Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Programação Genética

Programação Genética (PG) é um dos ramos da Computação Evolucionária (CE), sendo o membro mais jovem da família de algoritmos evolucionários. Esta técnica se propõe a encontrar novos programas que possam resolver problemas do mundo real.

Considerando um algoritmo de PG como um algoritmo evolucionário, percebe-se que a busca por novos programas acontece de forma evolutiva, assim como ocorre nos Algoritmos Genéticos (AG), estudados no capítulo anterior. Os operadores da PG também se assemelham aos operadores de um AG, portanto, os termos mutação, recombinação e seleção de indivíduos ocupam um lugar comum entre as técnicas.

A forma mais simples de entender PG comparando com AG, é considerando um indivíduo do AG como um conjunto de parâmetros do problema a ser solucionado, e um indivíduo da PG como sendo um programa que resolve o problema a ser solucionado. Este aspecto por si só acrescenta uma grande complexidade à técnica e justifica a existência de livros inteiros sobre o assunto. Portanto, a proposta deste capítulo é introduzir a técnica, cobrindo os algoritmos mais utilizados na prática atual de PG, sem a pretensão, no entanto, de esgotar o assunto.

1.1 Motivação e Inspiração

A inpiração natural para desenvolvimento de Programação Genética é basicamente a mesma inspiração compartilhada entre as técnicas evolucionárias.

Na natureza, estruturas biológicas que possuem maior capacidade de se adaptar ao seu ambiente sobrevivem e se reproduzem a uma taxa mais elevada. Biólogos interpretam o aparecimento ou modificação de estruturas que observamos na natureza como conseqüência da operação de seleção natural de Darwin em um ambiente durante um período de tempo. Em outras palavras, na natureza, o resultado das modificações ocorridas nas espécies é a conseqüência da aptidão. A aptidão causa, ao longo de um período de tempo, a criação de novas

estruturas por seleção natural e pelos efeitos criativos de recombinação sexual (cruzamento genético) e mutação.

Os programas de computador estão entre as mais complexas estruturas criadas pelo homem. O objetivo deste capítulo é o de aplicar a noção de que novas estruturas podem surgir usando o conceito de aptidão para resolver uma das questões centrais em ciência da computação (atribuída a Arthur Samuel na década de 1950):

> "Como os computadores podem aprender a resolver problemas sem estarem explicitamente programados? Em outras palavras, como os computadores podem ser criados para fazer o que precisava ser feito, sem ser dito exatamente como fazê-lo?"

1.2 Histórico

Vários pesquisadores na década de 1980 deram contribuições na área de programação automática de computadores, notadamente S. F. Smith em 1980 [7] e N. Cramer em 1985 [2]. Mas principalmente devido às contribuições de John R. Koza, o campo deslanchou a partir de 1990. O livro publicado por Koza em 1992 favoreceu a popularização da técnica a medida que expôs exemplos práticos de que o paradigma de PG funciona, usando evidências empíricas em uma grande variedade de problemas [4].

Pesquisadores da área acreditam em uma completa revolução da computação devido à PG. O cenário que está sendo vislumbrado é o de recursos computacionais ociosos devido à falta de mão de obra, ou seja, de programadores, capazes de produzir código-fonte para solução de problemas.

Sobre esse aspecto, pode ser feito um paralelo da indústria de software com o ocorrido na indústria do papel Idade Média [8]. No século 14, a Europa foi inundada de papel, as fábricas produziam muito e os preços caíram vertiginosamente. Não foi possível aproveitar essa abundância de recursos pelo fato de não existirem um número de escribas com conhecimento suficiente tanto para cópia quanto para elaboração de novos textos. Havia pressão para descoberta de um método mais eficiente para copiar textos. Neste cenário, a invenção da imprensa por Johannes Gutenberg e a propagação da nova tecnologia, foi um marco no desenvolvimento da humanidade.

Atualmente, é como se estivéssemos na Idade Média em termos de codificação artesanal de computadores. Um programador deve especificar, codificar e testar cada programa. A indústria de hardware desenvolveu-se de tal forma e temos tantos recursos computacionais disponíveis a preços acessíveis que parece não haver mão-de-obra qualificada para acompanhar a demanda da indústria de software. Desta forma, o paralelo entre os dois momentos históricos parece ser inevitável.

