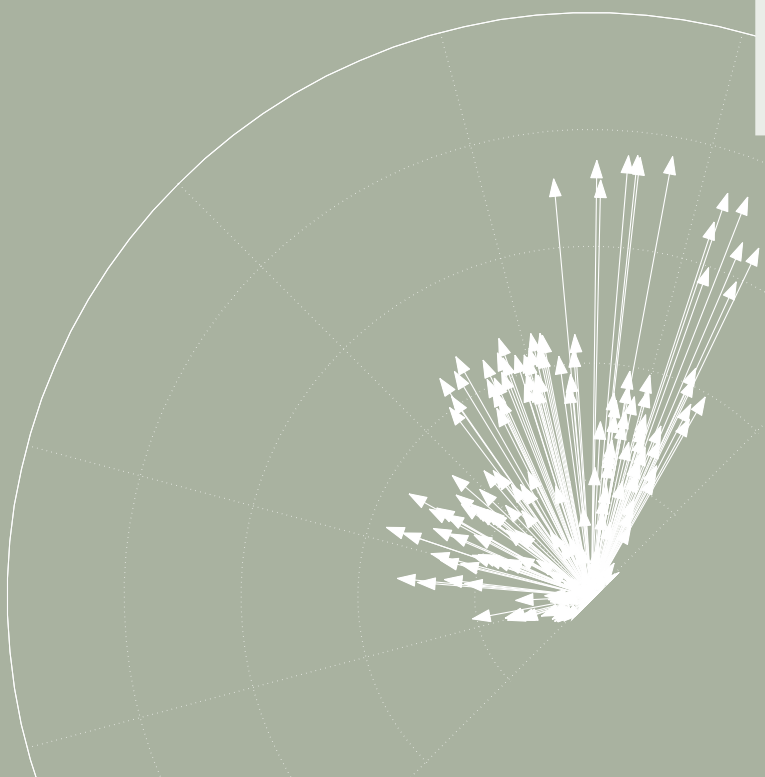
MANUAL DE COMPUTAÇÃO EVOLUTIVA E METAHEURÍSTICA

ANTÓNIO GASPAR-CUNHA
RICARDO TAKAHASHI
CARLOS HENGGELER ANTUNES
COORDENADORES



IMPRENSA DA UNIVERSIDADE DE COIMBRA
COIMBRA UNIVERSITY PRESS

EDITORA ufmg

CAPÍTULO 4

Programação Genética

Douglas A. Augusto *Helio J. C. Barbosa*

Laboratório Nacional de Computação Científica, Petrópolis, RJ

Este capítulo descreve a metaheurística evolucionária denominada *programação genética*, que visa a construção automática de programas de computador por meio de um processo iterativo inspirado na evolução por seleção natural.

A programação genética (PG), cujo desenvolvimento é atribuído a Koza (1992), é uma metaheurística estocástica de otimização global baseada no princípio darwiniano de seleção natural, sendo, pois, uma abordagem da computação evolucionária. A PG destina-se à evolução de programas de computador em linguagens arbitrárias, em outras palavras, a PG otimiza estruturas funcionais capazes de realizar operações, sejam elas lógicas, aritméticas, condicionais e de desvios, que normalmente mapeiam *entradas* em *saídas*.

Nesta concepção, submete-se uma população contendo um determinado número de indivíduos—programas candidatos criados aleatoriamente—ao processo simulado de evolução segundo a seleção natural. Neste processo iterativo figuram a cada geração: a seleção de indivíduos promissores para procriação, a formação de seus descendentes por meio de operações genéticas como cruzamento e mutação, e finalmente a inserção destes novos indivíduos na população, marcando-se o início de uma nova geração. Muito embora não existam garantias de progresso e tampouco de obtenção de soluções ótimas, como característico em qualquer metaheurística, essa dinâmica tende, ao longo das gerações, a produzir soluções candidatas incrementalmente mais adaptadas.

Destacam-se como características típicas da programação genética as seguintes qualidades:

- *Robustez* – o modelo populacional e o processo estocástico são os dois principais fatores por trás da programação genética responsáveis pela solidificação da tolerância a ruídos.
- *Exigência de pouco conhecimento* sobre o domínio – os requisitos acerca do domínio de aplicação podem ser relaxados; basicamente, espera-se apenas uma função capaz de comparar a qualidade relativa dos indivíduos com certa precisão.
- Produz *soluções simbólicas* – usualmente as soluções obtidas pela programação genética são imediatamente legíveis e interpretáveis ("código-fonte disponível"), em decorrência do processo evolutivo atuar diretamente no nível simbólico das estruturas.
- Possui *paralelismo natural* – as demandas computacionais da programação genética podem ser trivialmente particionadas em vários níveis, com emprego ou não de modelos distribuídos bioinspirados, permitindo-se assim redução no tempo de execução e/ou escalabilidade para problemas mais complexos.
- É facilmente *extensível/modificável* – a programação genética é uma metaheurística extremamente versátil, e dessa forma admite, por exemplo, hibridizações (funcionamento junto a outras técnicas), integração de outros modelos evolucionários (p. ex., co-evolução, nichos, múltiplos objetivos) e as mais diversas representações e linguagens para os programas (como representação formal por gramáticas).

O processo evolutivo da programação genética pode ser visualizado pelo fluxograma da Figura 4.1. Como exposto na figura, os componentes e etapas da PG, que são discutidos no decorrer deste capítulo,

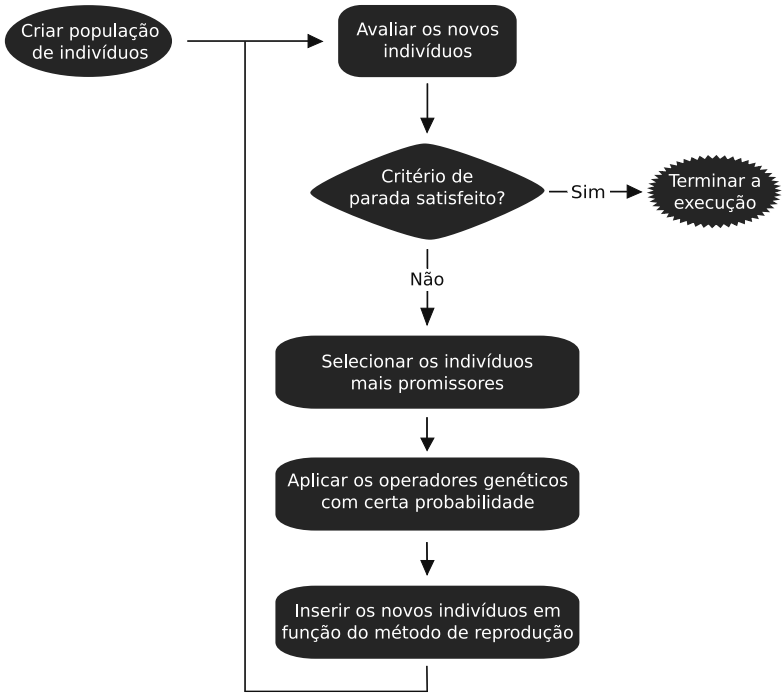


Figura 4.1: Fluxograma do funcionamento da programação genética.

podem ser sintetizados por:

1. *Criação da população* – determina quem e como são formados os primeiros indivíduos candidatos. Para tanto, dois conceitos devem estar definidos:
   (a) representação dos programas – a estrutura funcional na qual o processo evolutivo atuará, isto é, a entidade que será submetida às operações genéticas e avaliações.
   (b) conjuntos primitivos de funções e terminais – estabelece as instruções básicas ("palavras-chave") da linguagem dos programas que, quando devidamente combinadas, expressam a solução esperada.
2. *Avaliação dos indivíduos* – função que mede quão bem um determinado indivíduo executa a tarefa do problema, em outras palavras, é a função-objetivo a ser maximizada.
3. *Critério de parada* – determina quando o processo deve ser interrompido; normalmente por obtenção de uma solução satisfatória ou por limites de tempo de execução.
4. *Esquemas de seleção* – definem como os indivíduos mais bem adaptados são favorecidos na disputa pela procriação.
5. *Operadores genéticos* – são as técnicas de criação de novos indivíduos com base no material genético de seus genitores. Tipicamente subdividem-se em:
   (a) cruzamento – recombinação do material genético de dois indivíduos.
   (b) mutação – deriva um indivíduo como uma variação de um outro.
6. *Métodos de reprodução* – estipulam como os descendentes serão inseridos na população.

