3
Programação Genética

O termo "Programação Genética" passou a ser utilizado em 1990 nos trabalhos publicados por Koza [30] e De Garis [31]. A definição de Koza para este termo passou a predominar após a publicação de seu livro sobre este tema em 1992 [2]. A maioria dos pesquisadores passou então a representar a evolução de estruturas de programas por árvores ou, mais especificamente, na forma de programas em linguagem LISP. Entretanto, atualmente entende-se por programação genética toda forma evolutiva de indução de programas [64].

A PG faz parte de um paradigma maior: a computação evolutiva. Esta, por sua vez, é composta por algoritmos estocásticos inspirados no princípio da evolução das espécies, sendo os algoritmos genéticos (AGs) sua forma mais popular. Os AGs foram introduzidos por Holland [65] e são baseados no princípio da seleção natural de Darwin, segundo o qual uma população de indivíduos evolui, através de gerações ou ciclos, pela sobrevivência dos mais aptos.

Já a PG pode ser vista como sendo o uso de AGs para evoluir processos computacionais, como programas de computador. Em AGs, os indivíduos de uma população podem ser representados e interpretados de várias formas, enquanto que em PG eles são tratados explicitamente como programas escritos em um subconjunto ou variação de uma linguagem de programação convencional [66].

O algoritmo básico de um AG, de forma geral, é mantido em PG: a busca é feita avaliando-se iterativamente a aptidão dos indivíduos em uma população e aplicando-se operadores genéticos, como o cruzamento e a mutação, aos indivíduos selecionados probabilisticamente com base nas suas aptidões, de forma a explorar outras áreas promissoras do espaço de busca. Nos AGs, a avaliação da aptidão pode tomar várias formas. Enquanto que, em PG, um indivíduo tem sua aptidão avaliada, pelo menos em parte, pela execução do programa e pelo acesso à qualidade de suas saídas resultantes.

3.1
Algoritmo Básico

Na Programação Genética, uma população de programas de computador é evoluída. Cada programa de computador é chamado de *indivíduo* e representa uma solução potencial para o problema. O algoritmo básico segue o descrito na Figura 3.1 e é definido por [67]:

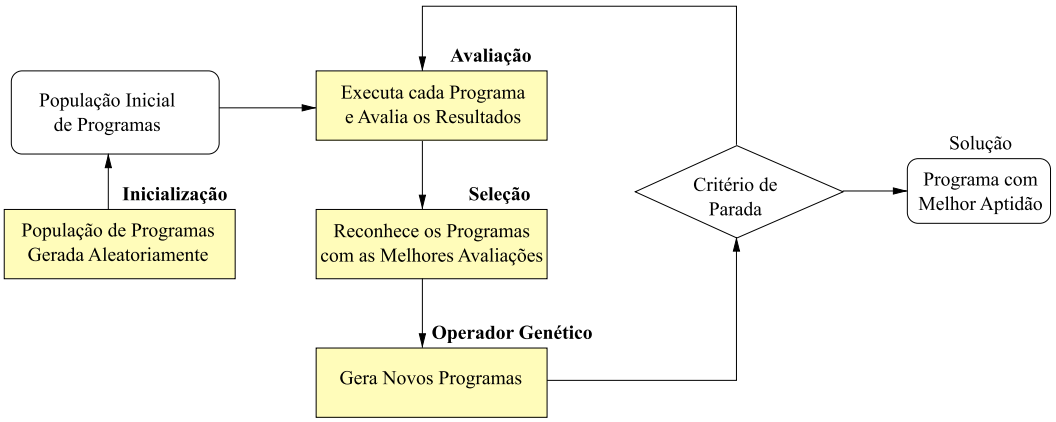

Figura 3.1: Fluxograma do algoritmo da Programação Genética.

1. *Conjunto de Terminais*: Um conjunto de variáveis de entrada ou constantes.

2. *Conjunto de Funções*: Um conjunto de instruções que, combinadas com os terminais, são usadas para construir um programa.

3. *Função de Avaliação da Aptidão*: Um valor de pontuação atribuído a cada indivíduo que fornece uma medida da aptidão do programa para resolver o problema, o que significa o quanto os resultados da execução do programa estão próximos dos resultados desejados. Por exemplo, no benchmark de regressão simbólica *Salutowicz* [68] o Erro Absoluto Médio (MAE) é usado como Função de Avaliação da Aptidão.