Evidentemente, os pesquisadores ainda não vislumbram PG como um método automático para invenção de quaisquer novos programas sob os moldes descritos por Arthur Samuel na Seção 1.1. Contudo, já foram obtidos resultados palpáveis

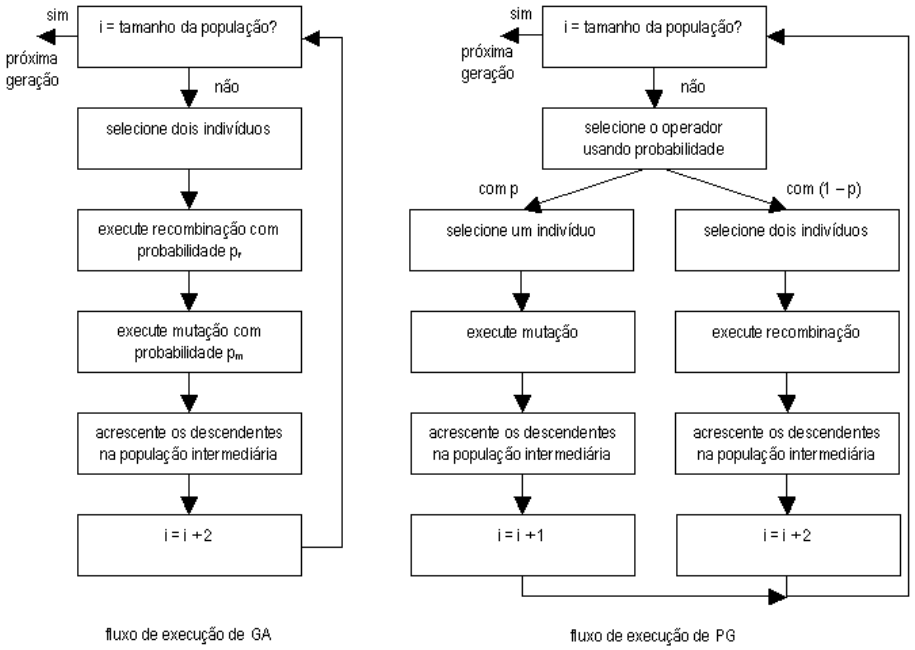

Figura 1.1: Fluxograma de execução comparativo de PG e AG.

em várias áreas do conhecimento usando PG. Já foram registrados 36 problemas em que PG produziu um resultado competitivo com o realizado por um pesquisador humano. Destes trabalhos, 21 deles correspondem a soluções que infringiam ou duplicavam patentes já registradas e em dois casos a PG criou uma invenção completamente nova e patenteável [5].

Talvez as próximas gerações ainda testemunhem uma das maiores revoluções da humanidade, rivalizando a invenção da imprensa há 400 anos atrás.

1.3 Visão Geral

Conforme já mencionado, um algoritmo da PG está preocupado em encontrar soluções que sejam elas próprias um programa passível de execução. Para Eiben PG pode ser vista como "programação de computadores através da seleção natural"ou "a evolução natural de programas de computadores" [1].

Analisando o fluxo de execução da PG percebem-se outras peculiaridades se compararmos a outros algoritmos evolucionários. No caso de PG normalmente usa-se mutação ou recombinação em um passo. O contraste neste ponto reside no fato de normalmente um AE usar ambos operadores de recombinação e mutação seguidamente no mesmo passo. Na Figura 1.1 comparamos um fluxograma típico de AG com um fluxo de execução da PG.

Conforme será visto nas seções 1.4 e 1.7, a semelhança de PG com as outras

```
int numero = 2;
if (numero < 3)
    numero *= numero;
```

Figura 1.2: Um programa na linguagem Java.

técnicas se resume apenas a forma evolutiva de busca por soluções, dada a peculiaridade na representação e nos operadores envolvidos. A Tabela 1.1 fornece um resumo desta técnica.

Tabela 1.1: Resumo de PG.

