer o gene do fim para o início. Prossegue-se a varredura inserindo todos os terminais encontrados em uma fila. Ao se achar um nó não-terminal, extrai-se da fila tantos terminais quanto for a aridade do não-terminal. Aplica-se a função indicada no nó não-terminal a estes terminais e o resultado é inserido ao final da fila. O algoritmo segue assim até encontrar o início do gene.

Para se saber qual é o fim de um gene, procede-se da seguinte forma: começando com um contador cujo valor inicial é um, parte-se do início do gene e, continuando para a direita, soma-se ao contador a aridade do nó atual menos um, se o contador zerar, chegou-se ao fim do gene, caso contrário continua-se com o próximo símbolo na seqüência. O algoritmo da figura 5.7 mostra esse processo.

O conceito chegou a ser implementado em Pliant, mas o programa em Ocaml foi mantido

```
Dados: gene: um vetor contendo o gene a ser avaliado
fim : índice do último elemento da árvore de expressão
Resultado: o valor final da árvore de expressão avaliada
para i = fim até 0 faça
    se Terminal(gene[i]) então
        InsereFila(Valor(gene[i]));
    senão
        k ← Aridade(gene[i]) − 1;
        para j = k até 0 faça a_j ← RetiraFila();
        InsereFila(Aplica(Função(gene[i]),a_0,a_1,...,a_k));
    fim
fim
retorna RetiraFila()
```

Figura 5.6: Avaliação de uma árvore de expressão contida em um gene.

```
Dados: gene: um vetor contendo o gene cujo final deve ser encontrado
Resultado: o índice da última posição do gene
contador ← 1;
i ← −1;
enquanto contador ≠ 0 faça
    i ← i + 1;
    contador ← contador + Aridade(gene[i]) − 1
fim
retorna i
```

Figura 5.7: Algoritmo para encontrar o final do gene.

conforme o algoritmo de Ferreira [46], por questão de compatibilidade com o conceito tradicional da variante. Além disso, não foi estudada a fundo a questão da eficiência da segunda abordagem.

5.7 Computação distribuída

Devido à natureza complexa do problema estudado, a necessidade computacional envolvida é bastante considerável. Testes efetuados com sistemas de programação genética simples não pareceram satisfatórios, o sistema exige longo tempo de processamento, ficando preso com freqüência em máximos locais.

Na natureza, os processos genéticos são altamente paralelos, dos processos que envolvem a reprodução à síntese de proteínas. Os algoritmos evolutivos em geral, herdam essa característica dos sistemas naturais, ou seja, são altamente paralelizáveis.

Esta característica representa uma saída bastante usada para amenizar o problema da complexidade computacional da programação genética. De fato, segundo Koza [3], os primeiros resultados úteis (patenteáveis) usando programação genética foram obtidos com *clusters* de mil máquinas. Por outro lado, essa necessidade de um grande número de máquinas representa uma barreira considerável para a maioria dos pesquisadores.

5.7.1 Computação em rede

Uma solução barata é o processamento distribuído nos moldes do projeto *SETI@home*. O projeto SETI (do Inglês *Search for Extra Terrestrial Intelligence*) surgiu da idéia de que haveria grande probabilidade de que culturas inteligentes em outros planetas também utilizassem ondas de rádio para comunicação, portanto, descobrindo-se um “padrão inteligente” em ondas vindas do espaço, estaria provada a existência de vida extra-terrestre. O problema é que, além da quantidade de informação coletada pelos telescópios envolvidos no projeto SETI ser vasta, a varredura do sinal para a descoberta de um “padrão inteligente” é um problema computacional complexo. Para solucionar a questão da necessidade de processamento, surge a idéia do *SETI@home*: processamento distribuído voluntário. Os interessados em ajudar na busca baixam um programa cliente que trabalha durante o tempo ocioso da máquina. Detectado certo grau de inatividade, o programa conecta-se ao SETI, requisita um trecho de sinal, o processa e devolve o resultado.

O esquema de processamento voluntário funcionou tão bem que o sistema clássico SETI@home

foi um dos computadores com maior capacidade de processamento no mundo sustentando 60 TeraFLOPS durante anos [61]. Atualmente, os idealizadores do projeto abriram sua arquitetura de forma a permitir a criação de outros projetos usando o mesmo modelo de computação distribuída [62]. Esta arquitetura aberta recebeu o nome de BOINC (*Berkeley Open Infraestructure for Network Computing*). Hoje, o projeto BOINC envolve projetos científicos para cura de doenças, predição de clima, melhoria no projeto do acelerador de partículas do CERN, entre outros.

5.7.2 Modelo de paralelização

A aplicação utilizando o modelo de computação em rede estilo "BOINC", deve possuir alguns pré-requisitos [63]:

Paralelismo independente: a aplicação deve ser divisível em partes paralelas com pouca ou nenhuma dependência;

Baixa taxa dados/computação: como os dados de entrada e saída de uma unidade de processamento cliente são obtidas e enviadas via *Internet*, em conexões que podem ter custo alto, não se pode abusar da rede. Aplicações com altas taxas de entrada e saída devem usar *clusters* locais.

Tolerância a falhas: dados em uma rede de processamento compartilhado estão sujeitos a erros, como falhas de conexão e dados incompletos. Portanto, a aplicação deve ser altamente tolerante a falhas.

Dos modelos distribuídos disponíveis para programação genética (seção 2.17.1) o que mais se encaixa nesses requisitos é o modelo de ilhas (migração). Os outros dois modelos teriam sérios problemas quanto à taxa de transferência de dados e tolerância a falhas.

5.7.3 Topologia de migração adotada

Como a idéia é utilizar computação distribuída nos moldes de computação em rede para evoluir populações usando-se o modelo de ilhas com migração, deve-se escolher uma das topologias descritas na seção 2.18.

Pode-se optar por duas formas de implementar essas topologias de conexão. Uma delas é por meio de conexões reais entre as máquinas que evoluem as subpopulações, outra é manter um gerenciador central que emula a topologia. Optou-se pela segunda, por vários motivos:

Facilidade de implementação: uma rede do tipo ponto-a-ponto complicaria muito os protocolos de conexão;

compatibilidade com network computing : a forma de operar o controle dessas redes é centralizada;

controle de eventos : um centro de migrantes único ajuda a manter registro sobre o que está acontecendo com a evolução das subpopulações e migrações;

flexibilidade : permite uma flexibilidade maior em caso de necessidade de troca de topologia.

5.7.4 Arquitetura adotada

A arquitetura criada para o sistema distribuído é mostrada na figura 5.8. Nessa arquitetura, um servidor tem o controle sobre todo o processo de migração. Além disso, o servidor tem total controle sobre o processamento dos clientes, define sua configuração, pode definir uma população inicial, quando o processamento deve iniciar ou parar. Quando uma subpopulação atinge determinado número de gerações pré-estabelecido pelo servidor, o cliente envia-lhe seu melhor indivíduo. O servidor adiciona esse indivíduo ao seu *pool* (reservatório de migrantes) e espera até que os demais clientes conectados enviem-lhe também. No momento em que o *pool* está cheio (quando todos os clientes migraram seus indivíduos), o servidor faz um sorteio para cada cliente nos moldes da figura 2.12 e envia as migrações aos clientes.

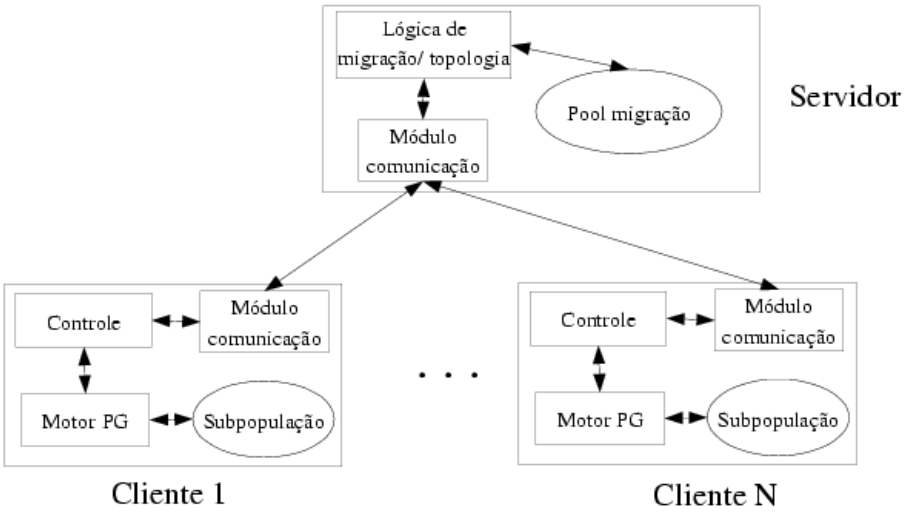


Figura 5.8: Arquitetura escolhida para o sistema distribuído.

Para a topologia lógica de migração emulada pelo servidor foi escolhida a totalmente conectada por questão de praticidade, mas qualquer outra topologia poderia ser implementada

sem grande necessidade de alterações, apenas com uso de vetores ou matrizes para controle de vizinhança.

5.7.5 Protocolo de comunicação

O diagrama de seqüência ilustrado na figura 5.9 mostra o protocolo de comunicação entre cliente e servidor. Ao receber um pedido de conexão, o servidor verifica a compatibilidade da versão do cliente com seu protocolo; caso não seja compatível, a conexão é abortada. Uma vez estabelecida a conexão, o cliente requisita a configuração inicial, e, em seguida, os dados (casos de aptidão) para trabalho. Recebidos os dois pacotes, o cliente responde ao servidor que está pronto esperando pela ordem de início. Esse estado de pronto em espera é necessário em vários pontos do processamento, como depois de uma ordem de parada enviada pelo servidor ou depois de finalizado o processamento. O servidor envia ao cliente uma ordem para iniciar processamento e logo em seguida requisita um indivíduo para o *pool*. O cliente responderá a essa requisição apenas após um determinado período de isolamento (certo número de gerações). O período de isolamento é definido pelas configurações que recebeu do servidor. A comunicação entra então em um ciclo em que o cliente envia um indivíduo para migração após cada período de isolamento. O cliente pode receber um indivíduo do servidor a qualquer momento após enviar o primeiro indivíduo. A incorporação das migrações à subpopulação local só acontece entre gerações, quando o motor de programação genética verifica se existe um indivíduo imigrante e o substitui pelo seu pior indivíduo se aquele for melhor que este.

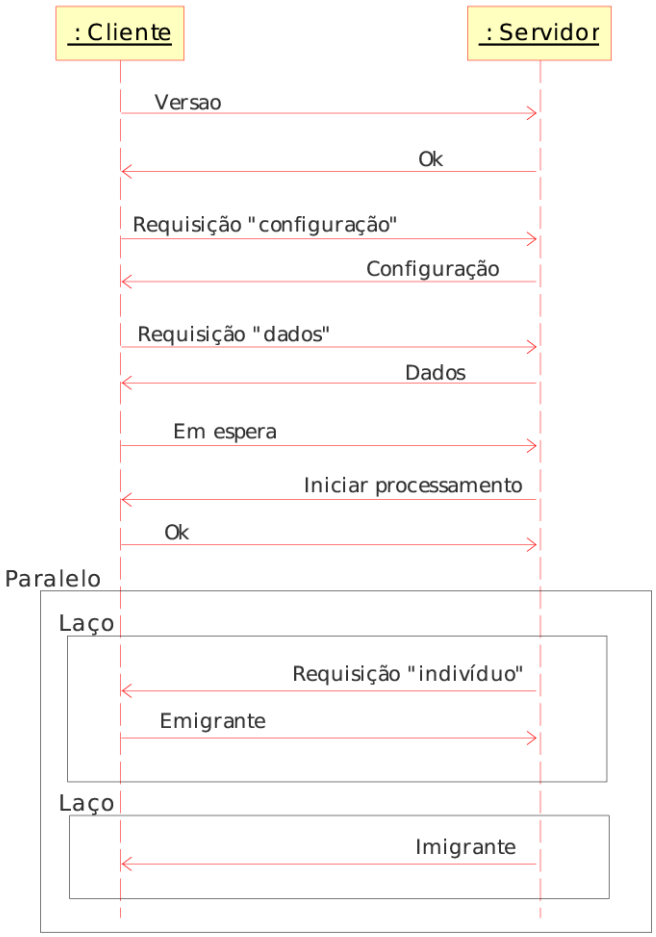


Figura 5.9: Protocolo de comunicação entre cliente e servidor. Os laços de emigração e imigração ocorrem em paralelo: após envio do emigrante, o cliente pode receber um imigrante a qualquer momento.