4. *Critério de Parada*: Pode ser um número pré-definido de gerações ou um valor de tolerância ao erro para a função de avaliação da aptidão. Estes critérios podem afetar o desempenho e a qualidade dos resultados.

O algoritmo inicializa através da criação e uma população inicial de indivíduos. Os indivíduos são criados aleatoriamente. Depois disso, cada indivíduo é executado e seu valor para a função de avaliação da aptidão é calculado. Os indivíduos com boa aptidão são reconhecidos e escolhidos para gerar novos indivíduos. A geração de novos indivíduos consiste em aplicar *operadores genéticos*, que produzem diversificação da busca. Os novos indivíduos seguem os mesmos passos até que o critério de parada seja atingido.

3.2 Programação Genética com Representação dos Indivíduos em Árvore

Uma das abordagens mais antigas e mais comumente empregadas de programação genética é a representação dos indivíduos com estruturas em árvore. Esta estrutura é utilizada durante a evolução através da utilização

de expressões de comprimento variável de uma linguagem de programação funcional, tal como a linguagem LISP [2]. Os nós mais externos da árvore, chamados de terminais ou nós folhas, são utilizados para representar as variáveis de entrada e também as constantes. Os nós internos são utilizados para representar as funções ou instruções, que podem ser operações aritméticas ou booleanas.

Para realizar a avaliação de um indivíduo, é necessário percorrer recursivamente a árvore em uma ordem pré-definida, através da utilização de interpretadores. Inicialmente, os valores das constantes e das variáveis de entrada que encontram-se nos nós folhas são utilizados para calcular os valores dos nós internos mais próximos. Este processo é repetido ciclicamente até que o valor final seja encontrado no último nó da árvore, representando a saída do programa ou o resultado da avaliação do indivíduo. Uma das formas de representar programas com múltiplas saídas é através da utilização da representação de um indivíduo com a utilização de múltiplas árvores, uma para cada saída.

O operador de cruzamento ou *crossover* é utilizado para recombinar soluções antigas ou indivíduos já avaliados e gerar novas soluções ou indivíduos potencialmente melhores. A Figura 3.2 ilustra a representação dos indivíduos em estruturas em árvore e a aplicação do operador de cruzamento. Nesta figura estão representados quatro indivíduos, sendo dois genitores e dois descendentes. O operador de cruzamento é aplicado em cada um dos indivíduos genitores, através da seleção aleatória de um ponto de corte em cada uma das duas árvores, dividindo cada uma das duas árvores em duas subpartes. Estas subpartes das árvores são trocadas entre os indivíduos genitores para gerar os indivíduos descendentes.

O operador de mutação é aplicado nos nós internos, através da seleção aleatória de um dos nós da árvore e da substituição da função deste nó por outra função escolhida aleatoriamente, dentro do conjunto das possíveis soluções, que necessita do mesmo número de argumentos da função substituída.

A programação genética com representação dos indivíduos em árvore utiliza alguns parâmetros, entre eles é utilizado um parâmetro para especificar a profundidade máxima da árvore, evitando que o programa com representação em árvore torne-se desnecessariamente muito grande. Além disso, a inicialização dos indivíduos geralmente é feita aleatoriamente, através da escolha aleatória de funções válidas para os nós internos e de variáveis de entrada ou constantes para os nós folhas.