Representação	Estrutura de árvore
Recombinação	Troca de sub-árvores
Mutação	Geração aleatória em árvores
Seleção de pais	Proporcional à aptidão
Seleção de sobreviventes	Substituição dos pais

1.4 Representação de Indivíduos

Em PG os indivíduos são representados por árvores. Estas árvores representam expressões que possuem uma sintaxe predefinida. Dependendo do problema considerado, esta sintaxe pode representar uma expressão aritmética, fórmulas da lógica de 1a ordem, ou código-fonte de alguma linguagem de programação. Para entender melhor a diversidade dos tipos de expressões da PG e as respectivas representações em estrutura de árvore, considerem os exemplos a seguir:

- Uma fórmula aritmética:
$$2x^2 - 4x + 10 \tag{1.1}$$
- Uma fórmula lógica:
$$(p \vee true) \rightarrow (p \wedge q) \vee (\neg q) \tag{1.2}$$
- Um programa escrito em uma linguagem de programação de alto nível, como a linguagem Java (Fig. 1.2):

As estruturas de árvore para as Eq. 1.1, 1.2 e para o programa da Fig. 1.2, estão representadas pelas figuras 1.3 e 1.4, respectivamente.

Sob uma ótica estritamente técnica, pode-se dizer que PG é simplesmente uma variação de um AG que usa uma estrutura de dados diferente: os indivíduos são árvores. Tradicionalmente, essa representação em árvore incentivou os pesquisadores da área a adotarem linguagens de programação funcionais

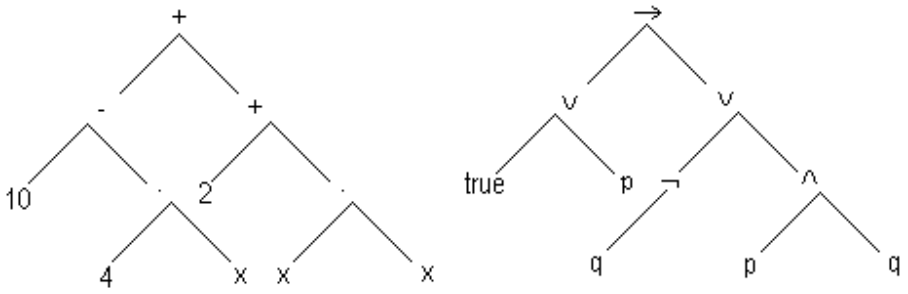

Figura 1.3: Árvores da Eq. 1.1 (esquerda) e Eq. 1.2 (direita).

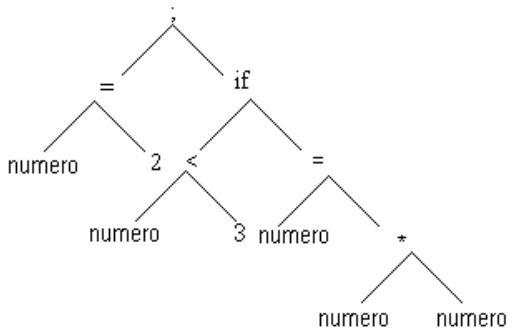

Figura 1.4: Árvore do programa Java da Fig 1.2.

na implementação dos algoritmos, destacando a linguagem LISP. A sintaxe de LISP se assemelha muito à notação polonesa de expressões. Para exemplificar, a notação polonesa para a fórmula da Eq. 1.1 seria:

$$+((-(.(2,.(x,x)),.(4,x))),10),$$

enquanto que o código executável em LISP seria semelhante a este trecho:

$$(+(-(.2(.xx))(.4x))10).$$

A tarefa de especificar a estrutura de um indivíduo recai, portanto, na tarefa de definir a sintaxe das expressões simbólicas. Ou seja, precisamos definir uma gramática da linguagem que os indivíduos pertencem. Isto geralmente é feito definindo-se um conjunto de símbolos **terminais** e um conjunto de **não-terminais**. No caso de problemas envolvendo expressões aritméticas, os terminais seriam as variáveis ou os números reais e os não-terminais seriam as funções matemáticas. Para o problema descrito pela Eq. 1.1 a sintaxe pode ser definida pela Tabela 1.2.