6 Resultados

6.1 Experimentos com o algoritmo tradicional

Os testes com os sistemas da modalidade tradicional de programação genética foram praticamente todos feitos tendo como casos de aptidão os dados do problema geofísico. O objetivo era testar à exaustão a capacidade de modelagem dos sistemas para o problema e, dessa forma, ambientar-se com a ferramenta. Algumas das questões observadas foram tempo de processamento, melhores parâmetros e desempenho das duas versões implementadas. Todos os testes foram feitos em microcomputadores isolados no laboratório de inteligência artificial em execuções seqüenciais, não distribuídas. A maior parte dos testes foi feita com a máquina de melhor configuração na época: *Intel Pentium IV, 2,4 GHz,* com 256 *megabytes* de memória. O sistema operacional usado foi o *Slackware Linux, kernel* 9.1.

6.1.1 Comparação entre as implementações

Para comparações entre as duas versões, *Little GP* e Read, fizeram-se alguns experimentos tentando igualar da melhor maneira os parâmetros. As comparações de desempenho em termos aptidão *versus* número de gerações entre as duas versões mostraram um desempenho bastante parecido, com o algoritmo tradicional ganhando com pequena vantagem. No entanto, ao se usar simplificação com a versão Read o desempenho se igualou. Esse comportamento é exemplificado pelos resultados mostrados na figura 6.1. Os parâmetros usados nesse teste são os da tabela 6.1. A função de aptidão utilizada foi a de erro quadrático normalizado e como método de seleção usou-se a roleta simples. Na execução com simplificação, este operador foi usado a cada dez gerações e a probabilidade de aplicação foi de 0,01.

Em geral, a aproximação de um modelo para o problema geofísico é muito lenta com o algoritmo tradicional de programação genética, chegando a executar por 72 horas para gerar aproximações razoáveis. Esperava-se um tempo de processamento mais baixo para a versão de Read, mas o que ocorreu foi o contrário. O problema é que a versão de Read tem a tendência

Tabela 6.1: Configuração de parâmetros para testes de comparação.

Parâmetro	Valor
Funções	$+, -, \times, /$
Tamanho da população	1000
Número de gerações	5000
Probabilidade de mutação	0,1
Probabilidade de recombinação	0,8
Probabilidade de replicação (reprodução)	0,1
Número máximo de nós na árvore	1024
Elitismo (número de indivíduos)	2

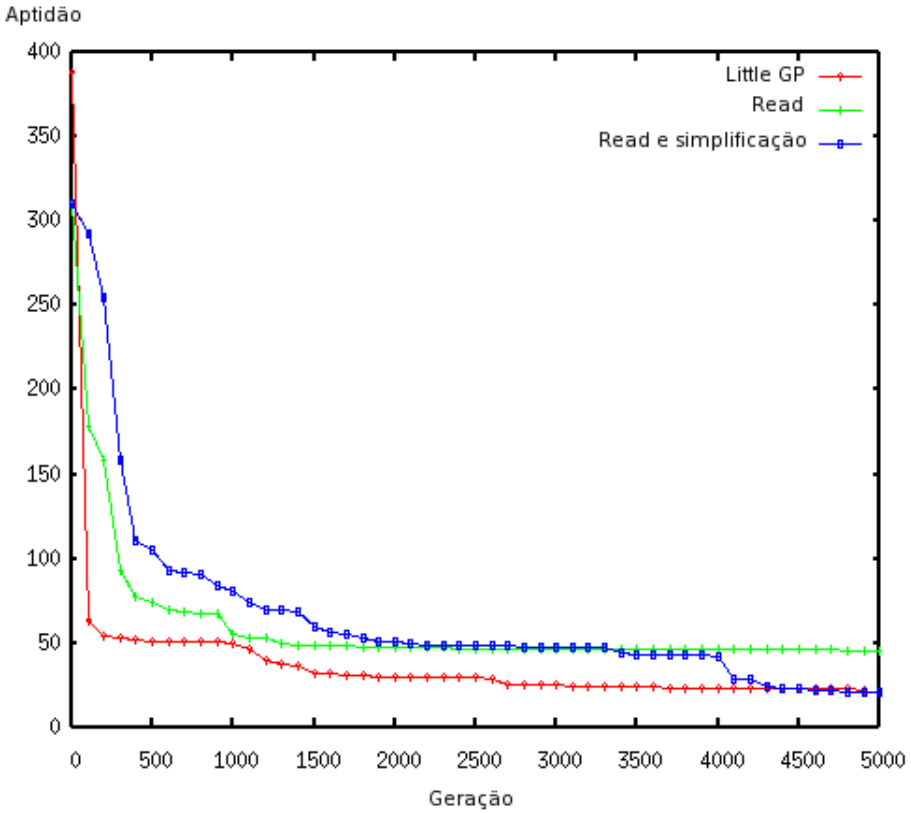


Figura 6.1: Comparação de aptidão *versus* número de gerações entre os programas: Little GP, Read e Read usando simplificação.

de gerar árvores cheias para os programas iniciais e isso o torna lento logo de início. A versão *Little GP*, com o uso do algoritmo "grow", cria indivíduos com menor número de nós para a população inicial e segue aumentando o tamanho dos indivíduos conforme o processamento avança. No experimento mencionado, a diferença de tempo chegou a sete horas para o mesmo problema.

A presença do fenômeno de inchaço é mais evidente na versão original do algoritmo. O gráfico da figura 6.2 mostra o aumento do tamanho do melhor indivíduo em número de nós para o programa *Little GP* no decurso das gerações para o mesmo experimento. Traçando um comparativo com seu gráfico de evolução de aptidão na figura 6.1, percebe-se o aumento excessivo do tamanho do melhor indivíduo a partir da geração 1000 com pouca melhora na solução.

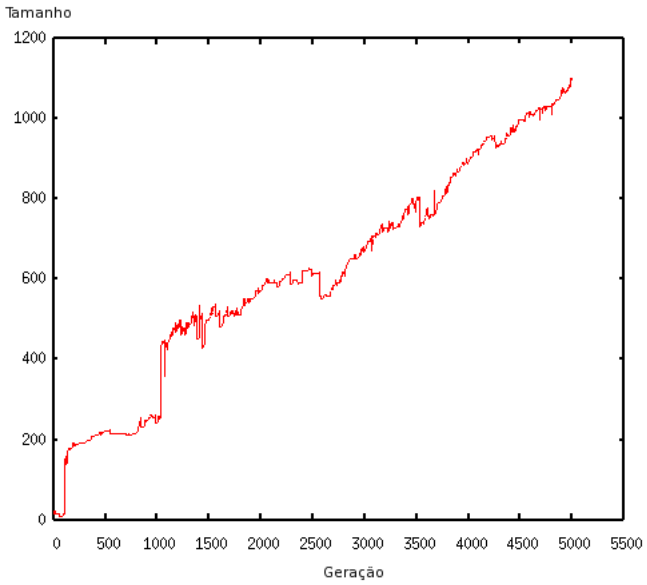


Figura 6.2: Tamanho do melhor indivíduo do programa "Little GP" no decorrer de gerações.

6.1.2 Testes com o operador de simplificação

Para verificar o comportamento dos sistemas com o uso de diferentes parâmetros usou-se um artifício pelo qual múltiplas execuções seqüenciais eram programadas através de um *script* com pequenas mudanças nos parâmetros a cada chamada de execução. Testes feitos desta forma com a simplificação mostraram que ela pode ser um operador com certa importância no processo de evolução para o algoritmo tradicional com o código de Read. Outro fator interessante foi que a resolução do problema evoluía melhor com pequenas populações. Testaram-se várias configurações mudando a freqüência e probabilidade de aplicação de simplificação com uma

população pequena (150 indivíduos). A tabela 6.2 mostra os parâmetros usados para os testes e os resultados encontrados. Usando-se a melhor dessas configurações e estendendo o número de gerações para 20.000, conseguiu-se o melhor indivíduo dentre as tentativas. Este indivíduo é mostrado na figura 6.3. Essa é a versão simplificada da expressão (depois de passar pelo Maxima). O formato de impressão está em modo matemático compacto, para economia de espaço. A aproximação não é ideal, não apresentando erro absoluto abaixo de $0,05$ com um erro médio de $8,47$.

Tabela 6.2: Testes com o uso do operador de simplificação com os parâmetros gerais: tamanho da população de 150 indivíduos, tamanho máximo de 1024 nós para o indivíduo, 300 gerações, probabilidade de mutação de 5%, probabilidade de recombinação de 80% e probabilidade de replicação de 15%.

Execução	1	2	3	4	5	6	7	8	9	10
Probabilidade	0,20	0,20	0,20	0,20	0,20	0,30	0,50	1,00	0,30	0,50
Freqüência	1	2	4	10	50	1	1	1	2	2
Melhor resultado	110,78	168,07	56,24	221,16	95,19	116,14	175,41	214,89	105,35	162,93
Execução	11	12	13	14	15	16	17	18	19	20
Probabilidade	1,00	0,30	0,50	1,00	0,30	0,50	1,00	0,30	0,50	1,00
Freqüência	2	4	4	4	10	10	10	50	50	50
Melhor resultado	119,69	115,88	106,82	169,93	153,14	150,78	139,27	185,03	132,61	242,95

```
23.66561049598496*(-f-6.15510011874)*t*((-(16.0132755853-t*(2*t+f)
   )*(-t**2/(6.187986341*(t+7.86794002178)+t-2*f)-2.90948761989)+1
   3.2617245503*t**2+5*t-2*f*(823.239765634/(4.6543288907*(t-2.909
   48761989)-f+16.528644192862)-8.73800419632)+9050.95587903632)/(
   t*(-1.75892965566098*((t-35.16474646812966*f*(t-2.9313693074)/t
   -61.317605443)*(t-.1325900711741921*((.1170635703003395*(5*f**2
   -716.15465278)/(t-.1964334747309882*t**2)-2*f)/f-t+339.71608666
   7))/(-.0112751486851512*(2*f-2.90948761989)*t**2/f-(4.632555243
   08-t)*t-7.54204286297)+t)/((t-6.99948761745)*t*(4.6543288907*(5
   .48720742757-t*(8.28078603028*(2*f-t)-(f-2.42082249468)*t**2))+
   t+11.2673040762))+.9289983327907188*(-t+2*f+2.9313693074)*t**2/
   ((2*f-8.73800419632)*t+f-8.73800419632)-4.50829095248))+f*t**2-
   6.99948761745*t-f-1077.591411994)/(f*(t*(4*t**2+(t-0.7979119341
   03)*t+3*t+2*f-16.0132755853)-.1560428548103749*(f*(t-(-7.999487
   61745*t-(f-1.28413095684)*f)/f)*(f*(-.0013963464401413*(.025916
   9256704457*t**2*(13.2617245503-2.381714363657553*(t-6.408495930
   27)*t)*(t**2+t**2*(t-2.08865689682*f/t))/(239.09518554-.1822420
   626884973*t**2)+8.73800419632*t**2*(2*t+f-117.9656265673)+(4*f+
   370.252241352)*t+137.142834558*t)-2.88099015031189*f*t*(t-t*(6.
   06211791645*t+8.13394438739))/(237.763735685/(137.142834558*t/f
   -6.40849593027)+f*t**2-16.0132755853))+1218.39818844)+f)/(-t+(f
   -4.50829095248)*(t-13.2617245503)-3*f)))+(f*(t+2*f)+2*f+435.289
   053064)/((f-28.3520384207)*(t**2+4.56268431429))
```

Figura 6.3: Melhor indivíduo encontrado com o algoritmo tradicional: t é a hora local (tempo) e f é o fluxo solar. O operador ** representa a exponenciação.

6.1.3 Problema com superfícies irregulares

As duas implementações, apesar de conseguirem expressões que gradualmente se aproximam dos pontos da curva esperada para o modelo geofísico, apresentaram um problema quanto

à suavidade das superfícies geradas por essas expressões. A suavidade da curva é um fator esperado no modelo e isto foi atingido no trabalho anterior realizado pelo mesmo grupo de pesquisa com o uso de polinômios de Bernstein [2]. Já com o uso de programação genética, entre as melhores aproximações conseguidas, a maioria apresentava pontos fora da curva esperada entre os pontos de avaliação. Um bom exemplo é o do indivíduo encontrado no experimento anterior (figura 6.3), cujo gráfico é mostrado na figura 6.4. As cruzes representam os casos de avaliação (os pontos a aproximar), as linhas ligam os pontos encontrados pelo sistema de programação genética (a linha apenas une os pontos para um melhor entendimento, não representando continuidade). Porém, quando o gráfico da função é plotado com maior definição, sua superfície revela irregularidades entre os pontos de avaliação. As figuras 6.5(a) e 6.5(b) mostram a superfície gerada pela mesma expressão.