Aplicações

Teoricamente, a programação genética pode ser aplicada em qualquer problema que atenda duas propriedades fundamentais:

1. *que a solução possa ser expressa como um programa de computador em uma dada linguagem.*
2. *que seja possível definir uma medida capaz de, dadas duas soluções candidatas, dizer com razoável precisão quais delas é a melhor* ("função de avaliação").

Satisfeitas estas propriedades é possível, em tese, aplicar a programação genética. Na prática, porém, em problemas excessivamente complexos—como os que envolvem avaliações/simulações computacionalmente custosas, ou aqueles cuja estrutura da solução esperada é demasiadamente grande e intricada—a PG esbarra nos limites computacionais no que tange tempo e espaço de processamento, inviabilizando portanto o seu uso.[1] Nestes casos, contudo, uma possível alternativa é o uso da PG em ambientes computacionais de alto desempenho, servindo-se de implementações massivamente paralelas/distribuídas.

A despeito da restrição em certas classes de problemas, a programação genética tem sido usada em áreas diversas e expressivas. A relação apresentada a seguir lista, em caráter ilustrativo, uma pequena parcela dos domínios e suas aplicações na qual a PG tem sido bem sucedida.

Reconhecimento de padrões Neste domínio procura-se a evolução de programas capazes de identificar padrões normalmente implícitos em massas de dados. Exemplos constituem o diagnóstico automatizado de doenças mediante informações de exames do paciente, análise de crédito e cálculo de risco baseando-se em informações históricas e vigentes acerca do tomador de crédito, ou reconhecimento óptico de caracteres digitais/manuscritos.

[1] Naturalmente, à medida que a tecnologia avança e computadores mais rápidos são construídos, a programação genética amplia o seu leque de atuação.

Regressão simbólica Esta área trata da descoberta de expressões matemáticas (expressões simbólicas). Existem aplicações de ajuste de curvas, indução de sequências, derivação e integração simbólica, descoberta de identidades matemáticas, previsão de séries temporais, entre outras.

Robótica Busca-se desenvolver programas que controlem precisamente ações de robôs em diversas tarefas, tais como as de automação industrial.

Estratégias de jogos O objetivo é evoluir estratégias que, dado o estado atual do jogo—e possivelmente o histórico de jogadas—, decidam otimamente qual deve ser a próxima jogada.

Processamento de imagens Uma das aplicações envolve a compactação de imagens, isto é, a descoberta de programas compactos que quando executados recriem a imagem original.

Arte A evolução de programas que codificam (artificialmente) pinturas de quadros, por exemplo, é uma das possíveis aplicações da PG nesta área.

1. Descrição da Programação Genética

Representação dos Programas

Existe uma grande variedade de estruturas capazes de representar programas de computador no âmbito da programação genética.[2] Há no entanto três principais grupos conceituais de representação: *linear* (Oltean et al., 2009), por *árvore* (Koza, 1992) ou por *grafos* (Poli, 1999; Teller, 1996; Miller e Smith, 2006).

Dentre os grupos, a representação por árvore é a mais popular, status este possivelmente decorrente da sua tradição de uso na programação genética, aliada ao poder/espontaneidade de expressão de programas e relativa simplicidade. Todavia, esta classificação é *conceitual*, e difere da *representação computacional*; por exemplo, muito embora a árvore seja uma estrutura naturalmente hierárquica, ela pode ser implementada e operada linearmente sem que haja violação de suas propriedades (Pelikan et al., 1997; Augusto e Barbosa, 2000). Ainda, um terceiro nível de classificação pode ser introduzido: a *representação semântica*. Neste, a representação conceitual é complementada com regras/restrições que governam como a estrutura em questão pode ser operada; pode-se citar como membros desta classificação, por exemplo, a representação por gramática sobre a estrutura árvore (Whigham, 1995) e sobre a estrutura linear (O'Neill e Ryan, 2001), e restrições de tipo para a estrutura árvore (Montana, 1994).

A despeito do extenso leque de opções de representação, considera-se neste capítulo a representação pela estrutura de dados *árvore*, cujo emprego é bastante comum na PG. Nesta representação, o cromossoma de um indivíduo apresenta um aspecto gráfico como exemplificado na Figura 4.2. O indivíduo **A** descreve a expressão

$$\text{se } A \wedge B \text{ então } C, \text{ senão } (3.14 \times X),$$

enquanto o indivíduo **B** descreve a função matemática

$$\sin\left(\sqrt{0.987} - \cos(X \times Y)\right).$$

[2] Essas estruturas são muitas vezes referidas por *cromossoma* ou *genoma* do indivíduo.

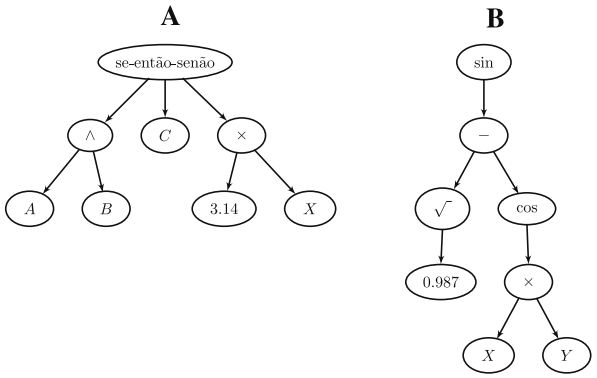


Figura 4.2: Indivíduos representados pela estrutura árvore.

Conjuntos Primitivos de Funções e Terminais

Essenciais ao funcionamento da programação genética, como originalmente concebida, são os denominados conjuntos de *funções* ($\mathbb{F}$) e *terminais* ($\mathbb{T}$), que são relativos ao domínio da aplicação. São eles que definem as ferramentas (as primitivas) que estarão à disposição do processo evolucionário para a construção de estruturas mais complexas, isto é, definem a linguagem—e o feitio do espaço de busca – onde a evolução poderá explorar.

O conjunto $\mathbb{F}$ é responsável pelo fornecimento de funções que requerem argumentos, ou seja, os *operadores*. Exemplos incluem operadores matemáticos ($+$, $-$, $\times$, $\div$, sin, cos, $\sqrt{}$, log, x^y), lógicos ($\vee$, $\wedge$, $\neg$), relacionais ($<$, $>$, $=$), condicionais (se x então y senão z) e laços iterativos (enquanto x faça y).

Por sua vez, o conjunto $\mathbb{T}$ provê os *operandos*, isto é, variáveis/atributos (x, y, *idade*, *salário*), constantes (π, e) e funções que não requerem argumentos (*random()*, *tempo_decorrido()*).

Os conjuntos de funções e terminais podem ser livre e recursivamente combinados; a única restrição é que seja obedecida a aridade[3] de cada primitiva. Portanto, todo fragmento ou estrutura de um programa na população, em qualquer estágio do processo evolutivo, nada mais é do que uma combinação particular entre elementos de $\mathbb{F}$ e $\mathbb{T}$. No entanto, para que estes conjuntos possam ser computacionalmente manipulados e permitam a expressão da solução procurada, duas propriedades devem ser satisfeitas:

Suficiência Do fato de que qualquer estrutura da população é um arranjo particular de $\mathbb{F}$ e $\mathbb{T}$, decorre que é condição *necessária* para a obtenção da solução esperada que esta possa ser expressa como uma combinação das primitivas definidas para o problema. Esta óbvia constatação é conhecida como *propriedade de suficiência*, e diz que os elementos dos conjuntos $\mathbb{F}$ e $\mathbb{T}$ devem ser suficientes para, em ao menos uma combinação específica, descrever a solução esperada.