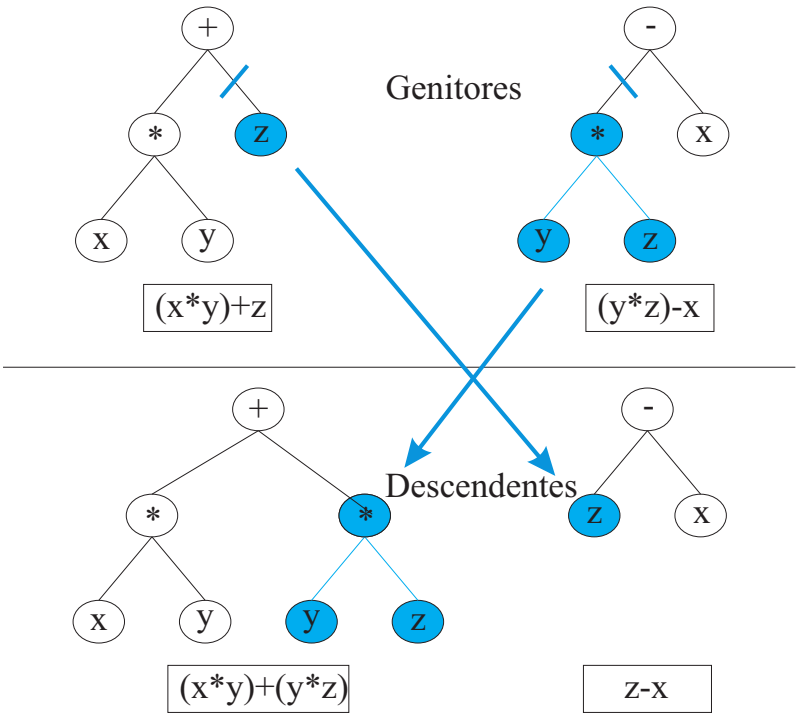

Figura 3.2: Operador de cruzamento ou *crossover*: quatro indivíduos são representados, dois genitores e dois descendentes, um ponto de corte é selecionado aleatoriamente nos indivíduos genitores, e a troca das subpartes das árvores dos indivíduos genitores gera os indivíduos descendentes.

3.3 Programação Genética Linear

Na programação genética linear, a representação dos programas é uma sequência linear de instruções, de tamanho variável. Sua principal característica, quando comparada à PG tradicional baseada em estruturas em árvores, é que são evoluídos programas em uma linguagem imperativa, como C [69] e código de máquina [33, 70]. Diferente de expressões de uma linguagem de programação funcional, como LISP [2].

A implementação da PG linear proposta por [69] representa um programa individual como uma lista de tamanho variável de instruções, as quais operam com variáveis indexadas ou constantes. Em PG linear, todas as operações, como por exemplo $a = b+1$, incluem implicitamente uma atribuição a uma variável. Depois que um programa é executado, os valores de saída são armazenados nas variáveis designadas. Na PG baseada em árvores, designações e saídas múltiplas têm que ser incorporadas explicitamente pelo uso de uma memória indexada extra e de funções especiais para tal manipulação.