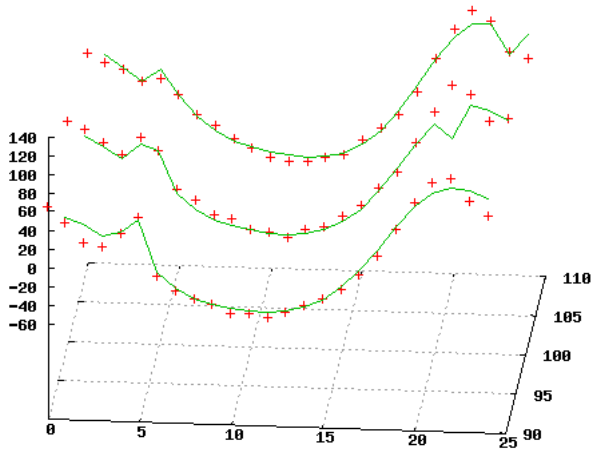
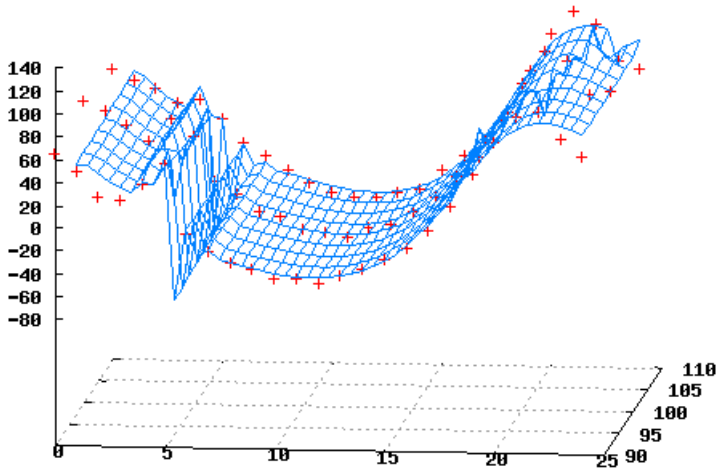
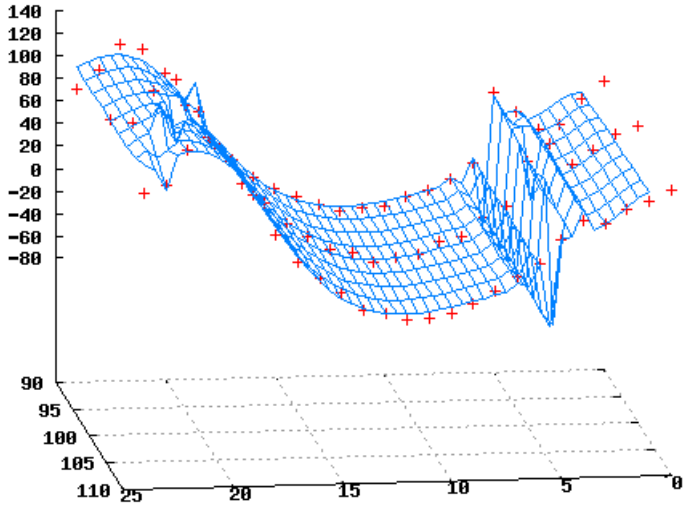


Figura 6.4: Aproximação parcial para os dados geofísicos com valores restritos apenas aos pontos de avaliação. As linhas identificam o modelo aproximado (ligam os pontos da aproximação), as cruzes indicam os pontos dos casos de avaliação.



(a) Visão “de frente” da superfície



(b) Rotação de 180 graus

Figura 6.5: Visões de diferentes ângulos da superfície gerada pelo indivíduo da figura 6.3.

6.2 Programação da expressão gênica

Com programação da expressão gênica resolveu-se usar uma abordagem diferente, partindo-se de experimentos mais simples para os mais complexos. Foram usados alguns experimentos da literatura da área para comparações com o algoritmo tradicional de programação genética. A maioria dos experimentos foi executada com o uso do sistema distribuído. A melhor das máquinas utilizadas foi um *Intel Pentium IV 2.4GHz* biprocessado, com 512 *megabytes* de memória. As piores máquinas eram *Pentium III*, com *256 megabytes*. Por essa diferença nas configurações, o tempo de processamento, em algumas máquinas, foi bem mais longo do que para outras. Durante o período de testes, as máquinas adequadas e disponíveis para uso foram no máximo oito, sendo quatro de configuração superior (Pentium IV) e quatro inferiores (Pentium III). Muitas vezes o processamento em uma das máquinas era interrompido por alguma falha do computador, como por exemplo superaquecimento. A maioria dos testes foi feita sem migração, como execuções independentes, pois as condições ideais para processamento distribuído com programação da expressão gênica ainda estão sendo estudadas. Portanto, quando os parâmetros do experimento informam 500 execuções, em geral isto significa cinco máquinas configuradas com 100 execuções cada. Os experimentos mostrados aqui foram executados durante o período noturno ou aos finais de semana.

6.2.1 Experimentos com polinômios de graus cinco e seis

Em [22] é proposta uma série de problemas para mostrar o uso de funções definidas automaticamente. O pressuposto é de que o uso dessas funções somente é válido para problemas de complexidade maior. Problemas simples não se comportam bem, em se tratando de esforço requerido, quando usam funções definidas automaticamente, tendo eficiência melhor quando não as usam.

Não se tratará aqui do mesmo assunto, mas será usado o exemplo com regressão de polinômios para uma comparação de eficiência com o algoritmo tradicional nos moldes definidos na seção 5.4.

A primeira expressão a ser usada no experimento é um polinômio de grau cinco, dado abaixo.

$$x^5 - 2x^3 + x \tag{6.1}$$

Para essa equação, o experimento exposto na seção 5.1 de [22] encontra um esforço computacional $E = 396000$ com o uso de funções definidas automaticamente e $E = 1200000$ sem

funções definidas automaticamente. Para este experimento, foram usados os parâmetros dados na tabela 6.3.

Tabela 6.3: Parâmetros usados na reprodução dos experimentos com os polinômios de graus cinco e seis.

número de execuções	500
número de gerações	51 (0 a 50)
tamanho da população	200
tamanho do cromossomo	99
genes por cromossomo	3
taxa de replicação	0,001
taxa de recombinação em um ponto	0,3
taxa de recombinação em dois pontos	0,3
taxa de mutação	0,044
taxa de inserção de seqüência de transposição	0,1
taxa de inserção de seqüência de transposição na raiz	0,1
taxa de transposição de gene	0,1
taxa de recombinação de gene	0,1
taxa de inversão	0,1
faixa de seleção	1000
método de seleção	torneio
número de casos de aptidão	50 pontos entre -1 e 1
conjunto de funções	+, *, - e % (divisão protegida)

Com base nos valores estimados de acerto para quinhentas execuções, pode-se montar uma tabela de desempenho pelas fórmulas dadas na seção 5.4. A tabela 6.4 mostra os valores encontrados.

As curvas de desempenho correspondentes, com probabilidade de acerto acumulada para o número de gerações e número de indivíduos a serem processados, é vista no gráfico da figura 6.6.

Para esse problema, o algoritmo de programação gênica apresenta-se dez vezes mais eficiente do que o algoritmo tradicional. O esforço computacional exigido (36400 avaliações de aptidão) é menor do que o resultado obtido pelo algoritmo tradicional (396000 avaliações).

Para o polinômio de grau seis, mostrado pela equação 6.2, foram usados os mesmos parâmetros. O gráfico de desempenho é mostrado na figura 6.7.

$$x^6 - 2x^4 + x^2 \tag{6.2}$$

O valor de esforço computacional obtido para esse problema com o algoritmo tradicional foi 1176000. No experimento com programação da expressão gênica, conseguiu-se E=129000, nove vezes mais eficiente.

Tabela 6.4: Valores estimados de probabilidade acumulada $P(M,i)$, número de execuções exigidas $R(M,i,z)$, número de indivíduos processados por execução até a geração i, $M(i)$, e número de indivíduos a processar $I(M,i,z)$ para uma probabilidade de acerto de 99 % para o problema do polinômio de grau cinco. O valor em negrito (36400) representa o valor estimado de esforço computacional E para um resultado satisfatório.

Geração i	$P(M,i)$	$R(m,i,z)$	M(i)	$I(M,i,z)$
0	-	-	200	-
1	0%	-	400	-
2	0,4%	1149	600	919200
3	1,6%	286	800	286000
4	3,6%	126	1000	151200
5	5,8%	78	1200	109200
6	7,6%	59	1400	94400
7	10,4%	42	1600	75600
8	13,4%	33	1800	66000
9	15,2%	28	2000	61600
10	18,6%	23	2200	55200
11	21,4%	20	2400	52000
12	24,8%	17	2600	47600
13	27%	15	2800	45000
14	29,2%	14	3000	44800
15	32,2%	12	3200	40800
16	34%	12	3400	43200
17	37%	10	3600	38000
18	39,4%	10	3800	40000
19	41,4%	9	4000	37800
20	42,8%	9	4200	39600
21	44,6%	8	4400	36800
22	45,8%	8	4600	38400
23	46,2%	8	4800	40000
24	48,4%	7	5000	**36400**
25	49,8%	7	5200	37800
26	50,2%	7	5400	39200
27	51,6%	7	5600	40600
28	53,2%	7	5800	42000
29	53,4%	7	6000	43400
30	53,6%	6	6200	38400
31	54,6%	6	6400	39600
32	54,8%	6	6600	40800
33	55,8%	6	6800	42000
34	56,2%	6	7000	43200
35	57%	6	7200	44400
36	57,2%	6	7400	45600
37	58,2%	6	7600	46800
38	59,2%	6	7800	48000
39	59,4%	6	8000	49200
40	59,6%	6	8200	50400
41	60,4%	5	8400	43000
42	61,2%	5	8600	44000
43	61,2%	5	8800	45000
44	62%	5	9000	46000
45	62,2%	5	9200	47000
46	62,6%	5	9400	48000
47	62,6%	5	9600	49000
48	63%	5	9800	50000
49	63%	5	10000	51000
50	63,2%	5	10200	52000

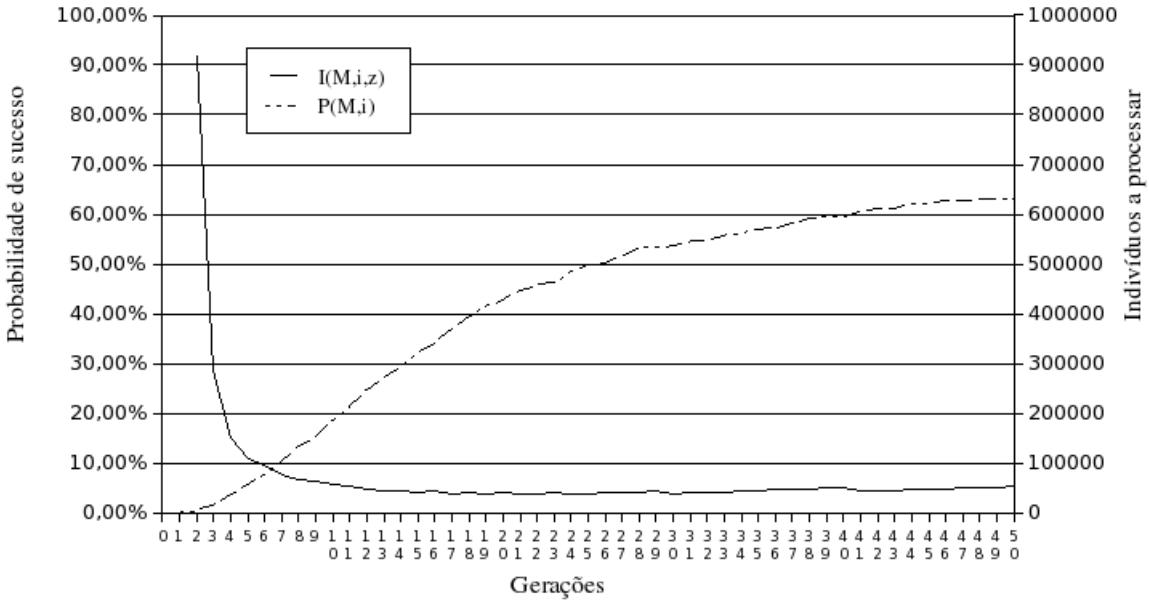


Figura 6.6: Curvas de desempenho para o polinômio de grau cinco. A curva tracejada representa a probabilidade de sucesso acumulada $P(M,i)$ e a curva com linha contínua representa o número de indivíduos a processar $I(M,i,z)$. O valor de *i* que minimiza a função é a geração 24 (E=36400).