Na prática, entretanto, a propriedade de suficiência não soa tão óbvia, devido ao fato de que em muitos problemas—especialmente os reais e de grande porte—não se conhece o aspecto da solução, e dessa forma pouco se pode cogitar a respeito da formação dos conjuntos de funções e terminais. Neste cenário, é tentador adicionar aos conjuntos tantas primitivas quanto se possa; contudo, tal procedimento aumenta enormemente o espaço de busca, e pode, portanto, seriamente degradar o progresso da evolução.

Consistência Uma segunda propriedade que concerne aos conjuntos de primitivas é a denominada *consistência*. Consistência é a condição de integridade, e requer que cada operador do conjunto

[3] *Aridade* é o número requerido de argumentos. Os terminais em $\mathbb{T}$ têm sempre aridade *zero*.

de funções seja suficientemente versátil e robusto para aceitar como argumento qualquer terminal em $\mathbb{T}$ ou valores de retorno de qualquer função em $\mathbb{F}$. Em outras palavras, os conjuntos de terminais e funções têm que ser tais que, quaisquer combinações entre $\mathbb{F}$ e $\mathbb{T}$, ou $\mathbb{F}$ e ele próprio, desde que respeitadas as aridades, sejam válidas.[4]

A consistência é uma propriedade demasiadamente relaxada no que se refere às relações e restrições entre as primitivas, isto é, no que tange a definição da linguagem dos programas. Ao mesmo tempo em que os conceitos dos conjuntos de funções e terminais da programação genética canônica proveem simplicidade e uniformidade, a ausência de restrições pode causar: (i) crescimento não justificado do espaço de busca – regiões conceitualmente sem sentido passam a fazer parte dele; e (ii) dificuldade de satisfazer a condição de consistência quando estão envolvidos diferentes tipos e estrutura de dados, tornando-a inviável ou, na melhor das hipóteses, forçando uma adaptação crassa.[5] Felizmente, na prática estes cenários não ocorrem com frequência. No entanto, para domínios de aplicação onde as deficiências e limitações da noção dos conjuntos de terminais/funções mostram-se preocupantes ou proibitivas, é possível adotar representações mais sofisticadas, como, entre outras, a representação formal por gramática (Whigham, 1995).

Criação da População

A criação da população visa formar uma amostragem representativa do espaço de programas, que seja preferencialmente bem distribuída e sem tendenciosidade a favor de configurações particulares. Entretanto, dado que o espaço de possíveis programas é geralmente infinito—quando não se impõe limites acerca do tamanho das soluções—, torna-se impossível amostragens estritamente uniformes; o que se pode fazer, todavia, é tentar evitar grandes distorções (Poli et al., 2008a).

Na prática, a criação dos indivíduos iniciais, e portanto da população, é realizada de maneira essencialmente *aleatória*, isto é, as instruções básicas dos conjuntos $\mathbb{F}$ e $\mathbb{T}$ são recursivamente combinadas ao acaso, em número e complexidade variáveis, na estrutura do indivíduo.[6] Embora o processo de criação seja não-determinístico, é preciso que se respeite a aridade de cada operador; em outras palavras, se uma determinada função possui n argumentos, então, considerando-se a estrutura árvore, o nó representando este operador terá exatamente n nós filhos, cada qual constituindo um argumento.

Na tentativa de se criar uma boa amostragem inicial, métodos que delineiam a forma como as árvores iniciais são construídas foram propostos. Alguns desses métodos são descritos a seguir.

Método de Criação Completa (*Full Creation Method*)

Na PG é interessante que se imponha limites nos tamanhos dos cromossomas (árvores), evitando assim, que as soluções candidatas cresçam indefinidamente, provocando esgotamento de memória e degradação no desempenho. Um destes limites está relacionado ao momento da criação dos indivíduos, e é geralmente especificado por profundidade máxima.[7]

Neste método de criação, exemplificado pela Figura 4.3, todo comprimento entre o nó raiz e qualquer nó terminal é sempre igual à profundidade máxima especificada. Isto significa que, até se alcançar a profundidade máxima, todos os nós terão um ou mais filhos (funções com mais de

[4] Uma das situações mais comuns envolve o operador de divisão, que normalmente precisa ser definido de modo a tratar a divisão por zero, p. ex. retornar a constante 1 neste caso.

[5] Por exemplo, não faz sentido conceitual a multiplicação de valores booleanos, no entanto, seria necessário garantir a factibilidade desta operação para que o processo evolutivo fosse viável.

[6] Em certas aplicações, no entanto, conhecendo-se alguns padrões úteis e promissores, é possível introduzi-los diretamente na população inicial, mas é preciso cautela para que estas sementes não causem convergência prematura.

[7] Profundidade é o maior comprimento (ramo) entre o nó raiz e os demais nós de uma árvore. Quando se tem noção da dimensão da solução do problema, além do limite máximo de profundidade, é usualmente também especificado um limite mínimo.

um argumento) e, ao se atingir a profundidade determinada, os nós não possuirão filhos (funções sem argumentos, variáveis ou constantes).

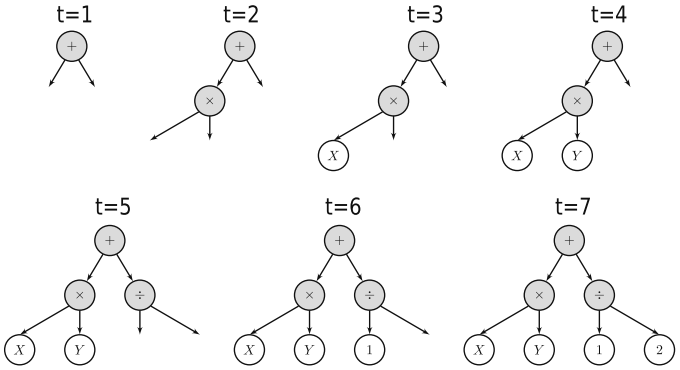


Figura 4.3: Exemplificação passo-a-passo do método de criação *completa*.

Método de Criação Livre (*Grow Creation Method*)

Diferentemente do método completo, este permite que nós com qualquer número de filhos (argumentos) sejam adicionados no momento de criação, assim, não é necessário que a árvore seja completa. Isto é, se ocorrer em algum ramo um nó sem filhos (terminal), este ramo será encerrado. A Figura 4.4 ilustra esse processo.

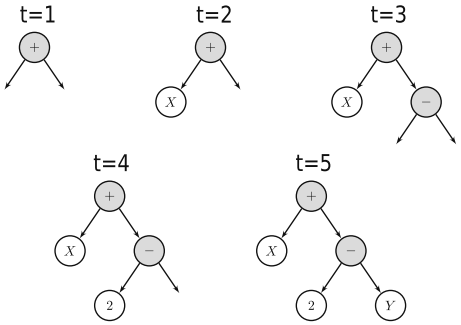


Figura 4.4: Exemplificação passo-a-passo do método de criação *livre*.

Efeito Escada (*Ramping*)

No momento da criação, a população é dividida em segmentos. A cada segmento é atribuído uma profundidade máxima, de modo que esta profundidade máxima varie entre a profundidade mínima real e a máxima real. Esta atribuição geralmente é feita em ordem crescente. Por exemplo, considere uma população com 50 indivíduos, e suponha que a profundidade mínima seja 2 e a máxima 6. Então, a população seria dividida em cinco segmentos, cada qual com dez indivíduos. Dessa forma, o primeiro segmento receberia uma profundidade máxima de 2, o segundo 3, e assim, até que o último receba a profundidade 6.

Método Misto de Criação (*Ramped Half-and-Half*)

Segundo Koza (1992), este é o método mais recomendado. É capaz de gerar uma ampla diversidade de indivíduos, produzindo várias formas e tamanhos distintos de árvores. A ideia consiste

em fazer com que cada metade de um segmento produzido por "*ramping*" seja criada ou com o método completo (*full*) ou com o método livre (*grow*). Isto é, metade de um segmento será criada pelo método completo e a outra metade com o método livre.