Tabela 1.2: Conjunto de terminais e de funções para definição da sintaxe do problema da Eq. 1.1

Conjunto de funções (símbolos não-terminais)	{+, -, ., /}
Conjunto de símbolos terminais	$\mathbb{R} \cup \{x, y\}$

.

Normalmente não se distingue entre diferentes tipos de expressões, no sentido de que cada símbolo de função pode receber qualquer outra expressão como argumento. Esta característica é conhecida como **propriedade de clausura** em PG.

Embora a maioria dos textos da área tradicionalmente usem a linguagem LISP como linguagem-alvo das soluções da PG, neste trabalho é adotada a linguagem de programação Java. Já ficaram evidentes vantagens em adotar uma linguagem como LISP devido a compatibilidade com a notação polonesa mencionada acima. Serão destacadas agora a motivação para escolher uma outra linguagem.

Primeiramente, Java tem se mantido como a linguagem de programação mais popular nos últimos 10 anos, de modo que adotando esta linguagem pode-se alcançar uma quantidade maior de leitores. As características da própria linguagem no tocante aos conceitos de orientação a objetos, e especialmente a facilidade de reuso, nos estimula a usá-la tanto no projeto dos algoritmos de PG quanto na própria estrutura sintática das soluções do nosso programa evolutivo. Além disso, Java possui uma gama de ferramentas que facilitam o processo de análise sintática e geração de código. Também há exemplos de trabalhos na área de PG que utilizaram Java como linguagem de programação [3].

1.5 Inicialização

O método mais comum de inicialização em PG é denominado de *ramped half-and-half*. Nessa estratégia define-se a profundidade máxima D_{max} das árvores e então cada indivíduo da população inicial é criado usando os conjuntos de terminais (T) e não-terminais ou funções (F) com igual probabilidade. Os métodos mais comuns para geração dos ramos das árvores são:

- Método completo: cada ramo da árvore tem profundidade D_{max}. O conteúdo dos nós de profundidade d são escolhidos de F se $d < D_{max}$ ou de T se $d = D_{max}$.

- Método de crescimento: os ramos podem ter profundidades diferentes, limitadas a D_{max}.

1.6 Avaliação de Sucesso

As funções de avaliação da PG atuam de forma semelhante à observada nas funções encontradas nos outros ramos de CE, ou seja, obtemos um valor de aptidão para cada indivíduo avaliado.

Em situações típicas têm-se pares de entradas e saídas de um problema e precisa-se descobrir a função que aproxima esses exemplos. Neste tipo de situação, a avaliação de um indivíduo consistirá na área entre duas curvas, sem considerar sinal, e o problema torna-se um problema de minimização da distância entre as curvas.

A implementação natural consiste em calcular a distância entre os pontos conhecidos e aqueles calculados pela função candidata referente ao indivíduo avaliado. A fórmula usual para este cálculo é a da distância euclidiana, reproduzida na Eq. 1.3.

$$d(y, \overline{y}) = \sqrt{|y_1 - \overline{y}_1|^2 + |y_2 - \overline{y}_2|^2 + ... + |y_n - \overline{y}_n|^2}, \tag{1.3}$$

onde y é a função representada no indivíduo e $\overline{y}$ o conjunto de valores fornecidos como entrada do problema.

O problema de adotar a distância euclidiana é quando há a valores muito altos ou muito baixos dentre os exemplos da amostra (*outlier*). Estes pontos podem aumentar muito a distância entre os elementos, distorcendo a avaliação dos indivíduos. A solução para contornar este problema é buscar por estes pontos e eliminá-los da amostra.

Finalmente, é válido reforçar que a avaliação de um indivíduo em PG consiste em executar o programa correspondente a solução testada e verificar a aptidão do programa. Esse processo consome muito tempo, por conseguinte, a avaliação de indivíduos em PG é o passo mais custoso do processo evolutivo.