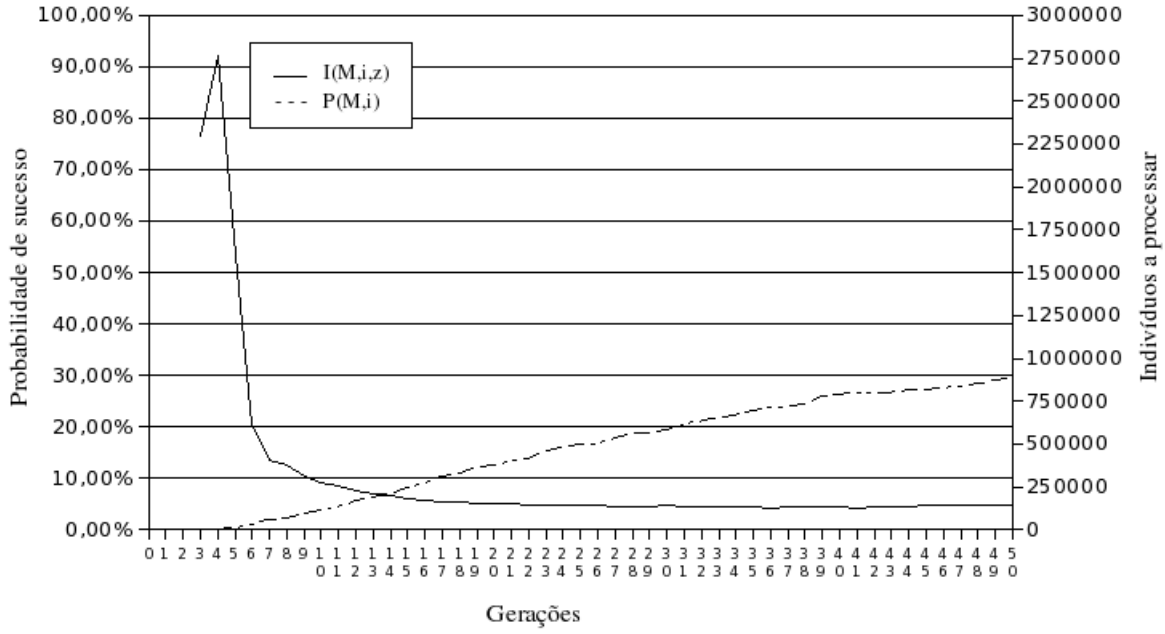


Figura 6.7: Curvas de desempenho para o polinômio de grau seis. A curva tracejada representa a probabilidade de sucesso acumulada $P(M,i)$ e a curva com linha contínua representa o número de indivíduos a processar $I(M,i,z)$. O valor de *i* que minimiza a função é a geração 42 (E=129000).

6.2.2 Funções com curvas irregulares

O algoritmo de programação da expressão gênica também pode incorrer em erros de julgamento induzidos por funções que se aproximam apenas nos pontos de avaliação. Por exemplo, no polinômio dado pela expressão abaixo (6.3), proposto em [48]:

$$f(x) = x^3 - 0,3x^2 - 0,4x - 0,6 \tag{6.3}$$

Duas aproximações obtidas nos experimentos feitos com o sistema implementado no presente trabalho são dadas pelas expressões 6.4 e 6.5.

$$g(x) = x^3 - \frac{13x^2}{45} - \frac{2x}{5} - \frac{8}{9} \approx x^3 - 0,289x^2 - 0,4x - 0,888 \tag{6.4}$$

$$h(x) = \frac{(-x + \frac{0,5}{x} + 1)(2x^3 - x)}{1 - 2x} + \frac{x^3 - x^2 - 2x}{5x - 1} - \frac{0,5x}{x^3 + x} + x \tag{6.5}$$

A expressão 6.5 tem valor de aptidão 20998,4523687 e seu gráfico é mostrado na figura 6.9. A expressão 6.4 tem um valor menor de aptidão (20993,2888889), mas percebe-se tanto por sua expressão simplificada quanto pelo gráfico (figura 6.8) que essa é uma solução mais exata.

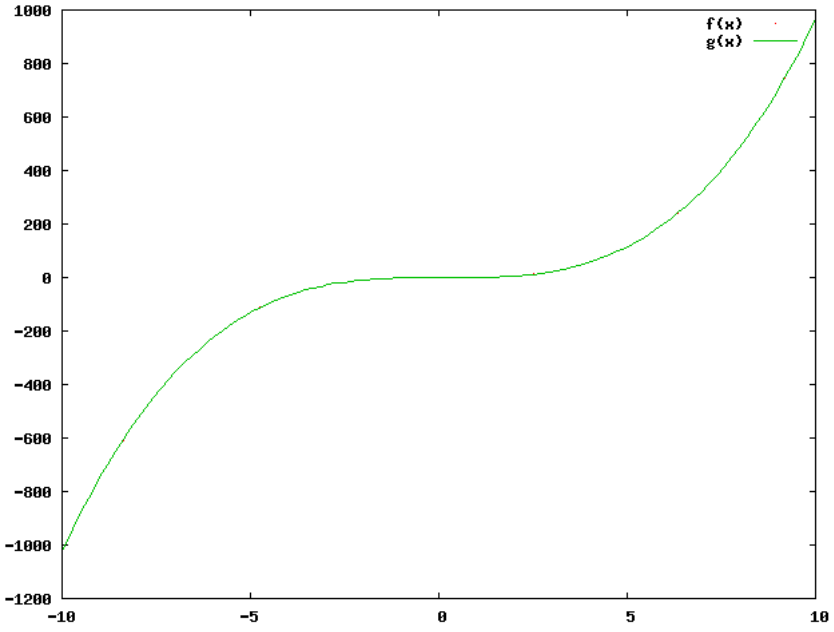


Figura 6.8: A função $f(x)$ dada na equação 6.3 é aproximada de forma quase perfeita pela equação 6.4, $g(x)$. O pontilhado da função $f(x)$ praticamente desaparece sob $g(x)$.

O que acontece é que o pequeno pico no centro do gráfico da figura 6.9 situa-se exatamente

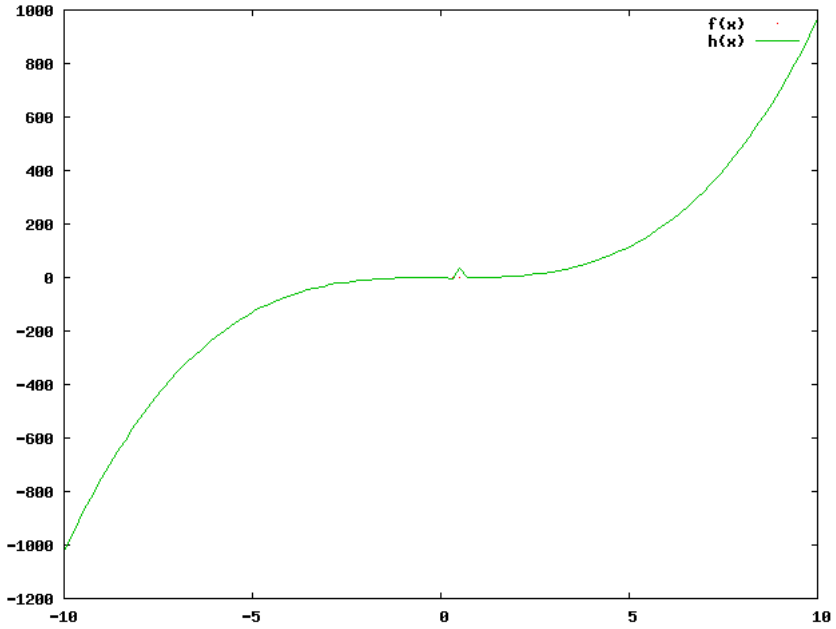


Figura 6.9: A função $f(x)$ dada na equação 6.3 é aproximada pela equação 6.5, $h(x)$. No centro há um pequeno pico fora da curva.

entre os pontos de avaliação. Esse trecho não conta no cálculo do erro para a função e, no restante dos pontos, a avaliação é superior à da função "correta". Trata-se de um problema diferente do mostrado na seção 5.2.1, pois naquele caso a função "errada" representa uma aproximação aceitável dentro do intervalo $(0,5;0,5)$. No caso mostrado acima, a função não é uma boa aproximação contínua, apesar de representar melhor aproximação discreta para os pontos de avaliação.

6.2.3 Regressão simbólica para um problema físico simples

Para pôr à prova a capacidade de o sistema encontrar um modelo para um sistema da Física, foi testada a regressão simbólica para um problema simples de lançamento vertical no vácuo. Segundo as leis da cinemática, o movimento retilíneo de um corpo sob a ação de uma aceleração constante, desprezando-se o atrito, pode ser descrito pela fórmula:

$$S(t) = S_0 + V_0 t + \frac{at^2}{2}, \tag{6.6}$$

onde $S(t)$ é a posição do corpo em movimento no momento t, S_0 é o ponto de partida, V_0 é a velocidade do corpo em S_0 e a é a aceleração constante aplicada ao corpo.

Para este problema, adotou-se o lançamento de um objeto para cima com uma velocidade $V_0 = 25m/s$, tomando-se o referencial do chão como origem e ponto inicial do lançamento

$S_0 = 0$, e a orientação do sistema como positivo para cima. Dada uma aceleração aplicada ao corpo pela gravidade com o valor de $-9,8m/s^2$, tem-se então, pela substituição desses valores na equação 6.6:

$$\begin{aligned} S(t) &= 25t + \frac{-9,8t^2}{2} \\ &= 25t - 4,6t^2 \end{aligned}$$

A princípio, uma fórmula simples a ser encontrada pelo sistema. A tabela 6.5 mostra os parâmetros usados no experimento.

Tabela 6.5: Parâmetros usados no experimento com movimento retilíneo uniformemente variado.

número de execuções paralelas	9
número de execuções seqüenciais	100
número de gerações	200
tamanho da população	100
tamanho do cromossomo	99
genes por cromossomo	3
taxa de replicação	0,001
taxa de recombinação em 1 ponto	0,3
taxa de recombinação em 2 pontos	0,3
taxa de mutação	0,044
taxa de inserção de seqüência de transposição	0,1
taxa de inserção de seqüência de transposição na raiz	0,1
taxa de transposição de gene	0,1
taxa de recombinação de gene	0,1
taxa de inversão	0,1
faixa de seleção	1000
método de seleção	torneio
número de casos de aptidão	50 pontos entre t=0.1 s e t=5 s
conjunto de funções	+, *, - e % (divisão protegida)
critério de parada	número máximo de execuções

Fórmulas aproximadas nos moldes da expressão abaixo (6.7), são encontradas rapidamente. Mas há alguma dificuldade de se obter a constante adequada para a aceleração da gravidade ($9,8m/s^2$), de forma que o número de acertos de pontos dentro de uma faixa de erro absoluto de $0,01$ é baixo.

$$5t^2 + 25t \tag{6.7}$$

O gráfico da figura 6.10 mostra os pontos dos casos de avaliação e a aproximação dada pela expressão 6.7.

Muitas vezes, o sistema tenta compensar o erro na constate de aceleração com a agregação de trechos manipulando a variável t (que representa o tempo). Nesse contexto, a fórmula apro-

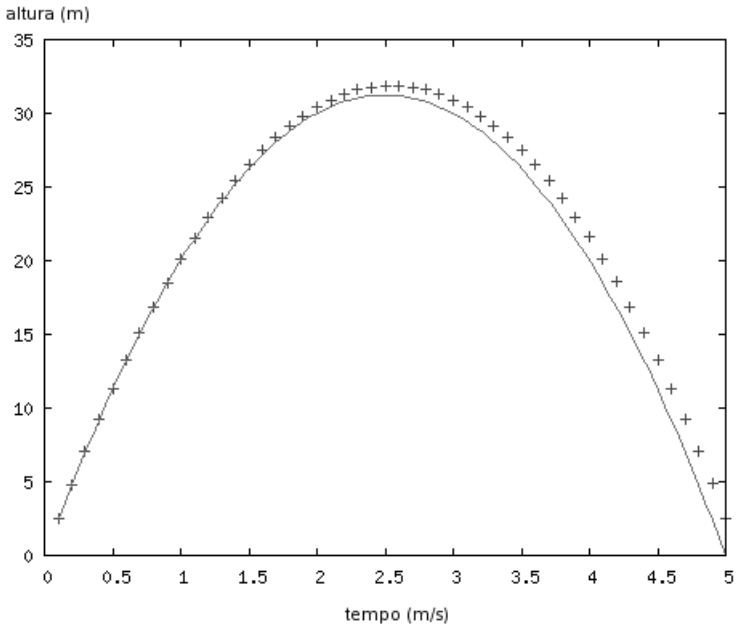


Figura 6.10: Lançamento vertical no vácuo. O sistema a aproximar está demarcado pelas cruzes e aproximação dada pela função $5t^2 + 25t$ é representada pela linha contínua.

ximada 6.7 é quase sempre o núcleo da expressão encontrada. Exemplos são mostrados pelas expressões 6.8 e 6.9 (o "núcleo" da fórmula está em negrito).

$$\frac{t^2 - t}{2t - 1} - \mathbf{5t^2} + \mathbf{25t} \tag{6.8}$$

$$4(5t - t^2) + \frac{t^2}{2t + 3} - t^2 + 5t \;=\; \frac{t^2}{2t + 3} - \mathbf{5t^2} + \mathbf{25t} \tag{6.9}$$

Apesar de esses artifícios encontrados pelo sistema melhorarem a aptidão da solução, a maioria não tem nenhum significado físico. Isso acaba obrigando a verificação das expressões uma a uma para saber se aproximações parciais têm alguma valia.

Uma aproximação total, com acerto nos cinqüenta pontos, é encontrada com bastante dificuldade. As estatísticas coletadas durante o experimento mostram um esforço computacional de 32721800. A figura 6.11 mostra as curvas de desempenho para o problema com os parâmetros especificados.