Avaliação dos indivíduos

A avaliação dos indivíduos, que depende fundamentalmente do domínio da aplicação, mede o êxito com que os programas realizam a tarefa alvo, isto é, define a *aptidão*. Tipicamente, cada programa da população é executado sobre conjuntos de argumentos, e então os valores de retorno são comparados com os valores esperados, que são conhecidos; quanto mais próximos os valores de retorno estão dos valores esperados, mais bem adaptado é o indivíduo em questão.

Em problemas de regressão, por exemplo, interessa minimizar a discrepância entre os valores esperados para os pontos fornecidos e os valores obtidos pela função sendo avaliada; portanto, uma medida de avaliação poderia ser o *inverso* da soma do quadrado dos erros:

$$f_{\text{aptidão}} = \left[\sum_{i=1}^{k}[f(p_i) - Y_i]^2\right]^{-1},$$

ou então o *negativo* da soma do quadrado dos erros:

$$f_{\text{aptidão}} = -\sum_{i=1}^{k}[f(p_i) - Y_i]^2,$$

onde k é o número de pontos, $p_i \in \mathbb{R}^n$ é o i-ésimo ponto, f é a função—codificada por um indivíduo qualquer—sendo avaliada ($f : \mathbb{R}^n \mapsto \mathbb{R}$), e $Y_i \in \mathbb{R}$ é o valor esperado para o ponto de índice i.

Já em problemas de classificação de dados[8] o objetivo primário é medir a acurácia de predição sobre um determinado conjunto de amostras de treinamento; dessa forma, a aptidão de uma árvore classificadora C de um indivíduo ($C : \mathbb{R}^n \mapsto \mathbb{N}$), normalizada para o intervalo $[0, 1]$, poderia ser dada pela *taxa de acertos*:

$$f_{\text{aptidão}} = \frac{1}{k}\sum_{i=1}^{k}\mathbb{I}[C(a_i) = Y_i],$$

onde k é o número total de amostras de treinamento, $a_i \in \mathbb{R}^n$ e $Y_i \in \mathbb{N}$, respectivamente, a i-ésima amostra e sua classe esperada, e $\mathbb{I}[\pi]$ retorna 1 se a condição π é satisfeita ou 0 caso contrário.

No entanto, nesses casos geralmente é útil ponderar a aptidão do indivíduo em função de sua complexidade/tamanho, visando assim a obtenção de soluções mais compactas, legíveis e potencialmente com maior poder de generalização. Uma medida comum é a introdução de um coeficiente de penalização por complexidade, que se propõe a reduzir a aptidão de um indivíduo proporcionalmente ao tamanho de sua estrutura.[9] A aptidão de um indivíduo pode ser, portanto, reescrita como:

$$f'_{\text{aptidão}} = f_{\text{aptidão}} - c \times \ell,$$

onde c é o coeficiente de penalização, que usualmente é uma constante, e ℓ é uma medida de complexidade, onde normalmente emprega-se o tamanho da estrutura do indivíduo como, por exemplo, o número de nós da árvore que representa o programa.

[8] A classificação de dados é a tarefa que, fornecido um conjunto de dados como treinamento, visa construir uma entidade com a propriedade de mapear atributos de entrada em um valor discreto que representa a classe/categoria de uma dada amostra. Esta entidade é denominada *classificador* ou *preditor*.

[9] Soluções mais sofisticadas para o controle de complexidade são discutidas em (Poli et al., 2008a).

Critério de parada

Idealmente, uma execução da programação genética deveria terminar quando, e somente quando, uma solução ótima fosse encontrada segundo a função de aptidão definida. Infelizmente, dependendo da complexidade do problema e dos recursos computacionais disponíveis, isto pode não ser viável. Nesses casos, o relaxamento desta condição no intuito de obtenção de soluções "suficientemente boas" pode ser admissível—ou ser de fato a única alternativa. A implementação deste conceito é feita pela adição de um ou mais critérios de parada, comumente:

- Aptidão dentro da faixa aceitável – o processo é interrompido quando uma solução aproximada é obtida.
- Número máximo de gerações ou avaliações – a execução é interrompida ao se atingir um número pré-estabelecido do gerações ou avaliações.
- Tempo limite de execução – análogo ao item anterior, mas é guiado pelo tempo despendido.
- Estagnação do processo – a execução termina quando é detectada estagnação do processo evolutivo, isto é, quando por um certo período nenhuma solução aprimorada é obtida.

Esquemas de seleção

A seleção é o instrumento pelo qual os algoritmos evolucionários conduzem a busca para as regiões mais promissoras do espaço e, que, efetivamente, permite o processo de otimização. Ela simula o efeito da *seleção natural*, sendo responsável por escolher os indivíduos da população que deixarão descendentes para a geração seguinte, ou seja, aqueles que participarão das operações genéticas. Portanto, é natural esperar que os esquemas de seleção favoreçam aqueles indivíduos mais bem adaptados, pois a transmissão de bom material genético potencializa o desenvolvimento de soluções aperfeiçoadas.

A diferença entre a probabilidade de seleção dos melhores indivíduos e a probabilidade de seleção dos piores denomina-se *pressão de seleção*. Quando a pressão de seleção é baixa, indivíduos de menor qualidade terão mais chance de disseminar seus genes; por outro lado, quando a pressão é alta, somente aqueles indivíduos muito bem avaliados, em média, têm oportunidade de ser selecionados. Embora possa parecer que, a princípio, a alta pressão de seleção é sempre vantajosa, na realidade ela pode levar o processo evolucionário rapidamente à indesejável convergência prematura. Isto ocorre, notoriamente, devido à degradação da diversidade na população, já que apenas um pequeno seleto grupo de indivíduos domina o processo.

Embora existam distintas maneiras de se implementar a seleção (Goldberg e Deb, 1991), cada qual diferindo particularmente no que tange a dinâmica da pressão de seleção e custo computacional, uma variação bastante simples, eficiente e conveniente, nomeada *torneio* (Brindle, 1981), é comumente empregada na programação genética. Neste esquema, k indivíduos são selecionados *aleatoriamente* da população, suas k aptidões são comparadas e o indivíduo detentor da melhor aptidão é selecionado. Nota-se que o valor de k, de fato, define a pressão de seleção, isto é, quanto maior o número de competidores mais rigorosa a seleção se torna.

Operadores genéticos

A função dos operadores genéticos é construir soluções derivadas a partir de outro(s) indivíduo(s). É através desse processo de experimentação de recombinações e/ou variações sobre padrões promissores que o espaço de busca é explorado.

Os dois principais operadores genéticos, descritos a seguir, são o *cruzamento*, que atua sobre dois indivíduos, e *mutação*, que atua sobre um único indivíduo.

Cruzamento

O operador de cruzamento propõe-se a recombinar partes de estruturas previamente selecionadas pelo processo de seleção, na esperança de se derivar estruturas ainda melhores. Devido ao status de operador primário na programação genética, o cruzamento é tipicamente aplicado com alta probabilidade.

Na representação por árvore, o operador de cruzamento comuta sub-árvores, cujos nós raiz são arbitrariamente escolhidos, de dois indivíduos, gerando dessa maneira dois novos descendentes. A Figura 4.5 exemplifica a aplicação do operador de cruzamento sobre os pais **A** e **B**, produzindo os filhos **A'** e **B'**.