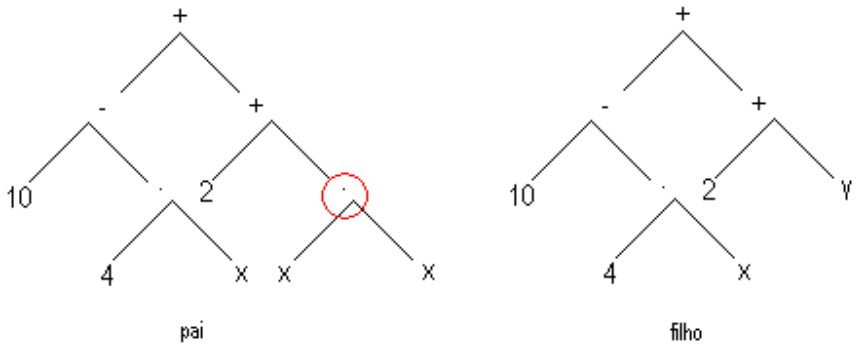

Figura 1.5: Mutação da árvore da Eq. 1.1. O nó destacado pelo círculo na árvore pai é substituído pela nova árvore gerada aleatoriamente, que consiste apenas da variável y.

1.7 Operadores

1.7.1 Mutação

Em princípio, o objetivo da mutação em PG seria o mesmo dos outros ramos de CE, ou seja, criar um indivíduo novo a partir de um existente através de uma pequena alteração randômica neste indivíduo, produzindo diversidade.

A forma mais comum de alcançar este efeito em PG é através da substituição de uma sub-árvore da solução atual, escolhida ao acaso, por uma outra sub-árvore, gerada aleatoriamente. Essa geração de sub-árvore ocorre de modo semelhante a geração de um indivíduo da população inicial.

A Figura 1.5 ilustra uma possível mutação na árvore referente à Eq. 1.1. Note que podemos incorrer na geração de um indivíduo maior do que o anterior, no tocante à profundidade da árvore gerada.

A mutação em PG deve considerar dois parâmetros:

- A probabilidade de realizar uma mutação no indivíduo considerado.
- A probabilidade de selecionar um dos nós da árvore representativa do indivíduo para ser substituído.

Assim como nas outras técnicas, é possível que PG funcione sem mutação, e é exatamente o que recomenda Koza em seu livro clássico sobre o tema [4]. Outros trabalhos sugerem uma taxa de mutação baixa, em torno de 5%. Um dos argumentos para reforçar uma taxa de mutação baixa ou mesmo para não usar a mutação, é que o operador de recombinação em PG de certa forma pode atuar como uma macro-mutação. A prática atual de PG, contudo, sugere o uso de uma taxa baixa de mutação e indica que o uso apenas de recombinação não fornece os melhores resultados.

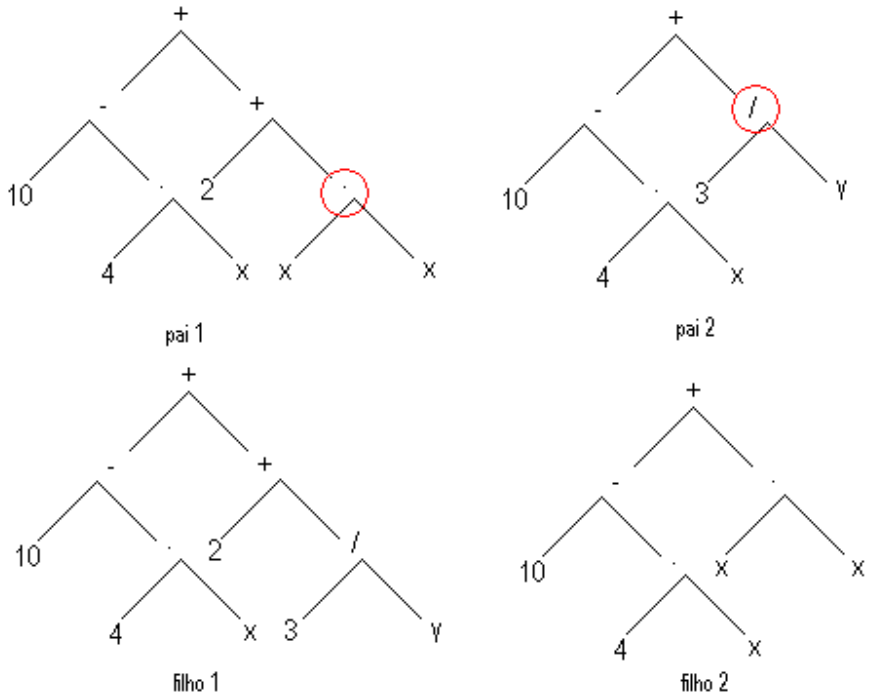

Figura 1.6: Operador de recombinação de sub-árvores. Os nós circulados nas árvores pai são usados como ponto de corte. As sub-árvores a partir dos nós circulados são trocadas, gerando dois novos indivíduos filhos.