Testes com o aumento em número de gerações, do tamanho da população, e do número de genes por cromossomo não resultaram em desempenho melhor.

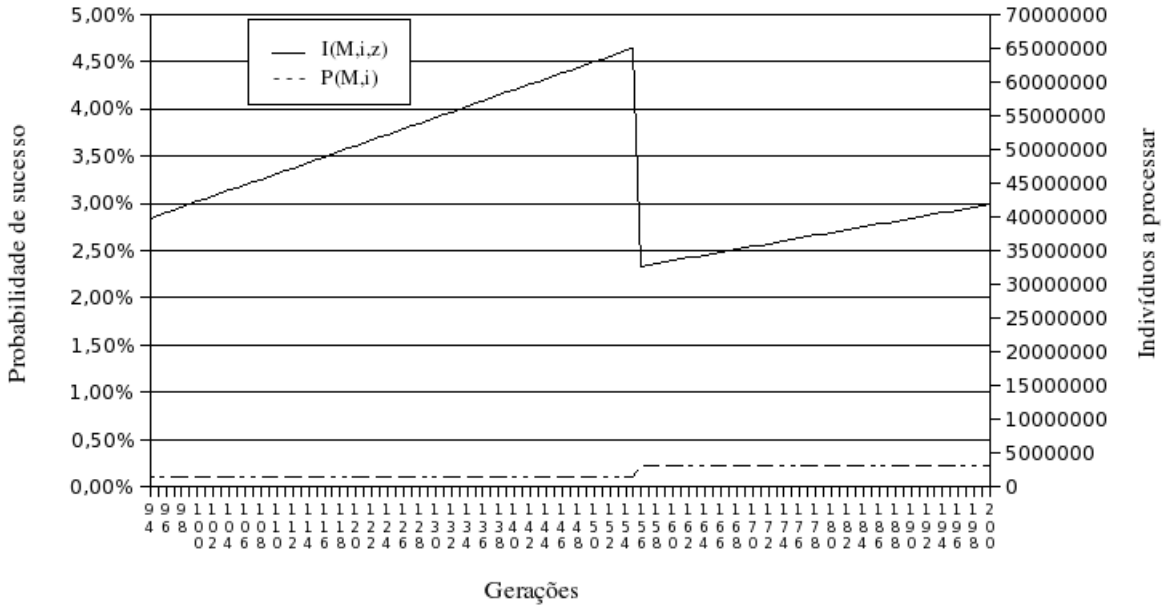


Figura 6.11: Curvas de desempenho para o problema de lançamento vertical no vácuo. A curva tracejada representa a probabilidade de sucesso acumulada $P(M,i)$ e a curva com linha contínua representa o número de indivíduos a processar $I(M,i,z)$. O valor de *i* que minimiza a função é 157 (E=32721800).

6.2.4 Aumentando o número de variáveis

Com o intuito de verificar como o sistema se comporta com o aumento no número de variáveis, o mesmo problema de lançamento vertical foi testado para duas variáveis independentes. Neste experimento, além do tempo t, usou-se também a velocidade inicial V_0 como variável. Os valores para t foram os mesmos do experimento anterior, para valores de V_0 usou-se 20, 25 e $30m/s$. O gráfico com esses valores passa a ser então tridimensional (figura 6.12)

Para esse problema, apenas alguns parâmetros foram alterados em relação aos descritos na tabela 6.5. Os parâmetros modificados são mostrados na tabela 6.6.

Tabela 6.6: Parâmetros alterados para o experimento com a adição da velocidade inicial como variável.

número de execuções seqüenciais	20
número de gerações	5000
tamanho da população	200
número de casos de aptidão	150 pontos

Não se alterou o tamanho do cromossomo, porque a adição da nova variável não torna a expressão mais complexa. Pelo contrário, como não há mais necessidade de geração de uma

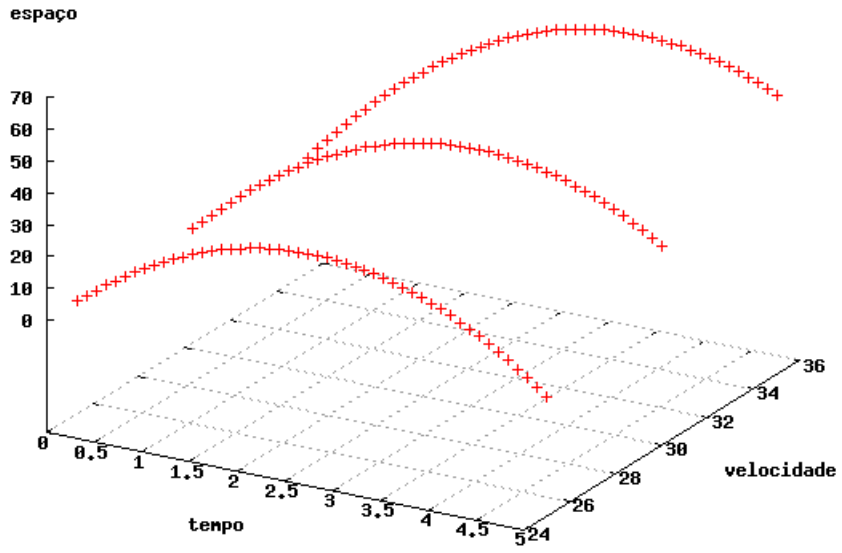


Figura 6.12: Gráfico tridimensional descrevendo três lançamentos verticais no vácuo com velocidades iniciais 20, 25 e 30m/s. As unidades estão em metros por segundo para a velocidade, segundos para o tempo e metros para altura (espaço).

constante para a velocidade inicial, a expressão fica mais simples de ser representada.

A aproximação ocorre bem mais lentamente que o caso anterior, não houve acerto total nas vinte execuções, a melhor solução obteve erros menores do que $0,01$ em apenas 65 pontos.

O sistema parece comportar-se de modo semelhante ao do problema anterior quanto à geração de constantes. O indivíduo de 65 pontos de aproximação é mostrado pela expressão 6.10. A parte em negrito é o "núcleo" da expressão, o sistema apenas errou por pouco o valor da constante da gravidade que multiplica o t (representando o tempo). O v_0 representa a velocidade inicial.

$$\frac{3t^2}{v_0} - \mathbf{5t^2} + \mathbf{v_0 t} \tag{6.10}$$

6.3 Processamento distribuído e migração

O objetivo do experimento a seguir era verificar o comportamento do sistema distribuído com diferentes freqüências de migração: sem migração (equivalente a múltiplas execuções seqüenciais), com baixa e alta freqüência. Os casos de avaliação utilizados foram os do fenômeno geofísico.

Montaram-se três grupos de quatro subpopulações cada. O primeiro grupo evoluía as subpopulações como ilhas separadas sem migração. O segundo grupo, com migração a cada 50 gerações. O terceiro grupo com migração a cada 200 gerações. Todos os grupos foram postos para evoluir por 2000 gerações. Foram efetuadas 18 execuções para cada grupo. Todos os outros parâmetros para as execuções dos grupos, excetuando o período de isolamento (migração), foram igualados conforme a tabela 6.7. Para o experimento não ser afetado pelas taxas de latência de rede e as diferenças na capacidade de processamento entre as máquinas, as subpopulações e o servidor de migração foram mantidos no mesmo computador.

A população pequena (50 indivíduos) foi escolhida propositadamente por dois motivos: para aumentar os efeitos da entrada de imigrantes nas subpopulações e para homogeneização mais rápida nos períodos de isolamento. Essa configuração foi estabelecida com o objetivo de tornar mais evidentes, e de maneira mais rápida, os efeitos de migração com diferentes períodos de isolamento.

O gráfico da figura 6.13 mostra a média de aptidão do melhor indivíduo para as 18 execuções de cada grupo. A média do melhor indivíduo tende a ser bem melhor nas populações com migração. O aumento do período de isolamento diminui a média em curto prazo, mas tende

Tabela 6.7: Parâmetros usados no experimento com os dados do fenômeno geofísico usando-se subpopulações com migração.

número de execuções seqüenciais	1
número de gerações	2000
tamanho da população	50
tamanho do cromossomo	195
genes por cromossomo	3
taxa de replicação	0,001
taxa de recombinação em 1 ponto	0,3
taxa de recombinação em 2 pontos	0,3
taxa de mutação	0,044
taxa de inserção de seqüência de transposição	0,1
taxa de inserção de seqüência de transposição na raiz	0,1
taxa de transposição de gene	0,1
taxa de recombinação de gene	0,1
taxa de inversão	0,1
faixa de seleção	100
método de seleção	roleta
conjunto de funções	+, *, - e % (divisão protegida)
critério de parada	número máximo de execuções

a evitar estagnação da evolução em longo prazo. Isso fica evidente na comparação entre as curvas para 50 e 200 gerações de isolamento: o grupo com isolamento menor tem uma subida de aptidão média mais rápida, mas tende a estabilizar-se e é ultrapassada por volta da geração 1200 pelo grupo de isolamento maior. No caso do grupo com isolamento total, a baixa aptidão da maioria das subpopulações mantém o valor médio baixo.

O cenário se inverte quando se considera a aptidão máxima encontrada em cada geração. Os valores de aptidão máxima encontrados por cada grupo nas 18 execuções são mostrados na figura 6.14. Nesse caso, quanto maior o isolamento, melhor o resultado em médio prazo. Embora o papel da migração seja reinserir diversidade nas subpopulações, evitando assim convergência prematura, migrações freqüentes têm um efeito de igualar o espaço de busca, enquanto populações isoladas representam um espaço de busca maior. Por outro lado, em longo prazo, a situação melhora gradativamente para o grupo com freqüência de migração média, enquanto o grupo de populações isoladas tende à estagnação.

Uma comparação entre as aptidões mínimas conseguidas entre os grupos mostra novamente melhor desempenho da opção de meio termo em longo prazo (figura 6.15). Alta freqüência de migração tende a nivelar as populações “por baixo”, evitando populações locais pobres. O contrário acontece com populações isoladas: populações com soluções pobres tendem a permanecer pobres até que se ache uma solução local interessante, mas a homogeneização causa o nivelamento da curva em longo prazo. Na migração de meio termo, a curva tem uma ascensão mais rápida e acaba por ultrapassar a curva “50 G” ao final.

É claro que os parâmetros locais, além da freqüência de migração, podem influenciar nos

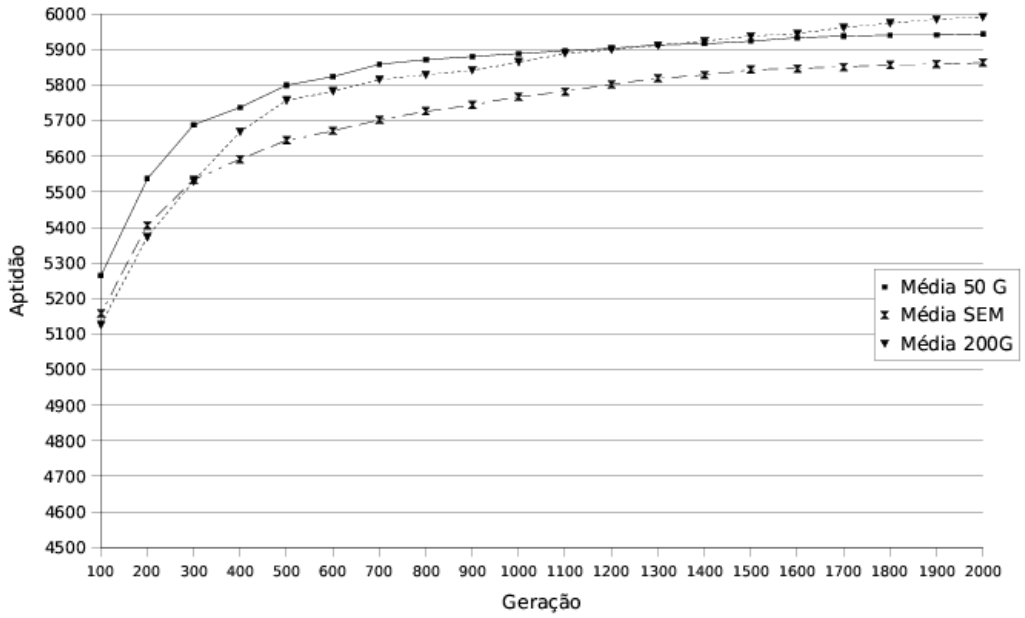


Figura 6.13: Gráfico de comparação de evolução do valor médio de aptidão para o melhor indivíduo em todas as gerações dentre 18 execuções para três grupos de populações: 50G (migração a cada 50 gerações), 200G (migrações a cada 200 gerações) e SEM (sem migração).