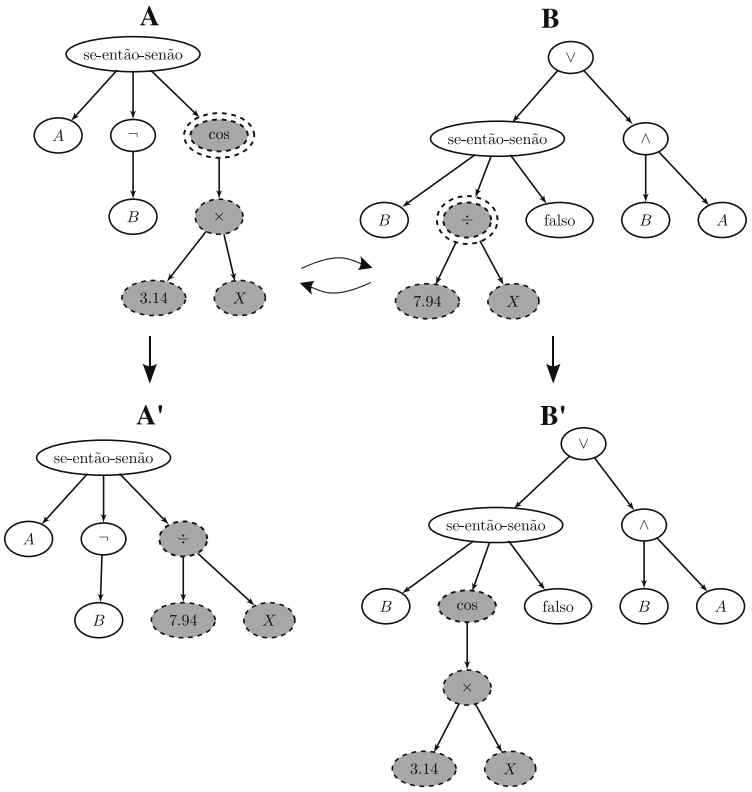


Figura 4.5: Cruzamento padrão.

Mutação

A mutação deriva um novo descendente mediante modificações na estrutura de um indivíduo selecionado. Estas modificações são normalmente de pequena escala ("perturbações"), e visam tanto a exploração de regiões vizinhas àquele indivíduo quanto a (re)introdução de material genético a fim de preservar a diversidade da população. As modificações sobre as estruturas são realizadas, tipicamente, de maneira absolutamente aleatória.[10]

A Figura 4.6 ilustra uma aplicação hipotética do operador padrão de mutação sob a representação por árvore. Primeiramente, uma sub-árvore do indivíduo **A** é escolhida ao acaso, então removida,

[10] Mas existem operadores especializados de mutação que agem deterministicamente; por exemplo, operadores de eliminação de redundância ou de simplificação de expressões em árvores.

e finalmente uma nova sub-árvore, criada aleatoriamente, é introduzida no local, gerando assim o descendente **A'**.

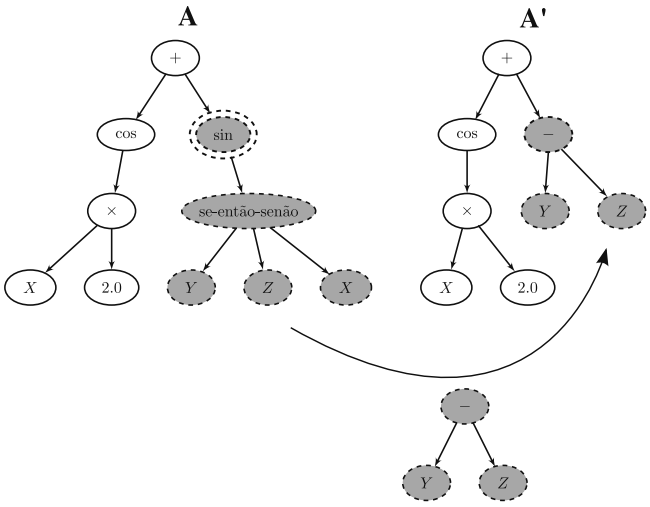


Figura 4.6: Mutação padrão.

Visando a promoção da diversidade da população e a exploração do espaço de busca por caminhos diferentes – às vezes inusitados –, variações do operador padrão de mutação foram desenvolvidas. Dentre estas, podem ser citadas a mutação tipo *nó*, *encolhimento* e *permutação*.

A mutação nó, como o nome indica, provoca uma alteração sobre algum nó da árvore. Assim, após selecionado um elemento, arbitrariamente, este é substituído por algum outro, de tal forma que o novo nó possua o mesmo número de argumentos que o original. A Figura 4.7 ilustra graficamente este processo.

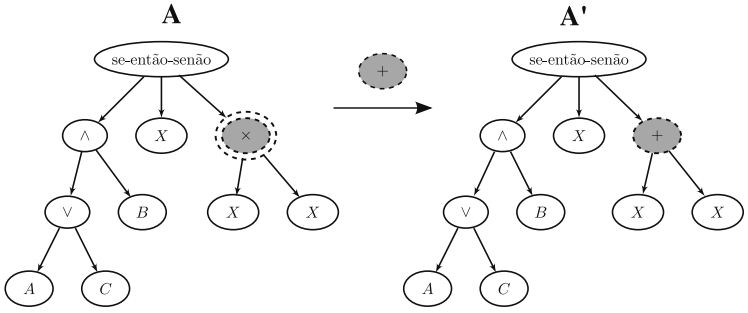


Figura 4.7: O processo de mutação tipo *nó*.

Por sua vez, a mutação tipo encolhimento objetiva reduzir o tamanho do cromossoma (árvore), inspirada no fato de que soluções menores (menos complexas) tendem a ser mais compreensíveis, e portanto desejadas. Seu procedimento consiste em, após escolhido um elemento aleatoriamente, selecionar um filho qualquer e, então, substituí-lo na posição do elemento escolhido (nó pai), excluindo-se todos os outros filhos e suas ramificações. Isto "compacta" a sub-árvore em questão, e portanto reduz o cromossoma como um todo. A Figura 4.8 ilustra visualmente a aplicação da mutação encolhimento.

Já a mutação por permutação provoca uma alteração na ordem (permuta) dos argumentos de uma

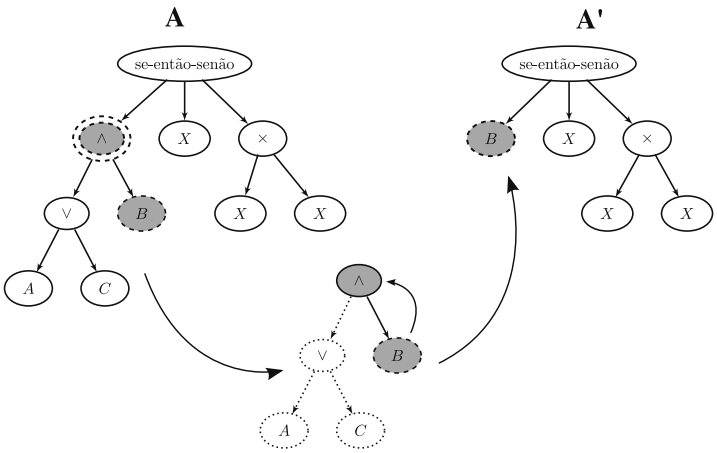


Figura 4.8: O processo de mutação tipo *encolhimento*.

função qualquer. Naturalmente, este operador só se aplica às funções com no mínimo dois argumentos. Um exemplo pode ser visto na Figura 4.9.

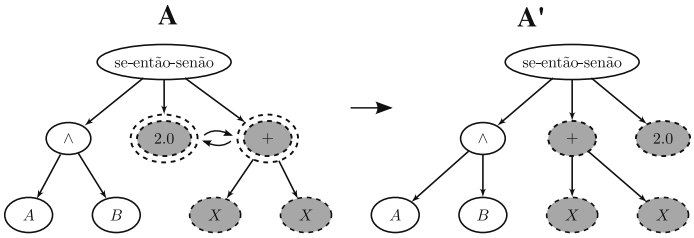


Figura 4.9: O processo de mutação tipo *permutação*.

Métodos de Reprodução

Os métodos de reprodução dizem respeito à política sob a qual os descendentes recém criados são introduzidos no processo evolucionário. Existem dois métodos fundamentais: *geracional* e *steady-state* (Syswerda, 1989; Whitley, 1989).

O método geracional foi o primeiro a surgir na literatura dos algoritmos de computação evolucionária; funciona basicamente assim: a cada geração os descendentes recém criados são alocados em uma população temporária; quando o número de indivíduos criados se iguala ao da população principal, a população temporária substitui a principal, e assim todos os indivíduos da geração passada são descartados—uma variação muitas vezes útil, conhecida por *elitismo*, é preservar os n melhores indivíduos (a "elite") da geração passada.