1.7.2 Recombinação

Os operadores de recombinação criam descendentes através da troca de material genético entre os pais selecionados. Tecnicamente, em algoritmos de PG é um operador binário que cria duas novas árvores filhas através de duas árvores pais.

O tipo de recombinação mais comum em PG, o **crossover de sub-arvores**, seleciona um nó nas árvores pai de forma aleatória e troca as sub-ávores relacionadas entre os indivíduos. Assim com ocorre com o operador de mutação, as árvores filhas geradas podem ter uma profundidade diferente dos ancestrais. Uma ilustração deste operador de recombinação é visto na Figura 1.6.

O operador de recombinação deve considerar dois parâmetros:

- A probabilidade de realizar a recombinação entre dois indivíduos.
- A probabilidade de selecionar um dos nós em cada árvore-pai para servir como ponto de corte.

1.7.3 Seleção

Seleção dos Pais

A PG geralmente usa seleção proporcional à aptidão, contudo, devido ao tamanho típico de uma população ser na ordem de centenas, quiçá milhares de indivíduos, um método chamado *over-selection* geralmente é usado para populações acima de 1000 indivíduos.

Nesta metodologia a população é ordenada de acordo com a aptidão e então separada em dois grupos. O primeiro grupo conterá x% indivíduos e o segundo grupo (100 - x%). Na etapa de seleção dos pais, 80% dos selecionados devem ser provenientes do primeiro grupo e os 20% restantes do segundo grupo. Estes percentuais foram obtidos empiricamente e dependem do tamanho da população. A tabela 1.3 fornece os valores dos percentuais de indivíduos do primeiro grupo de acordo com o tamanho da população [1].

Tabela 1.3: Proporção de seleção de pais do grupo 1 (x) de acordo com o tamanho da população.

Tamanho da população	Proporção da população do grupo 1 (x)
1000	32%
2000	16%
4000	8%
8000	4%

.

Seleção de Sobreviventes

Nos algoritmos típicos de PG os pais são substituídos pelos descendentes, ou seja, o tempo de vida de um indivíduo é de uma geração e não há mecanismos de elitismo. Evidentemente, não há nenhuma restrição técnica para fixar essa estratégia, apenas é uma convenção histórica na área de PG. Há estudos recentes que sugerem que a inclusão de algum mecanismo de elitismo pode ser benéfica para evitar o efeito destrutivo da recombinação e mutação.

1.8 Engorda

Uma das observações feitas sobre os operadores de recombinação e mutação é que os descendentes gerados podem aumentar o tamanho da árvore representativa da solução. Este efeito de crescimento das soluções durante da execução de um algoritmo em PG é muito comum e é conhecido como engorda (*bloat*). Há muitos estudos para o entendimento do processo e sobre medidas para contornar o problema.

Uma das correntes de pesquisa relaciona o crescimento excessivo dos indivíduos com o fenômeno de *overfitting* de uma RNA. Ou seja, as soluções do PG estariam se acostumando excessivamente com os pares de entrada-saída do problema que são utilizados na função de avaliação [6].

A despeito do motivo pelo o qual o fenômeno ocorre, é preciso adotar medidas para evitar a engorda excessiva. A forma mais simples de evitar engorda é limitar o tamanho máximo das árvores geradas. Usando essa estratégia, caso um operador gere uma árvore com tamanho maior do que o permitido, a operação seria descartada. O tamanho máximo seria, portanto, um parâmetro adicional para os operadores de mutação e recombinação. Diversas outras estratégias sofisticadas e complexas foram propostas, mas uma delas parece ter alcançado maior popularidade, denominada *parsimony pressure*. Usando essa metodologia, devemos acrescentar um termo na fórmula da função de aptidão que diminua o valor de aptidão dos grandes indivíduos ou usar algum algoritmo de otimização multi-objetivo.

1.9 Aplicações

1.9.1 Onde Usar

Generalizando, pode-se afirmar que PG pode ser aplicada a qualquer problema que seja possível expressar por um código-fonte de alguma linguagem de programação, se for considerado que o objetivo da PG é produzir programas automaticamente. Historicamente, os mais céticos tem citado problemas relacionados a programação automática que desencorajariam o estudo da área. É notável a diferença de complexidade entre áreas como otimização de problemas combinatoriais através de CE e programação automática através de PG.

À primeira vista, um dos grandes problemas seria o tratamento de restrições, ou seja, geração de programas inválidos. Esse tipo de desafio, contudo é comum em vários problemas já resolvidos através de computação evolucionária. Neste sentido, as formas de contornar o tratamento de restrições são transferidas para PG com as soluções já adotadas em outros ramos da CE.

O maior desafio está relacionado ao espaço de busca de um algoritmo da PG. Um problema com espaço de busca na ordem de $10^{100.000}$ obviamente é mais complexo do que um problema com espaço de busca na ordem de 10^{100}. Então por muitos anos os pesquisadores acreditavam ser infrutífero investir na área de geração automática de programas devido ao tamanho do espaço de buscas [8].

Conforme já mencionado na seção 1.2, mesmo considerando os desafios inerentes à esta técnica, já foram obtidos resultados palpáveis em várias áreas do conhecimento, compatíveis com invenções humanas patenteadas. A título de ilustração, vejamos alguns destes progressos da PG:

- circuitos de filtro que infringem patentes emitidas para George Campbell, Otto Zobel, e Wilhelm Cauer entre 1917 e 1935;
- 12 amplificadores e outros circuitos que infringem patentes de Sidney Darlington de 1953;
- uma rede de ordenação que é melhor do que a de O'Connor e Nelson (patente de classificação de redes de 1962);

- um circuito de filtro cruzado que infringe a patente de Otto Zobel de 1925;
- um algoritmo para a computação quântica do problema de "promessa precoce"de Deutsch-Jozsa, o problema de busca em banco de dados de Grover, e o problema de consulta com profundidade-2 que são melhores do que os algoritmos anteriormente publicados;
- uma regra de autômatos celulares para o problema de classificação da maioria que é melhor do que a regra de GaCS-Kurdyumov-Levin de que todas as outras regras conhecidas, escritas por humanos;
- um programa para jogar futebol, que foi classificado dentre outros 34 programas escritos por humanos na competição *Robo Cup* de 1998;
- um algoritmo para identificação de segmentos de proteínas que é melhor do que os algoritmos anteriormente publicados;
- algoritmos para detecção de famílias de proteínas que são iguais ou melhores do que os algoritmos anteriormente conhecidos elaborados por humanos; e
- Circuitos amplificadores, referência de tensão, Nand, conversores digital-analógico e analógico-digital que duplicam as funcionalidades previamente patenteadas.

De forma resumida, pode-se dizer que as aplicações reais de PG variam desde a **geração de pequenas sub-rotinas**, até **problemas em tempo real** (sistemas de controle para robótica), passando por aplicações para **análise de séries temporais** e **mineração de dados**. Todas as **aplicações de aproximação de funções** através de Redes Neurais Artificiais (RNA) são candidatas óbvias para serem resolvidas através de PG. Também pode-se incluir aqui **sistemas baseados em regras**, como provas automáticas de teoremas.

1.9.2 Estudo de Caso: Regressão Simbólica

Na seção 1.9.1 foi apresentado uma gama de problemas que podem ser solucionados usando PG. Para uma exposição prática da técnica, será apresentado aqui um problema simples de entender, de modo a alcançar os objetivos didáticos relacionados ao aprendizado de uma nova técnica de CE, sem preocupação com às idiossincrasias de um problema complexo.

Ajuste de curvas é um problema que já foi atacado por diversas técnicas de inteligência computacional, sendo a mais popular através do uso de RNA. O grande atrativo em utilizar PG para regressão simbólica é o fato de obtermos o texto da função que aproxima a curva, sendo uma vantagem expressiva sobre as abordagens de RNA, que aproximam as funções usando uma abordagem de caixa preta.

Definição do Problema

O interesse é encontrar uma função que represente os pares de entrada-saída de uma dada função. Considere a função de exemplo da Eq. 1.4.

$$3x^4 - 3x + 1 \tag{1.4}$$

A partir desta função serão gerados um conjunto de pontos. Para obter representatividade da função considere 20 pontos. Estes pontos serão usados na função de avaliação do algoritmo. Sobre os conjuntos de terminais e funções do nosso problema, são definidos os conjuntos especificados na Tabela 1.4.

Tabela 1.4: Conjunto de terminais e de funções para definição do problema de regressão simbólica.

Conjunto de funções (símbolos não-terminais)	$F = \{+, -, .\}$
Conjunto de símbolos terminais	$T = \mathbb{R} \cup \{x\}$

Perceba que poderiam ter sido definidas outras funções no nosso conjunto, tais como {/, sen, cos, exp, log}, mas optou-se por usar um conjunto mínimo de funções. Em uma aplicação de PG quanto mais simples a definição do problema, mais eficiente será a busca pela expressão ou pelo programa. Com base nos pontos gerados e nos conjuntos T e F, o problema de regressão simbólica está definido e os experimentos podem ser realizados.

Experimentos e Resultados

Usando os valores da Tabela 1.5, é obtida, após 747 gerações, a função em azul desenhada no gráfico da Figura 1.7. Nesta mesma figura, a curva em vermelho representa a função original da equação 1.4.

Tabela 1.5: Parâmetros usados na simulação de regressão simbólica.

Número de gerações	1.000
Tamanho da população	100
Profundidade máxima das árvores	6
Probabilidade Recombinação	80%

1.9.3 Perspectivas Futuras

O rápido decrescimento do custo dos recursos computacionais e a receptividade do uso de paralelismo na PG favorecem o desenvolvimento da área. À medida que a capacidade de computação aumenta e que mais programas elaborados paralelamente e com eficiência surgem, espera-se que PG consiga resolver problemas progressivamente mais complexos. A perspectiva é que os pesquisadores utilizem cada vez mais PG para produzir novas invenções ou melhorar soluções existentes.

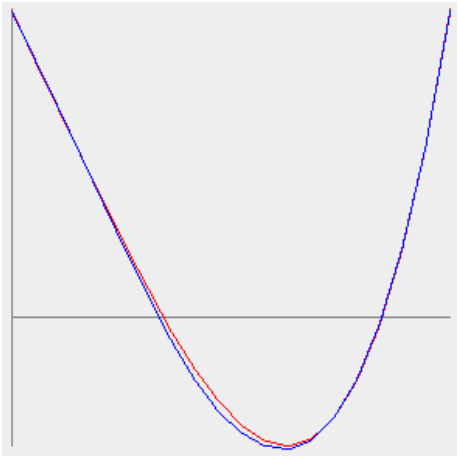

Figura 1.7: Curva real x curva obtida por PG.

Referências Bibliográficas

[1] J.E. Smith Agoston E. Eiben. *Introduction to Evolutionary Computation.* Springer, 2003.

[2] N. L. Cramer. A representation for the adaptive generation of simple sequential programs. pages 183–187, 1985.

[3] A.M.S. Zalzala; D. Green. Mtgp: a multithreaded java tool for genetic programming applications. *Congress on Evolutionary Computation, 1999. CEC 99.*, 1999.

[4] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* The MIT Press, 1992.

[5] John R. Koza. www.genetic-programming.org, 2007.

[6] Ricardo Linden. *Algoritmos Genéticos*, volume 2. 2008.

[7] S. F. Smith. A learning system based on genetic adaptive algorithms. page 214, 1980.

[8] C. Ryan L. Spector C. Jacob. W. Banzhaf., J. R. Koza. Genetic programming. *Intelligent Systems and their Applications, IEEE*, 15:74–84, 2000.