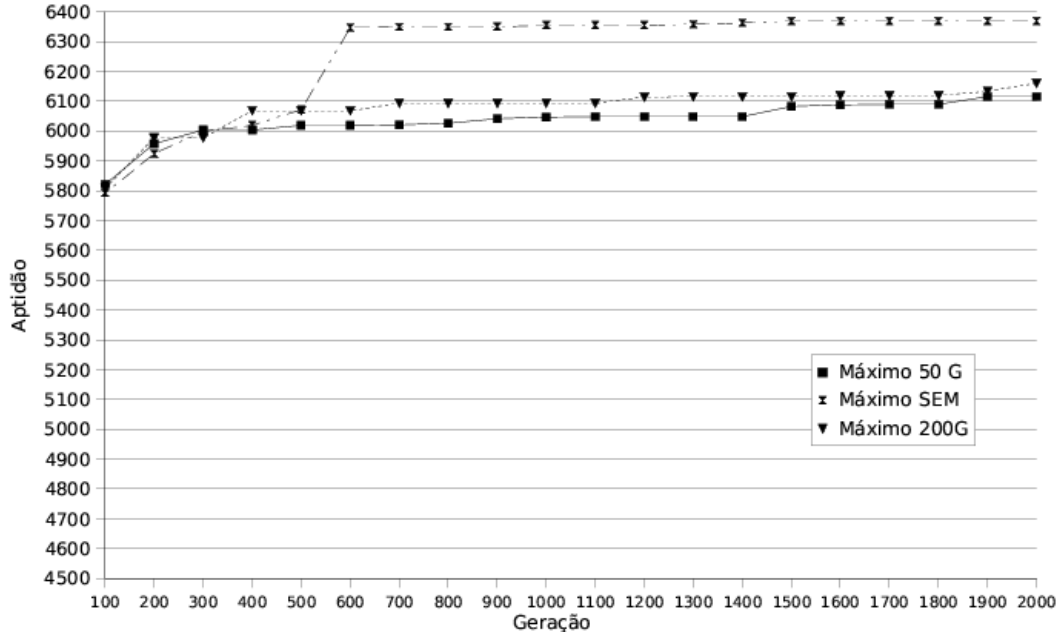


Figura 6.14: Gráfico de comparação de evolução do maior valor de aptidão para o melhor indivíduo em todas as gerações dentre 18 execuções para três grupos de populações: 50G (migração a cada 50 gerações), 200G (migrações a cada 200 gerações) e SEM (sem migração).

resultados de uma estratégia de evolução com migração, testá-los exaustivamente é uma tarefa demorada e caso de um estudo mais completo à parte.

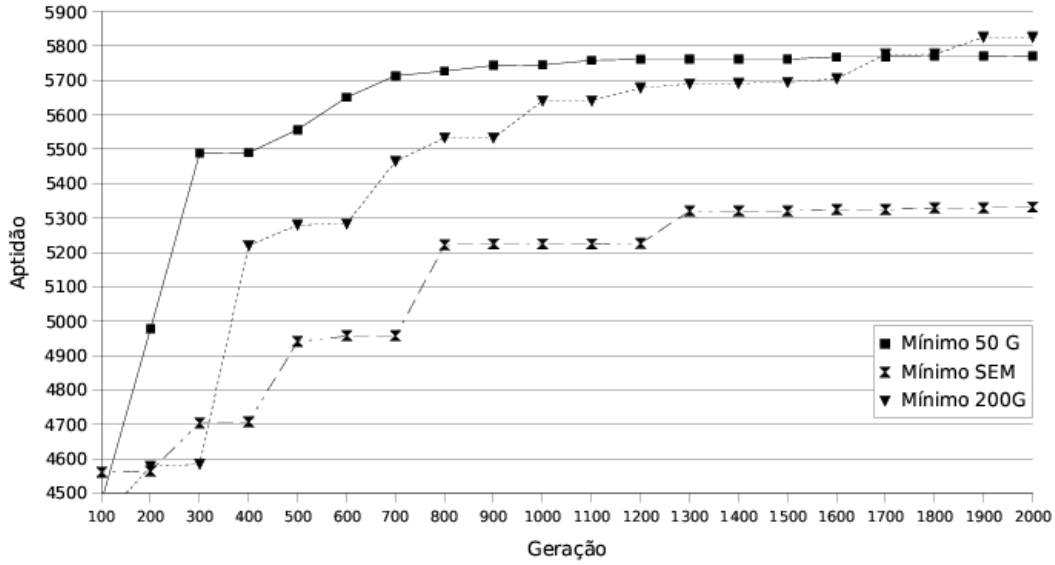


Figura 6.15: Gráfico de comparação de evolução do menor valor de aptidão para o melhor indivíduo em todas as gerações dentre 18 execuções para três grupos de populações: 50G (migração a cada 50 gerações), 200G (migrações a cada 200 gerações) e SEM (sem migração).

6.4 Resultados para o sistema geofísico

Apesar de a programação da expressão gênica se comportar muito bem para problemas simples e mesmo com a maior capacidade de processamento disponível com o sistema distribuído, os resultados obtidos para o sistema geofísico parecem indicar que, pelo menos na configuração implementada do sistema, esta não se comporta bem para problemas deste tipo.

6.4.1 Isolando fluxos

A mesma idéia que foi aplicada para o problema de lançamento vertical usou-se também com o sistema geofísico. Apenas um dos fluxos foi isolado deixando a deriva zonal apenas em função da hora do dia.

A figura 6.16 mostra o gráfico resultante (sistema a aproximar).

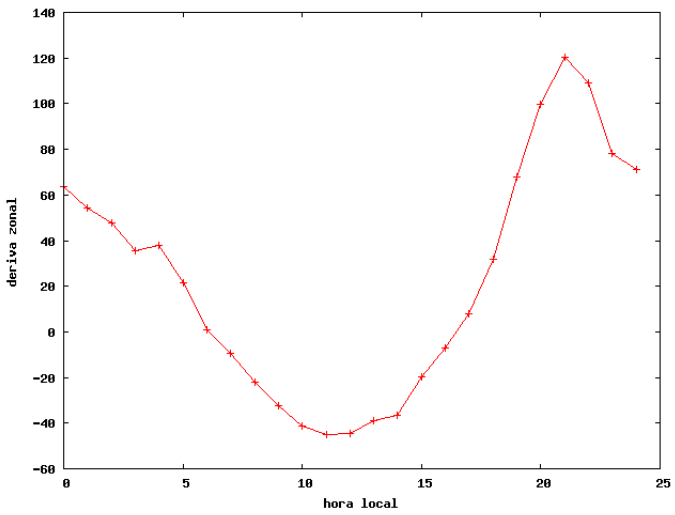


Figura 6.16: Curva formada pelos dados geofísicos usando apenas o fluxo de 90 sfu. As linhas são apenas para melhor compreensão, os pontos a aproximar são representados pelas cruzes.

Os resultados de aproximações para esse fluxo ficaram abaixo das expectativas. Tendo sido tentadas configurações de parâmetros diferentes que envolveram variações de tamanho do cromossomo, número de genes por cromossomo, método de seleção, probabilidades para os operadores, tamanho da população, freqüências de migração, número máximo de gerações e número de execuções, os resultados foram, quase sempre, parecidos.

Nas melhores aproximações conseguidas, o sistema aproxima fracamente o vale central da função por meio de uma parábola e nos picos das extremidades opta por uma linha média (figura 6.17). Nos testes com um fluxo, pontos fora da linha (as irregularidades) aparecem com

freqüência.

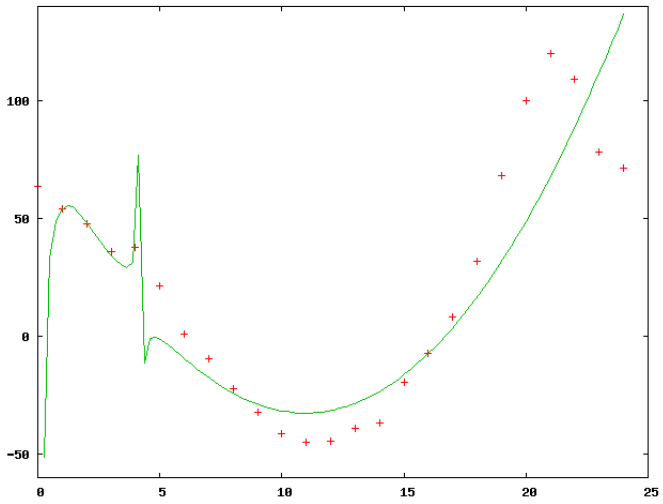


Figura 6.17: Exemplo de resultado obtido para problema simplificado com um fluxo: problemas com a aproximação nas pontas e irregularidades de entre os pontos de avaliação.

A maioria dos testes não envolveu funções trigonométricas por uma exigência estabelecida pelos geofísicos. Apesar disso, efetuaram-se algumas execuções com o uso das funções seno e co-seno apenas a título de teste. Os resultados mostraram certa convergência ao formato da curva esperada, apesar de gerar superfícies bem irregularidades (figura 6.18).

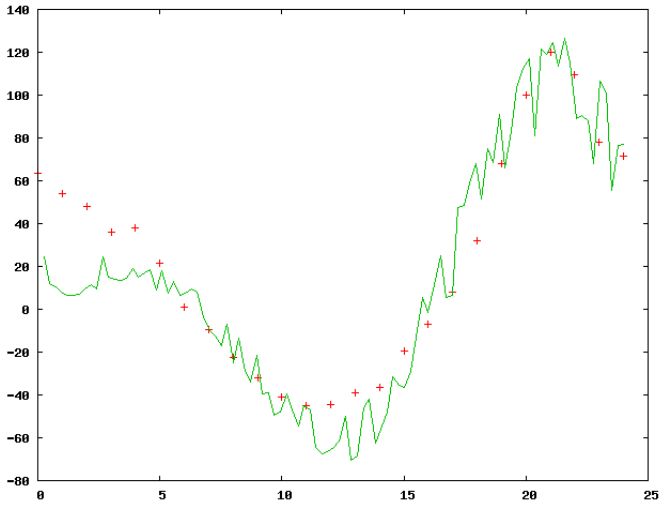


Figura 6.18: Exemplo de resultado obtido para problema simplificado de um fluxo com o uso das funções trigonométricas seno e co-seno.

6.4.2 Modelo com três fluxos

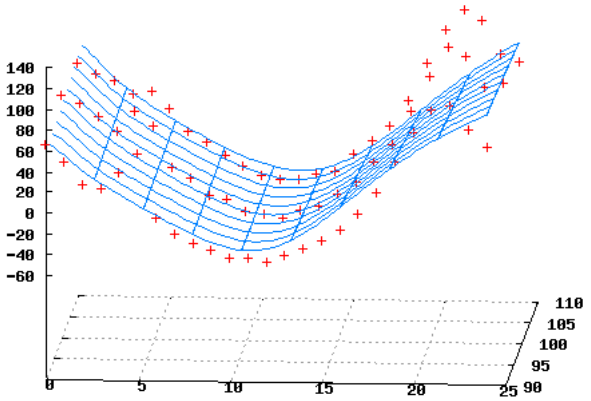
Para o problema completo com três fluxos, as funções encontradas se comportam da mesma forma. O vale central é moldado, mas há dificuldade de aproximar o modelo nas extremidades. Dentre todas as configurações de parâmetros tentadas, o resultado mais comum é o mostrado nas figuras 6.19(a) e 6.19(b). Os parâmetros de configuração para o teste que gerou esse resultado são mostrados na tabela 6.8. O parâmetro "execuções paralelas" indica o número de máquinas usadas no experimento. O tempo de processamento total foi de aproximadamente três horas para a máquina mais rápida e de dezessete horas para a mais lenta.

Tabela 6.8: Parâmetros usados em experimento com os dados geofísicos.

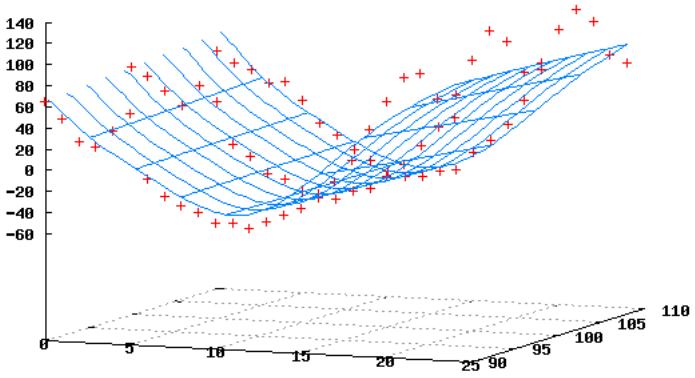
número de execuções paralelas	8
número de execuções seqüenciais	10
número de gerações	5000
tamanho da população	200
tamanho do cromossomo	315
genes por cromossomo	3
taxa de replicação	0,001
taxa de recombinação em 1 ponto	0,3
taxa de recombinação em 2 pontos	0,3
taxa de mutação	0.0063
taxa de inserção de seqüência de transposição	0,1
taxa de inserção de seqüência de transposição na raiz	0,1
taxa de transposição de gene	0,1
taxa de recombinação de gene	0,1
taxa de inversão	0,1
faixa de seleção	1000
método de seleção	roleta
conjunto de funções	+, *, - e % (divisão protegida)
critério de parada	número máximo de execuções

A análise das curvas de aptidão média e aptidão máxima da população para esse mesmo teste mostra que o algoritmo, usado com as taxas de aplicação de operadores genéticos sugeridos pela autora, tem uma grande tendência a valorizar a exploração em detrimento da prospecção. A curva de aptidão média não mostra sinal algum de convergência para o valor máximo (figura 6.20).