Uma desvantagem do método geracional, que pode ser decisiva em certas aplicações, é a necessidade de se alocar uma segunda população durante o processo evolutivo. O método de reprodução *steady-state*, por outro lado, insere os descendentes recém criados imediatamente na população atual, passando assim a coexistir com seus antecedentes. Quando um novo descendente é inserido, um indivíduo da população vigente deve ser eliminado para que o tamanho da população permaneça constante. Habitualmente, escolhe-se para ser substituído um indivíduo pouco adaptado (ou o menos adaptado), dessa forma, a evolução descarta soluções pouco promissoras e obtém-se, naturalmente, o

efeito de conservação dos melhores indivíduos análogo ao *elitismo* da reprodução geracional.

2. Algoritmo Protótipo

A programação genética, assim como a maioria das técnicas da computação evolucionária, não define passos algorítmicos precisos e imutáveis, sendo portanto, em sua essência, um modelo. Nesse sentido, é apresentado no Algoritmo 1 um protótipo em pseudo-código da PG segundo a reprodução do tipo geracional. Por sua vez, o Algoritmo 2 descreve a PG em sua variação de reprodução tipo *steady-state*.

A título de simplificação na listagem, ambos os algoritmos adotam como critério de parada tão somente um número máximo de iterações (gerações ou avaliações, dependendo do tipo de reprodução). Normalmente, entretanto, seria conveniente checar a cada avaliação de indivíduo se a solução candidata encontra-se dentro de uma faixa admissível de qualidade. Outros critérios de parada, como os descritos anteriormente, também podem ser usados.

Algoritmo 1 Programação Genética (reprodução geracional).

```
1: Criar aleatoriamente a população inicial P
2: Avaliar todos indivíduos de P
3: para geração ← 1 to N_G faça
4:    Copiar a elite de indivíduos de P para a população temporária P_tmp
5:    enquanto |P_tmp| < |P| faça
6:       Selecionar e copiar de P dois indivíduos bem adaptados, p_1 e p_2
7:       se [probabilisticamente] cruzamento então
8:          Cruzar p_1 com p_2, gerando os descendentes p'_1 e p'_2
9:          p_1 ← p'_1; p_2 ← p'_2
10:      fim se
11:      se [probabilisticamente] mutação então
12:         Aplicar operadores quaisquer de mutação em p_1 e p_2, gerando p'_1 e p'_2
13:         p_1 ← p'_1; p_2 ← p'_2
14:      fim se
15:      Avaliar p_1 e p_2 e inseri-los em P_tmp
16:   fim enquanto
17:   P ← P_tmp; descartar P_tmp
18: fim para
19: retorno melhor indivíduo encontrado
```

Parâmetros

Na Tabela 4.1 são descritos os parâmetros usuais cujos valores podem ser ajustados em execuções da PG para um certo problema. Em se tratando de valores ótimos para estes parâmetros, comumente diferentes problemas requerem diferentes valores de parâmetros; em outras palavras, dificilmente um conjunto ótimo de parâmetros para um problema será ótimo em outro problema.

Execução de Programas

Fundamental à avaliação dos indivíduos é a execução propriamente dita do programa codificado; por meio desta é possível medir quão bem o indivíduo desenvolve a tarefa em questão. O Algoritmo 3

Algoritmo 2 Programação Genética (reprodução *steady-state*).

1: Criar aleatoriamente a população inicial P
2: Avaliar todos indivíduos de P
3: **enquanto** número de *avaliações* $< N_A$ **faça**
4: Selecionar e copiar de P dois indivíduos *bem adaptados*, p_1 e p_2
5: **se** [probabilisticamente] *cruzamento* **então**
6: Cruzar p_1 com p_2, gerando os descendentes p'_1 e p'_2
7: $p_1 \leftarrow p'_1$; $p_2 \leftarrow p'_2$
8: **fim se**
9: **se** [probabilisticamente] *mutação* **então**
10: Aplicar operadores quaisquer de mutação em p_1 e p_2, gerando p'_1 e p'_2
11: $p_1 \leftarrow p'_1$; $p_2 \leftarrow p'_2$
12: **fim se**
13: Avaliar p_1 e p_2 e inseri-los em P, substituindo indivíduos *mal adaptados*
14: **fim enquanto**
15: **retorno** melhor indivíduo encontrado

Parâmetro	**Descrição**
Tamanho da população ($\|P\|$)	Número de indivíduos em evolução
Número de gerações (N_G)	Número máximo de gerações
Número de avaliações (N_A)	Número máximo de avaliações ("execuções")
Probabilidade de cruzamento	Probabilidade de aplicação do operador de cruzamento (tipicamente *alta*)
Probabilidade de mutação	Probabilidade de aplicação do operador de mutação (tipicamente *baixa*)
Pressão de seleção	Importância dada aos indivíduos mais bem adaptados durante o processo de seleção
Limites de profundidade	Profundidade máxima das árvores geradas
Tamanho da elite (geracional)	Quantidade de indivíduos mais bem adaptados que serão copiados para geração seguinte
Conjuntos de funções e terminais	Operadores e operandos primitivos que estabelecem as instruções básicas dos programas

Tabela 4.1: Parâmetros usuais da programação genética.

mostra em pseudo-código uma função recursiva para execução de programas codificados pela estrutura árvore.[11]

A execução de um programa inicia-se pela chamada da função Executa() sobre o nó raiz da árvore representada pelo indivíduo sendo avaliado. O algoritmo então testa o tipo do nó: se for *terminal* (nó folha) o seu valor é retornado imediatamente, e o percurso da recursão daquele ramo é finalizado; caso seja o tipo *função* (nó interno), o operador codificado pelo nó é aplicado sobre os seus argumentos[12] (nó filhos)—cujos valores são obtidos recursivamente—e finalmente o valor computado é devolvido. Comparando-se o valor final retornado por Executa() (ou os valores de sucessivas execuções) com o valor esperado obtém-se a discrepância do programa frente à tarefa alvo.

Um *terminal* pode codificar basicamente três espécies de valores: (i) constantes; (ii) variáveis; e (iii)

[11] Se o tempo de execução for crucial então uma versão não-recursiva do algoritmo seria mais apropriada, embora possivelmente menos elegante.
[12] O número de argumentos requerido pelo operador coincide com o número de filhos do nó.

Algoritmo 3 Execução de um programa.

EXECUTA(nó $raiz$)

1: **se** $raiz$.**tipo** = terminal **então**
2: **retorno** $raiz$.**valor**()
3: **fim se**
4: **retorno** $raiz$.**operador**(EXECUTA($raiz.filho_1$), EXECUTA($raiz.filho_2$), ...)

funções que não requerem argumentos. O tratamento da sentença "$raiz$.**valor**()" nos casos (i) e (iii) é trivial: basta que a constante seja retornada ou o valor da função, que não precisa de argumentos, seja computada. Porém, no caso (ii), o valor da variável precisa antes ter sido atribuído. Normalmente, um programa é executado sobre um conjunto finito de pontos (amostras de treinamento), e as variáveis, em cada execução, devem representar os valores do ponto em questão. Dessa forma, imediatamente antes de cada execução do programa os valores relativos ao ponto corrente são atribuídos, respectivamente, às variáveis do conjunto de terminais. Este procedimento assegura que os valores retornados das variáveis na i-ésima execução correspondem aos valores do i-ésimo ponto.

3. Exemplo de Aplicação: Regressão Simbólica

No intuito didático, e visando introduzir os conceitos da programação genética em uma aplicação prática, esta seção explica o que é regredir simbolicamente uma expressão, como os métodos tradicionais implementam a "regressão" e qual a vantagem da utilização da PG sobre estes métodos. Adicionalmente, uma execução hipotética passo-a-passo da PG em um problema simples de regressão simbólica é exemplificada.

Regressão Simbólica de uma Expressão

Regredir simbolicamente uma expressão é o ato de inferir simbolicamente uma expressão algébrica, dispondo apenas de um conjunto finito de amostras numéricas.[13] Basicamente, procura-se uma curva que melhor se ajuste aos dados fornecidos. O objetivo é encontrar uma função que minimize uma medida dos erros (ou discrepâncias) entre os valores previstos pela função e aqueles de fato observados.

Seja F o conjunto de todas as funções $f : \mathbb{R}^n \mapsto \mathbb{R}$ admissíveis para o problema, $\{p_1, p_2, \ldots, p_k\}$ o conjunto de pontos conhecido como amostragem e $Y \in \mathbb{R}^k$ o vetor cuja componente Y_i é o valor observado em $p_i \in \mathbb{R}^n$, onde k é o número de pontos fornecidos. Então, regredir simbolicamente é buscar por uma função f^* tal que

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

onde $d(f, Y) \geq 0$ mede a discrepância entre os valores previstos por f e aqueles observados. A função f^* é dita a mais adaptada (ou melhor ajustada) aos pontos fornecidos. Uma possibilidade é fazer

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

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

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

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

e ainda outra é

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

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

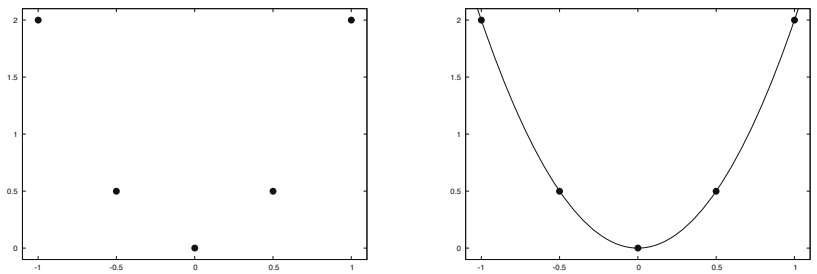
Exemplo de regressão simbólica

Como exemplo, suponha que se deseja descobrir qual a melhor função que se ajusta aos pontos (x_i, y_i) mostrados na Tabela 4.2.

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

Tabela 4.2: Pontos discretos da função exemplo.

São representadas graficamente na Figura 4.10 as amostras cujas coordenadas encontram-se na Tabela 4.2. Na Figura 4.10a são exibidos os pontos discretos, enquanto que a Figura 4.10b ilustra o ajuste exato desses pontos por meio da função $f(x) = 2x^2$.



(a) Pontos discretos

(b) Curva baseada na forma exata da função $f(x) = 2x^2$

Figura 4.10: Exposição gráfica dos pontos e da função exata com base nas coordenadas da Tabela 4.2.

Métodos Clássicos *versus* Programação Genética

Há na literatura várias técnicas de ajustes de funções, como a regressão linear, quadrática e polinomial, porém, estas limitam-se ao ajuste de coeficientes. Dessa forma, requerem um modelo de expressão definido, sendo fortemente dependentes da classe das funções, e por isso são específicas e pouco flexíveis. Isto é, considerando-se o exemplo em questão, seria fornecido ao método a forma pré-definida $f(x) = cx^2$ e este ajustaria apenas o coeficiente real c. Ou ainda, em um ajuste mais complexo, seria provido o modelo pré-estabelecido $f(x) = c_1 x^{c_2}$ e então o método, além do ajuste do coeficiente c_1, também seria responsável por ajustar o expoente c_2.

Muitas vezes, entretanto, não é suficiente ou desejável apenas a regressão linear, quadrática ou mesmo polinomial. Busca-se em muitos casos regredir, também, funções não lineares e sem qualquer forma ou complexidade pré-definida. Em outras palavras, deseja-se identificar simultaneamente a *estrutura* e os *coeficientes* da função, o que torna a tarefa vastamente mais árdua devido ao aumento

considerável do espaço de busca. A programação genética é capaz de gerenciar grandes e intricados espaços de busca, e não requer uma estrutura de solução definida, sendo portanto adequada a problemas de regressão simbólica de complexidade arbitrária.

Execução hipotética passo-a-passo

Esta seção apresenta uma execução hipotética passo-a-passo, extensamente simplificada, da programação genética no problema de regressão simbólica da função algébrica que se ajusta aos pontos da Tabela 4.2. Em outras palavras, o objetivo é evoluir a função $f(x) = 2x^2$ ou alguma identidade desta.

Não são relevantes neste exemplo didático alguns parâmetros e definições, são eles: o número máximo de gerações, a função específica de avaliação dos indivíduos, o modo de criação da população, o esquema e pressão de seleção, as probabilidades de aplicação dos operadores genéticos, e os limites de tamanho. Os parâmetros que possuem alguma relevância neste exemplo, no entanto, são listados na Tabela 4.3, onde é possível notar a simplicidade dos valores a fim de manter o problema ilustrativo e gerenciável. Foi escolhido para o conjunto primitivo de funções as operações aritméticas básicas, enquanto que o conjunto de terminais possui apenas a constante numérica 1 e, naturalmente, a variável X.

Parâmetro	**Valor**
Tamanho da população	4 indivíduos
Tipo de reprodução	geracional
Tamanho da elite	0 (sem elitismo)
Operadores genéticos	cruzamento e mutação padrão
Conjunto de funções	$\mathbb{F} = \{+, -, \times, \div\}$
Conjunto de terminais	$\mathbb{T} = \{1, X\}$
Critério de parada	discrepância nula (ajuste perfeito)

Tabela 4.3: Parâmetros do exemplo hipotético.

Sejam os quatro programas **A**, **B**, **C** e **D**, codificados por árvore, exibidos na Figura 4.11. Eles representam a população inicial, e portanto foram criados aleatoriamente por um método qualquer de criação. Imediatamente abaixo de cada um dos indivíduos é possível visualizar graficamente quão próximos seus ajustes estão da curva alvo representada pela função $f(x) = 2x^2$; isto fornece uma avaliação visual da qualidade do indivíduo.[14] Pode-se notar que os indivíduos **A** e **B** codificam a mesma função $f(X) = X$, embora o indivíduo **B** seja menos parcimonioso—o que não é necessariamente ruim, pois o material genético "redundante" pode ser útil ao processo evolucionário. Curiosamente, o indivíduo **C** não emprega a variável X, que representa a abscissa dos pontos do problema, ou seja, retorna sempre uma constante (no caso $1 + 1 = 2$) seja qual for o ponto; dessa forma, é muito pouco provável que estruturas como estas tenham prosperidade ao longo do processo evolutivo, exceto talvez como formas intermediárias ("blocos de construção"). Em termos de discrepâncias, percebe-se que os ajustes de **A**, **B**, e **C** são razoáveis, enquanto que o ajuste de **D** é nitidamente inferior aos dos demais.

Com base na qualidade dos ajustes, suponha que os indivíduos **B** e **C**, que pode-se dizer são relativamente bem adaptados, foram selecionados para cruzarem e assim deixar descendentes; processo este ilustrado na Figura 4.12a, onde os nós escurecidos denotam os pontos de cruzamento sorteados[15].

[14] Naturalmente, em problemas reais de regressão simbólica não se conhece a função que se deseja regredir—do contrário o problema já estaria resolvido—, e portanto a avaliação de um indivíduo ficaria restrita à medição da discrepância nos pontos fornecidos.

[15] Um dos pontos aleatoriamente escolhido foi o nó raiz do indivíduo **C**. Neste caso, a árvore inteira substitui o nó (ou sub-árvore, se fosse um nó interno) selecionado do indivíduo **B**, enquanto que o descendente **C'** será o próprio nó (ou

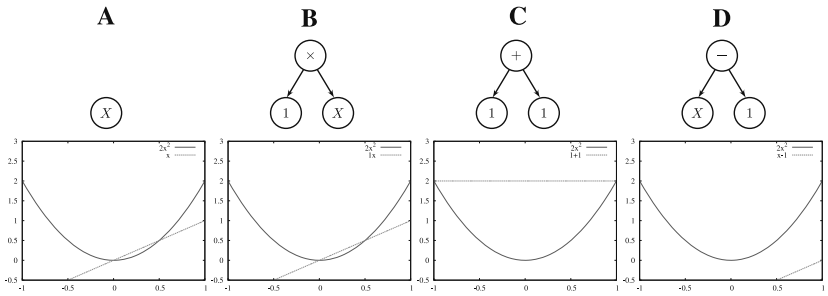
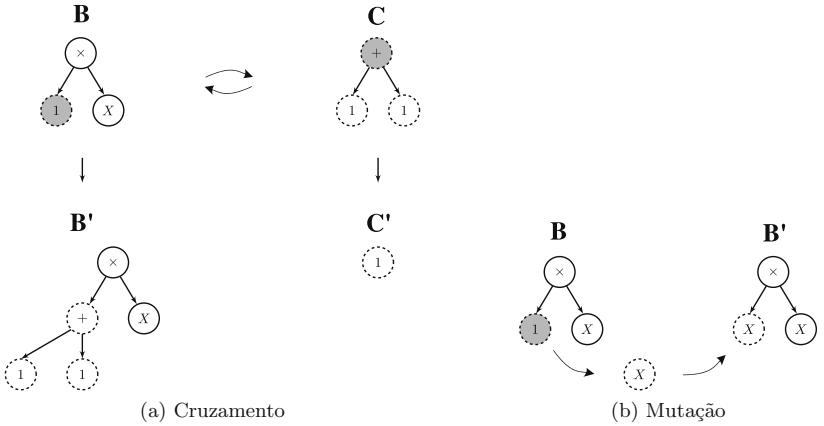


Figura 4.11: População inicial.



(a) Cruzamento (b) Mutação

Figura 4.12: Aplicação de operadores genéticos sobre a população inicial.

Esta recombinação deu origem aos indivíduos **B'** e **C'**, que então foram alocados na próxima geração (segunda geração, Figura 4.13) com os rótulos **A** e **B**, respectivamente.

Considere também que o indivíduo **B** da população inicial foi novamente selecionado, no entanto, desta vez ele foi submetido ao operador de mutação, como consta na Figura 4.12b, onde o primeiro operando da multiplicação (a constante 1) foi substituído pela variável X, dando origem à expressão $f(X) = X^2$. O programa derivado (**B'**) foi então alocado na geração seguinte com índice **C** (terceiro descendente gerado).

Suponha ainda que, dada a natureza probabilística de aplicação de operadores genéticos, que o indivíduo **A** ($f(X) = X$), da população inicial, tenha sido selecionado, mas ao invés de gerar uma estrutura derivada, foi simplesmente copiado sem modificação—isto é, clonado—à geração seguinte; em outras palavras, nem o cruzamento nem a mutação foram aplicados. Este indivíduo foi portanto inserido na próxima geração com rótulo **D**.

Percebe-se que, devido à baixa aptidão do indivíduo **D** da população inicial, este em nenhuma oportunidade foi escolhido pelo processo de seleção como genitor, portanto, o seu material genético não foi passado adiante—este processo de procriação a favor dos mais aptos é uma característica fundamental dos algoritmos da computação evolutiva.

Com a população da segunda geração lotada (quatro indivíduos), mostrada na Figura 4.13, o processo iterativo dá início à formação da geração seguinte, valendo-se de sucessivas aplicações dos operadores genéticos sobre os indivíduos mais bem adaptados. Nota-se pelos gráficos que os indivíduos

sub-árvore) escolhido de **B**.

da segunda geração são melhor adaptados quando comparados à geração anterior (população inicial)—esta *tendência* de aperfeiçoamento das soluções ao decorrer da evolução é uma propriedade das técnicas evolucionárias.

Dando sequência à execução hipotética, suponha agora que, da segunda geração, foram selecionados os indivíduos promissores **A** e **C** para cruzamento. Há uma boa chance de que os pontos escolhidos de cruzamento sejam os mostrados pela Figura 4.14 (nós escurecidos). Esta recombinação produziu os indivíduos **A'** e **C'**. É fácil perceber, no entanto, que o descendente **A'** codifica a solução esperada para o problema, isto é, $f(X) = (1 + 1) \times (X \times X) = 2X^2$. Este ajuste é apresentado graficamente na Figura 4.15, onde, obviamente, a curva da função alvo e da função evoluída coincidem. Como a discrepância neste caso é nula, isto é, o critério de parada foi satisfeito, o programa **A'** é retornado como solução desta execução ilustrativa.

4. Conclusões

Este capítulo apresentou a metaheurística evolucionária chamada programação genética, cuja especialidade é a otimização de estruturas funcionais capazes de realizar operações (lógicas, aritméticas, condicionais e de desvios), ou seja, programas de computador. A PG é uma técnica de otimização global que é *robusta*, *exige pouco conhecimento acerca do domínio*, produz *soluções simbólicas*, possui *paralelismo natural*, e é facilmente *extensível* e *modificável*.

O objetivo final da programação genética é a implementação plena da ambiciosa virtude que permitiria ao usuário, frente à tarefa alvo, especificar ao computador simplesmente *o que se quer* que seja feito (isto é, resolvê-la) em vez de *como* fazer (codificação manual dos passos/instruções) para solucioná-la. Naturalmente, em decorrência das limitações tecnológicas atuais e mesmo das limitações da metaheurística em si—que está em constante evolução e apresenta pontos de potencial aperfeiçoamento—, esta virtude ainda é um ideal a ser alcançado no que concerne à maioria dos problemas de interesse.

No entanto, dada a importância e apelo da tarefa de auto-programação, a PG tem atraído nos últimos anos esforços substanciais de pesquisa. Não é difícil imaginar que, ao passo que os computadores tornam-se mais acessíveis e poderosos, a PG torna-se capaz de resolver problemas mais complexos e extraordinários, por conseguinte despertando o interesse geral e inevitavelmente estimulando o desenvolvimento da área—que por sua vez expande o domínio de aplicação da metaheurística e novamente alavanca interesses e esforços de pesquisas, realimentando o círculo. A despeito deste futuro promissor, atualmente a PG já goza de um currículo expressivo de resultados obtidos que são tão bons ou melhores do que os produzidos por humanos, dentre estes descobertas patenteáveis (Poli et al., 2008a).

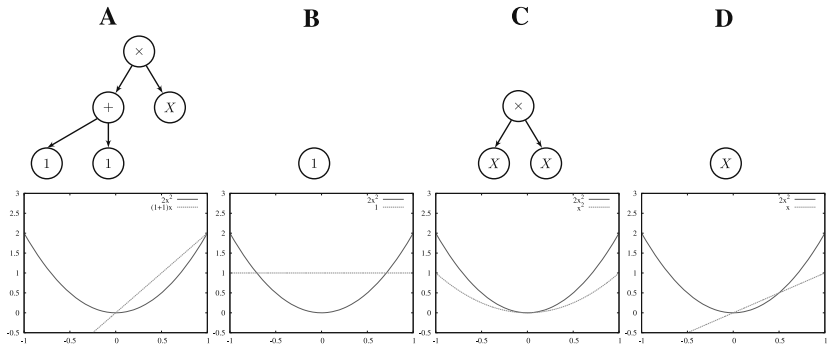


Figura 4.13: Segunda geração de indivíduos.

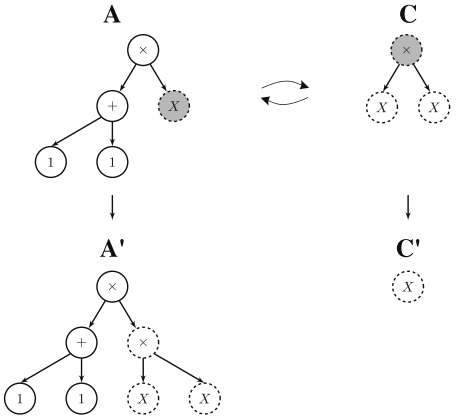


Figura 4.14: Aplicação do operador de cruzamento.

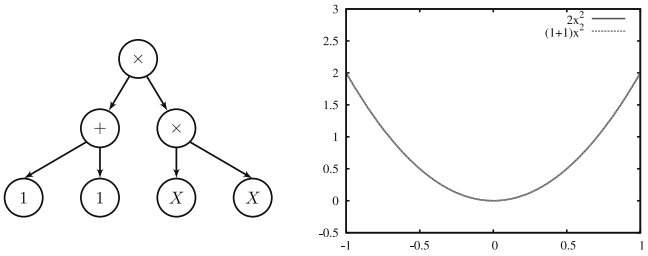


Figura 4.15: Terceira e última geração (solução encontrada).