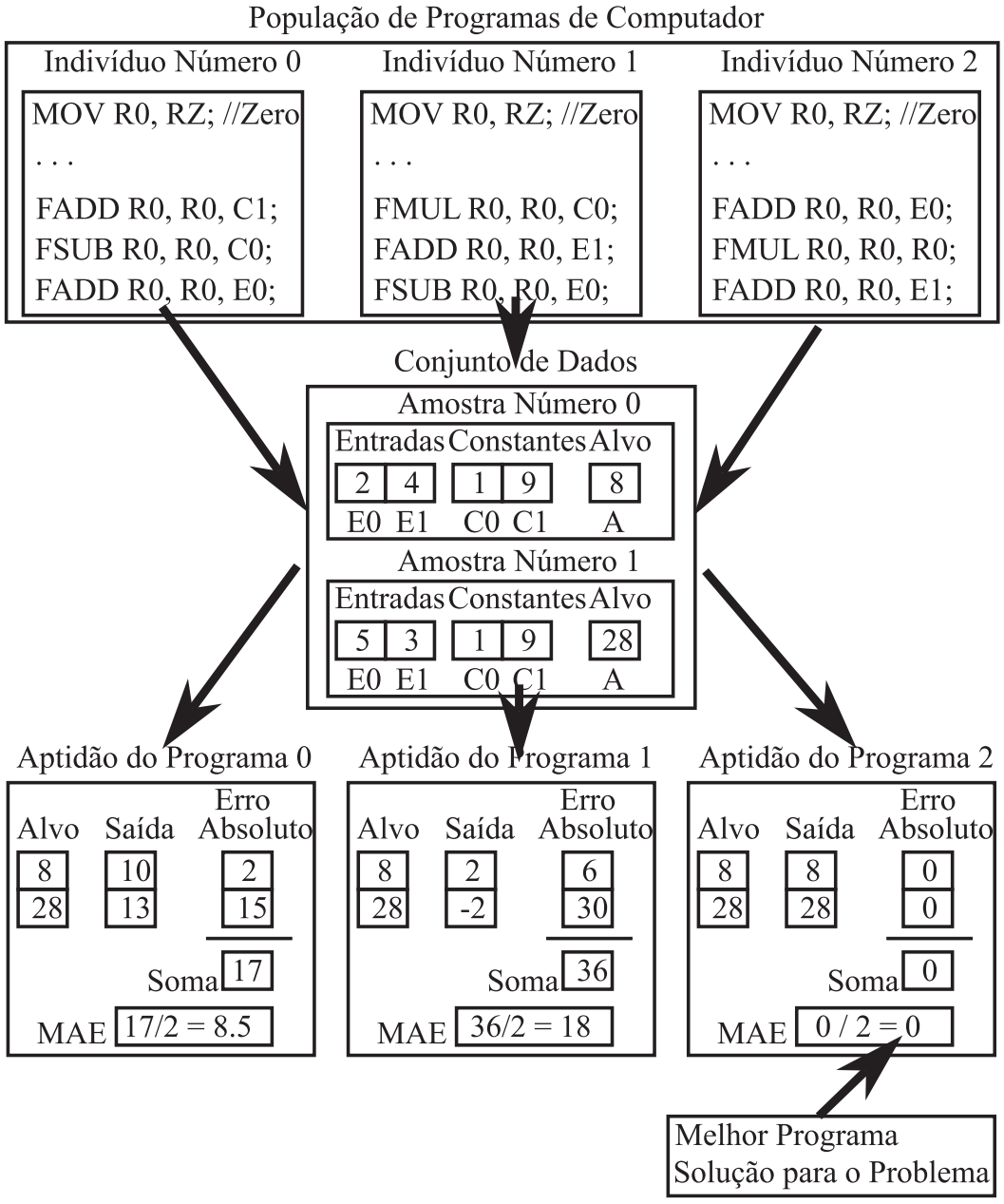

Figura 3.3: Cálculo da Função de Avaliação da Aptidão para os indivíduos da população de programas de computador em Programação Genética Linear.

A Figura 3.3 ilustra o processo de avaliação dos indivíduos da população em Programação Genética Linear, onde é calculado o valor da aptidão de cada indivíduo. Este exemplo simplificado apresenta uma população de programas de computador composta por três indivíduos. Cada indivíduo tem um máximo de três instruções, as quais são selecionadas pelo algoritmo de programação genética. Todos os indivíduos da população são executados utilizando o mesmo conjunto de dados, o qual é composto, neste exemplo simplificado, por apenas duas amostras de dados. O erro absoluto é a diferença entre a saída produzida

pela execução de cada indivíduo sobre o conjunto de dados e o valor alvo ou valor real de saída para cada amostra de dados. O sinal é desconsiderado uma vez que está sendo calculado o valor absoluto do erro. Após calcular o erro absoluto, é feita a divisão pelo número de amostras, duas neste exemplo simplificado, para obter o Erro Absoluto Médio (MAE) ou valor para a função de avaliação da aptidão para cada indivíduo. O indivíduo com o menor valor para o MAE é selecionado como sendo o melhor indivíduo.

3.4 Evolução de Programas em Linguagem de Montagem ou Linguagem de Máquina

A programação em linguagem de montagem ou linguagem de máquina é frequentemente usada quando há a necessidade de soluções muito eficientes, como aplicações que tenham fortes restrições de tempo de execução e de uso de memória. Em geral, as razões para se evoluir linguagem de montagem, em contraste com linguagens de alto nível, são similares às razões de se programar manualmente em linguagem de montagem (*assembly*), pelo uso de mnemônicos, ou em código de máquina:

- A otimização mais eficiente é obtida neste nível. Este é o nível mais baixo para se otimizar um programa e é onde também os maiores ganhos são possíveis. A otimização pode ser por velocidade, espaço ou ambos. Pequenos indivíduos podem ser promovidos através de uma punição por comprimento, geralmente chamada de pressão parcimoniosa (*parsimony pressure*) [71]. De forma semelhante, esta pressão pode ser aplicada considerando-se o tempo de execução das instruções.
- A linguagem de montagem é normalmente considerada difícil de se aprender, de se programar e de se dominar. Às vezes pode ser mais fácil deixar o computador evoluir pequenos programas em linguagem de montagem do que aprender a dominar a técnica de se programar neste nível.