Em testes comparativos com a versão de avaliação do programa comercial *Automatic Problem Solver* obtiveram-se resultados semelhantes para as mesmas configurações. O uso de constantes criadas de forma explícita parece melhorar bastante os resultados, de forma a mostrar que, pelo menos para o problema em questão, as afirmações sobre criação implícita de constantes por parte da autora não procedem.



(a) Visão “de frente” da superfície.



(b) Visão de outro ângulo.

Figura 6.19: Exemplo de resultado obtido para problema completo: o modelo não converge corretamente as extremidades da superfície.

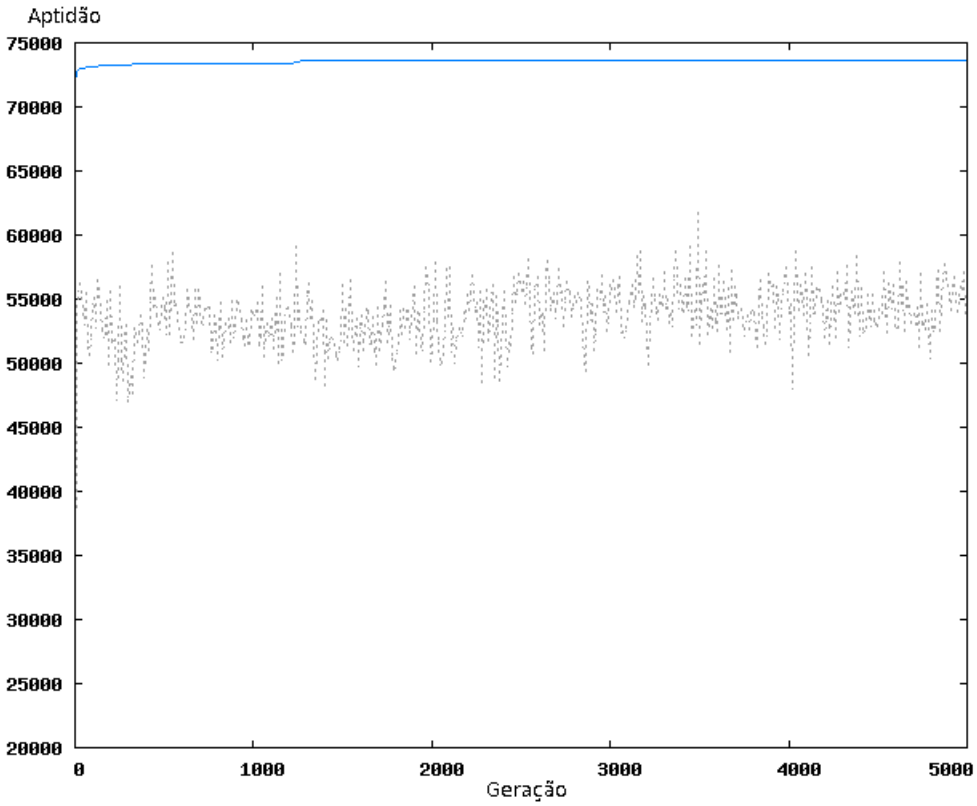


Figura 6.20: Curvas de aptidão média (linha tracejada) e máxima para execução de uma subpopulação no teste envolvendo os dados geofísicos.

7 Discussão e conclusões

A motivação original do grupo de pesquisa em que este trabalho esteve inserido foi a busca de um modelo para o sistema geofísico explicado no capítulo 4. Todavia, testar sistemas de programação genética desenvolvidos para esse fim tornou-se, mais do que um meio para a obtenção de um resultado, um objetivo em si. As dificuldades encontradas foram principalmente com relação recursos computacionais necessários para a obtenção de resultados satisfatórios. Nesse contexto, a principal contribuição deste trabalho foi a criação de um sistema de processamento distribuído para programação genética baseado na variante de programação da expressão gênica que permite contornar de maneira prática essas dificuldades de recursos.

Dado que o objetivo era testar programação genética como ferramenta para a busca de modelos para o sistema geofísico, pode-se afirmar que os resultados foram promissores, pois ela é capaz de encontrar modelos aproximados para um sistema já conhecido como o da cinemática. Por outro lado, os resultados obtidos com o modelo geofísico mostram que o algoritmo ainda precisa de ajustes para tornar-se viável.

Levando-se em conta a abordagem tomada para a resolução do problema para pode-se dividir esse trabalho em duas fases distintas: testes com a programação genética tradicional e a construção do sistema distribuído de programação da expressão gênica.

7.1 Sobre a primeira fase

Os testes com programação genética tradicional mostram uma boa aproximação inicial para o problema geofísico, mas não conseguem um ajuste mais fino na reta final. Métodos de ajustes de constantes talvez ajudassem nesse problema, mas não foram testados a fundo neste trabalho. Alterações de pequeno vulto no algoritmo não parecem ser suficientes para melhorar o desempenho de maneira satisfatória, pois a escala de capacidade de processamento exigida por problemas “reais” torna insignificante qualquer melhoria simples. De qualquer forma, a comparação com outras variantes na literatura da área mostra que o método tradicional tem ficado para

trás. E os testes comparativos mostram que, em relação à programação da expressão gênica, isso é verdade, pelo menos para os problemas de complexidade pequena e média.

7.2 Sobre a segunda fase

Os *toy problems* mostram aproximações interessantes e com significado matemático semelhante, porém, existe um alto custo de processamento mesmo para problemas triviais. O problema com lançamento vertical no vácuo mostra que o sistema é capaz de encontrar fórmulas com significado para a Física, mas evidenciam novamente a necessidade de uso intensivo de tempo de processamento. Nesse sentido, justifica-se a criação do sistema com processamento distribuído.

Os resultados para o problema geofísico, no entanto, estão aquém dos obtidos com a programação genética tradicional. A questão principal é se isso é apenas uma questão de poder computacional ou se há algum problema com o algoritmo de busca.

É fato que problemas complexos como este em estudo exigem longo tempo de ajuste de parâmetros. Talvez a questão do pouco tempo de testes com o sistema justifique a obtenção de resultados fracos. A adição do processo distribuído com migração aumenta a complexidade de ajuste, já que os resultados na seção 6.3 tornam evidente a participação da migração na pressão seletiva. Ajustar os parâmetros de migração corretamente é uma tarefa à parte dentro do sistema.

No entanto, para o caso da programação da expressão gênica, o problema parece ir mais longe. O fato de resolver problemas de complexidade baixa e média de maneira muito mais rápida do que a programação genética tradicional, mas ser incapaz de convergir para problemas de complexidade alta, parece ser inerente à forma como trabalha. Os resultados parecem confirmar a afirmação de Xin Li [48] de que os operadores dessa variante são excessivamente destrutivos, evitando a construção de funções mais complexas e, assim, ser dependente de métodos de criação explícita e de otimização de constantes. Portanto, o trabalho com otimização de constantes, tanto na modalidade tradicional quanto na variante de expressão gênica, parece ser algo realmente necessário para melhorar a qualidade das aproximações.

Variantes como a programação genética linear confiam em operadores genéticos menos destrutivos e parecem obter melhores resultados. Um artigo recente comparando programação genética linear, programação por multiexpressão, expressão gramatical e programação da expressão gênica [64], mostra um desempenho fraco desta última frente às duas primeiras variantes.

7.3 Outras considerações

Não foram feitos testes com o uso de diferentes funções de aptidão como funções com nivelamento 2.10. As funções utilizadas pela programação de expressão gênica parecem funcionar razoavelmente bem para a maioria dos problemas simples, mas, como há problemas com o aumento de escala do problema, talvez fosse interessante testar outras opções.

A existência de superfícies irregulares poderia ser evitada com ajustes da função de aptidão, incorporando-se nesta a taxa de inclinação de cada ponto. Funções com taxa de inclinação muito alta entre os pontos próximos teriam valores de aptidão menor e superfícies mais suaves seriam privilegiadas na seleção. O principal obstáculo é o custo computacional da operação. Uma opção mais barata seria simplesmente aumentar o número de pontos dos casos de aptidão, mas esse tipo de alteração exige cuidados. Não se pode inventar dados na curva onde não existem e a validade de uma alteração deste tipo teria que ser acertada com os geofísicos. Isto nos leva a um ponto crucial no presente trabalho: soluções ainda não se encontram por si mesmas.

A presença de um especialista da área é necessária para sugerir possíveis funções construtoras, descobrir relações entre as componentes da solução encontrada e apontar soluções que façam sentido para o sistema estudado. Em caso de uma continuação do estudo com o sistema geofísico em questão, quando os problemas de ordem computacional estiverem solucionados, a proximidade de um especialista em geofísica espacial, e em particular no fenômeno específico estudado, é indispensável.

Ainda com relação a especialistas de outras áreas, há ainda um problema com a forma pela qual são tratados os parâmetros da maioria dos sistemas de programação genética: o ajuste é muito dependente de particularidades da própria ferramenta e é necessário ao leigo entender bastante a fundo seus conceitos. O excessivo trabalho de ajuste de parâmetros parece incompatível com a proposta original da ferramenta que é a de programação automática.

7.4 Trabalhos futuros

Testes com a representação dos dados como uma série temporal mostraram resultados bastante animadores, mas a validade da representação usada e a relevância do uso de séries temporais para a pesquisa precisam ser mais bem discutidas com os especialistas no fenômeno geofísico.

A arquitetura de processamento distribuído mostrada na seção 5.7.4 é genérica e pode ser usada com qualquer outro algoritmo de programação genética. A implementação de outras

variantes, como programação genética linear e programação por multiexpressão, é algo a ser discutido.

O protocolo mostrado na seção 5.7.5 ainda não está totalmente adequado ao processamento para computação em rede (*network computing*). Os testes efetuados no laboratório funcionaram com auxílio de artifícios, como comandos de *script* para iniciar os clientes ao mesmo tempo. Isso não inviabiliza o projeto para uso na rede local, mas para um aumento na escala de processamento, com o uso de máquinas de qualquer lugar na *Internet*, exige-se um protocolo mais robusto, com um melhor controle de falhas.

A proposta de implementação de um controle inteligente para o processamento distribuído com interface visual ainda está em aberto. Em médio prazo, esta é uma melhoria essencial para administração centralizada do processamento. O ajuste automático de parâmetros é outra necessidade que pede um controle mais autônomo do processamento.

Referências Bibliográficas

[1] MOONEY, D.; SWIFT, R. *A Course in Mathematical Modeling*. Washington, DC, USA: Mathematical Association of America, 1999.

[2] PEREIRA, A. E. C.; FEJER, B.; SOARES, A. S. "climatology of f region zonal plasma drifts over jicamarca". *Journal of Geophysical Research*, 2005.

[3] KOZA, J. R. et al. *Genetic Programming 3: Darwinian Invention and Problem Solving*. San Francisco, CA, USA: Morgan Kaufman, 1999. ISBN 1-55860-543-6.

[4] HOLLAND, J. H. *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI, USA: University of Michigan Press, 1975.

[5] TURING, A. M. Computing machinery and intelligence. MIT Press, Cambridge, MA, USA, p. 11–35, 1995.

[6] FRIEDBERG, R. M. A learning machine: I. v. 2, n. 1, p. 2–13, jan. 1958. ISSN 0018-8646.

[7] NEUMANN, J. von. *Theory of Self-Reproducing Automata*. Urbana, Illinois: University of Illinois Press, 1966.

[8] KOZA, J. R. *Genetic programming: a paradigm for genetically breeding populations of computer programs to solve problems*. Stanford, CA, USA, 1990.

[9] CRAMER, N. L. A representation for the adaptive generation of simple sequential programs. In: *Proceedings of the 1st International Conference on Genetic Algorithms*. Mahwah, NJ, USA: Lawrence Erlbaum Associates, Inc., 1985. p. 183–187. ISBN 0-8058-0426-9.

[10] HICKLIN, J. F. *Application of the Genetic Algorithm to Automatic Program Generation*. Dissertação (Mestrado) — Department of Computer Science. Moscow, 1986.

[11] FUJIKO, C.; DICKINSON, J. Using the genetic algorithm to generate lisp source code to solve the prisoner's dilemma. In: *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*. Mahwah, NJ, USA: Lawrence Erlbaum Associates, Inc., 1987. p. 236–240. ISBN 0-8058-0158-8.

[12] DICKMANNS, D.; SCHMIDHUBER, J.; WINKLHOFER, A. *Der genetische Algorithmus: Eine Implementierung in Prolog*. [S.l.], 1987. Disponível em: <http://www.idsia.ch/ juergen/geneticprogramming.html>.

[13] FOGEL, L. J. Autonomous automata. In: *Industrial Research*. [S.l.: s.n.], 1962. v. 4, p. 14–19.

[14] BäCK, T.; HAMMEL, U.; SCHWEFEL, H.-P. *Evolutionary Computation: Comments on the History and Current State*. Disponível em: <citeseer.ist.psu.edu/601414.html>.

[15] BäCK, T.; SCHWEFEL, H.-P. An overview of evolutionary algorithms for parameter optimization. *Evol. Comput.*, MIT Press, Cambridge, MA, USA, v. 1, n. 1, p. 1–23, 1993. ISSN 1063-6560.

[16] DUMITRESCU, D. et al. *Evolutionary Computation*. Boca Raton, Florida, USA: CRC Press, 2000.

[17] SPEARS, W. M. et al. An overview of evolutionary computation. In: BRAZDIL, P. B. (Ed.). *Proceedings of the European Conference on Machine Learning (ECML-93)*. Vienna, Austria: Springer Verlag, 1993. v. 667, p. 442–459. Disponível em: <citeseer.ist.psu.edu/spears93overview.html>.

[18] KOZA, J. R. *Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems)*. [S.l.]: The MIT Press, 1992. Hardcover. ISBN 0262111705.

[19] LUKE, S. Two fast tree-creation algorithms for genetic programming. *IEEE Transactions on Evolutionary Computation*, v. 4, n. 3, p. 274–283, set. 2000. Disponível em: <http://ieeexplore.ieee.org/iel5/4235/18897/00873237.pdf>.

[20] LANGDON, W. B. et al. The evolution of size and shape. MIT Press, Cambridge, MA, USA, p. 163–190, 1999.

[21] LUKE, S.; PANAIT, L. A survey and comparison of tree generation algorithms. In: SPECTOR, L. et al. (Ed.). *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*. San Francisco, California, USA: Morgan Kaufmann, 2001. p. 81–88. ISBN 1-55860-774-9. Disponível em: <http://www.cs.bham.ac.uk/ wbl/biblio/gecco2001/d01.pdf>.

[22] KOZA, J. R. *Genetic Programming II: Automatic Discovery of Reusable Programs*. Cambridge Massachusetts: MIT Press, 1994. ISBN 0-262-11189-6.

[23] LANGDON, W. B. *Data Structures and Genetic Programming*. Tese (Doutorado) — University College, London, 27 set. 1996. Disponível em: <http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/langdon.ps.gz>.

[24] EKART, A. Shorter fitness preserving genetic programs. In: FONLUPT, C. et al. (Ed.). *Artificial Evolution. 4th European Conference, AE'99, Selected Papers*. Dunkerque, France: [s.n.], 2000. (LNCS, v. 1829), p. 73–83. ISBN 3-540-67846-8. Disponível em: <http://www.sztaki.hu/ ekart/ea.ps>.

[25] NORDIN, P.; FRANCONE, F.; BANZHAF, W. Explicitly defined introns and destructive crossover in genetic programming. MIT Press, Cambridge, MA, USA, p. 111–134, 1996.

[26] ANGELINE, P. *Two selfadaptive crossover operations for genetic programming*. 1995. Disponível em: <citeseer.ist.psu.edu/angeline95two.html>.

[27] MüHLENBEIN, H.; SCHLIERKAMP-VOOSEN, D. Predictive models for the breeder genetic algorithm I. Continuous parameter optimization. *Evolutionary Computation*, v. 1, p. 25–50, 1993.

[28] BAKER, J. E. *An analysis of the effects of selection in genetic algorithms*. Tese (Doutorado) — Vanderbilt University, Nashville, 1989.

[29] KRAMER, M. D.; ZHANG, D. Gaps: A genetic programming system. In: *COMPSAC*. [S.l.: s.n.], 2000. p. 614–.

[30] NORDIN, P.; BANZHAF, W.; FRANCONE, F. D. Efficient evolution of machine code for cisc architectures using instruction blocks and homologous crossover. MIT Press, Cambridge, MA, USA, p. 275–299, 1999.

[31] TELLER, A.; VELOSO, M. PADO: A new learning architecture for object recognition. In: IKEUCHI, K.; VELOSO, M. (Ed.). *Symbolic Visual Learning*. Oxford University Press, 1996. p. 81–116. Disponível em: <citeseer.ist.psu.edu/article/teller95pado.html>.

[32] POLI, R. Evolution of graph-like programs with parallel distributed genetic programming. In: BACK, T. (Ed.). *Genetic Algorithms: Proceedings of the Seventh International Conference*. Michigan State University, East Lansing, MI, USA: Morgan Kaufmann, 1997. p. 346–353. ISBN 1-55860-487-1. Disponível em: <citeseer.ist.psu.edu/article/poli97evolution.html>.

[33] MILLER, J. F.; THOMSON, P. Cartesian genetic programming. In: POLI, R. et al. (Ed.). *Genetic Programming, Proceedings of EuroGP'2000*. Edinburgh: Springer-Verlag, 2000. v. 1802, p. 121–132. ISBN 3-540-67339-3. Disponível em: <citeseer.ist.psu.edu/miller00cartesian.html>.

[34] ABRAHAM, A.; NEDJAH, N.; de Macedo Mourelle, L. Evolutionary computation: from genetic algorithms to genetic programming. In: NEDJAH, N.; ABRAHAM, A.; de Macedo Mourelle, L. (Ed.). *Genetic Systems Programming: Theory and Experiences*. Germany: Springer, 2006, (Studies in Computational Intelligence, v. 13). p. 1–20. ISBN 3-540-29849-5. Forthcoming.

[35] FAUSETT, L. (Ed.). *Fundamentals of neural networks: architectures, algorithms, and applications*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1994. ISBN 0-13-334186-0.

[36] LANGLEY, P. et al. Scientific discovery, computational explorations of the creative processes. 1987.

[37] LANGLEY, P.; SIMON, H. A.; BRADSHAW, G. L. *Heuristics for Empirical Discovery*. Pittsburgh, PA, June 1984.

[38] TODOROVSKI, L. *Declarative Bias in Equation Discovery*. Dissertação (Mestrado) — University of Ljubljana, 1998.

[39] TODOROVSKI, L. *Using domain knowledge for automated modeling of dynamic systems with equation discovery*. Dissertação (Mestrado) — University of Ljubljana, 2003.

[40] TOROPOV, V. V.; ALVAREZ, L. F. Application of genetic programming to the choice of a structure of global approximations. In: . University of Wisconsin, Madison, Wisconsin, USA: [s.n.], 1998. p. 387?–390.

[41] TOPCHY, A.; PUNCH, W. F. Faster genetic programming based on local gradient search of numeric leaf values. In: SPECTOR, L. et al. (Ed.). *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*. San Francisco, California, USA: Morgan Kaufmann, 2001. p. 155–162. ISBN 1-55860-774-9. Disponível em: <http://garage.cps.msu.edu/papers/GARAGe01-07-01.pdf>.

[42] FERREIRA, C. Function finding and the creation of numerical constants in gene expression programming. In: *7th Online World Conference on Soft Computing in Industrial Applications*. [S.l.: s.n.], 2002. On line.

[43] POHLHEIM, H. *Genetic and Evolutionary Algorithms: Principles, Methods and Algorithms*. [S.l.], 2005. Acessada em 30-Janeiro-2006. Disponível em: <"http://www.systemtechnik.tu-ilmenau.de/ pohlheim/GA_Toolbox/algindex.html">.

[44] OUSSAIDENE, M. et al. Parallel genetic programming: An application to trading models evolution. In: KOZA, J. R. et al. (Ed.). *Genetic Programming 1996: Proceedings of the First Annual Conference*. Stanford University, CA, USA: MIT Press, 1996. p. 357–380. Disponível em: <http://cuiwww.unige.ch/ chopard/Genetics/GP96.ps>.

[45] FOLINO, G.; PIZZUTI, C.; SPEZZANO, G. A cellular genetic programming approach to classification. In: BANZHAF, W. et al. (Ed.). *Proceedings of the Genetic and Evolutionary Computation Conference*. Orlando, Florida, USA: Morgan Kaufmann, 1999. v. 2, p. 1015–1020. ISBN 1-55860-611-4. Disponível em: <citeseer.ist.psu.edu/folino99cellular.html>.

[46] FERREIRA, C. *Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence*. [s.n.], 2002. ISBN 972-95890-5-4. Disponível em: <http://www.gene-expression-programming.com/gep/Books/index.asp>.

[47] FERREIRA, C. Mutation, transposition, and recombination: An analysis of the evolutionary dynamics. In: ROMAY, M. G.; DURO, R. (Ed.). *4th International Workshop on Frontiers in Evolutionary Algorithms*. North Carolina, USA: [s.n.], 2002. ISBN 0-9707890-1-7. Disponível em: <http://www.gene-expression-programming.com/webpapers/ferreira-FEA02.pdf>.

[48] LI, X. et al. Investigation of constant creation techniques in the context of gene expression programming. In: KEIJZER, M. (Ed.). *Late Breaking Papers at the 2004 Genetic and Evolutionary Computation Conference*. Seattle, Washington, USA: [s.n.], 2004. Disponível em: <http://www.cs.bham.ac.uk/ wbl/biblio/gecco2004/LBP023.pdf>.

[49] COSTA, W. B. E.; GRINGS, A. *Visual Prolog 6.3 for Tyros*. Pietro, 2005. Acessada em 01-Janeiro-2006. Disponível em: <http://www.visual-prolog.com/vip6/Tutorial/tut14/default.htm>.

[50] TONNEAU, H.; SANTOS, M. V. *The Pliant Documentation Initiative home page*. [S.l.]. Acessada em 01-Janeiro-2006. Disponível em: <http://playground.scs.ryerson.ca:8080/docs.html>.

[51] LEROY, X. *The Objective Caml system release 3.08: Documentation and user's manual*. [S.l.]. Acessada em 1-Dezembro-2005. Disponível em: <caml.inria.fr/pub/docs/manual-ocaml/>.

[52] TONNEAU, H.; SANTOS, M. V. *Fórum de discussão Pliant: sobre compilação em tempo de execução*. [S.l.]. Acessada em 01-Janeiro-2006. Disponível em: <http://fullpliant.org/pliant/browse/forum/pliant.question/04D9EP1/>.

[53] GRINGS, A.; SOARES, A. *Natural computing framework*. [S.l.]. Acessada em 01-Janeiro-2006. Disponível em: <http://nc-framework.sourceforge.net/>.

[54] TAHA, W. A gentle introduction to multi-stage programming. In: *Domain-Specific Program Generation*. [S.l.: s.n.], 2003. p. 30–50.

[55] COSTA, M. V. S. E.; GRINGS, A. Visual languages for interactive computing. In: ____. [S.l.]: Idea Group Inc., 2006. cap. Documentation Methods for Visual Languages. Aceito para publicação.

[56] SASTRY, N. *Visualize your data with gnuplot*. [S.l.], 2006. Acessada em 01-Janeiro-2006. Disponível em: <http://www-128.ibm.com/developerworks/library/l-gnuplot/>.

[57] ABREU, J. B. de. *Manual do Maxima*. [S.l.]. Acessada em 01-Janeiro-2006. Disponível em: <http://maxima.sourceforge.net/docs/manual/pt/maxima.html>.

[58] ROBBINS, A.; BEEBE, N. H. F. *Classic Shell Scripting*. [s.n.], 2005. xxii + 534 p. ISBN 0-596-00595-4. Disponível em: <http://www.oreilly.com/catalog/shellsrptg/>.

[59] KOZA, J. *Código fonte do "little Lisp GP"*. 2003. Acessada em 24-Dezembro-2005. Disponível em: <http://www.genetic-programming.org/gplittlelisp.html>.

[60] KVASNICKA, V.; POSPICHAL, J.; PELIKAN, M. Read's linear codes and evolutionary computation over population of rooted trees. In: *Intelligent Technologies '96*. Slovakia: [s.n.], 1996. II, p. 141–154. ISBN 80-88786-51-7.

[61] ANDERSON, D. P.; FEDAK, G. The computational and storage potential of volunteer computing. *Submetido para publicação*, December 2005. Acessada em 30-Janeiro-2006. Disponível em: <boinc.berkeley.edu/boinc_papers/internet/paper.pdf>.

[62] BUTLER, D. Alien search merges with other home projects. *Nature*, December 2005. Acessada em 01-Janeiro-2006. Disponível em: <http://www.bioedonline.org/news/news.cfm?art=2216>.

[63] BUCK, P. *Which applications are suitable for BOINC?* [S.l.], 2005. Acessada em 30-Janeiro-2006. Disponível em: <http://boinc.berkeley.edu/parallelize.php>.

[64] OLTEAN, M.; GROSAN, C. A comparison of several linear genetic programming techniques. *Complex Systems*, v. 14, n. 4, 2004. ISSN 0891-2513.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo