UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
ESCOLA DE INFORMÁTICA APLICADA

PROGRAMAÇÃO GENÉTICA APLICADA A ESTIMATIVAS DE PROJETO DE SOFTWARE

Arthur de Oliveira Lopes

Orientador:

Márcio de Oliveira Barros

RIO DE JANEIRO, RJ - BRASIL
ABRIL DE 2021

Catalogação informatizada pelo(a) autor(a)

de Oliveira Lopes, Arthur
d864 Programação Genética Aplicada a Estimativas de Projeto de Software / Arthur de Oliveira Lopes. -- Rio de Janeiro, 2021.
49 f

Orientador: Márcio de Oliveira Barros.
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal do Estado do Rio de Janeiro, Graduação em Sistemas de Informação, 2021.

1. Programação Genética. 2. Estimativas em projetos de software. 3. Algoritmos genéticos. 4. Magnitude média do erro relativo. I. de Oliveira Barros, Márcio, orient. II. Título.

PROGRAMAÇÃO GENÉTICA APLICADA A ESTIMATIVAS DE PROJETO DE SOFTWARE

Arthur de Oliveira Lopes

Projeto de Graduação apresentado à Escola de Informática Aplicada da Universidade Federal do Estado do Rio de Janeiro (UNIRIO) para obtenção do título de Bacharel em Sistemas de Informação.

Aprovado por:

_____

Márcio de Oliveira Barros (UNIRIO)

_____

Geiza Maria Hamazaki da Silva (UNIRIO)

_____

Morganna Carmem Diniz (UNIRIO)

RIO DE JANEIRO, RJ – BRASIL.

ABRIL DE 2021

Agradecimentos

Agradeço ao meu orientador Márcio Barros pela paciência e pelo bom direcionamento ao longo de todo o processo de desenvolvimento deste trabalho. Também agradeço a todos as outras pessoas que me ajudaram de forma direta ou indireta neste processo, através de sugestões, conversas e aconselhamentos relacionados ao projeto e ao meio acadêmico.

RESUMO

O trabalho publicado por J.J. Dolado serviu de inspiração para a implementação de um algoritmo de Programação Genética que resolve problemas de estimativas de esforço em projetos de desenvolvimento de software. As funções de esforço geradas pelo algoritmo proposto, que são expressões matemáticas baseadas em uma variável, relacionam medidas de tamanho dos projetos de software e o esforço necessário para o seu desenvolvimento. O algoritmo proposto nesse trabalho foi aplicado para gerar resultados para os mesmos conjuntos de dados utilizados no trabalho de Dolado e os resultados dos dois modelos foram comparados. Foi observado um ganho da ordem de 27,5% na métrica de comparação entre os modelos, a magnitude média do erro relativo das funções de esforço.

Palavras-chave: Programação Genética, estimativas de esforço, projetos de desenvolvimento de software, magnitude média do erro relativo.

ABSTRACT

The paper published by J.J. Dolado was an inspiration for the implementation of a Genetic Programming algorithm that solves problems addressing effort estimates for software development projects. The effort functions generated by the proposed algorithm, which are mathematical expressions based on a single variable, relate measures of software project size and the effort required for their development. The algorithm proposed in the current work generated results for the same data sets used in Dolado's paper and the results of the two models were compared. An expressive gain was observed in the metric used to compare the models, the mean magnitude of the relative error of the effort functions.

Keywords: Genetic Programmimg, effort estimation, software development projects, mean magnitude of the relative error.

Conteúdo

Lista de Figuras

Lista de Tabelas

1. Introdução

1.1 Motivação

A Programação Genética, uma técnica de geração de algoritmos, é a área central de estudo deste trabalho. Esta técnica faz parte do universo do aprendizado de máquina e nela os indivíduos de uma população evoluem ao longo várias gerações pela interação com dados de treinamento e através da ação de operadores genéticos. Estes indivíduos são armazenados em estruturas de dados e podem estar sob a forma de programas de computador, expressões matemáticas, entre outras representações. É comum que a Programação Genética seja utilizada para solucionar problemas de otimização, onde deseja-se minimizar ou maximizar uma ou mais funções objetivo, explorando um grande espaço de possíveis soluções.

O outro contexto teórico que permeia este trabalho é o das estimativas de esforço em projetos de desenvolvimento de software. Estimar o esforço de desenvolvimento para um projeto de software, a partir de dados de projetos já executados, é um problema de otimização e a Programação Genética é uma possível abordagem a ser utilizada no seu enfrentamento. J.J. Dolado [Dolado, 2001] propõe um modelo de Programação Genética para resolver problemas de estimativa de esforço, comparando seus resultados aos resultados obtidos por diversos autores em diferentes artigos anteriores. As soluções geradas pelo autor para este problema são expressões matemáticas que representam funções de esforço responsáveis por relacionar tamanho e esforço no desenvolvimento de projetos de software.

O trabalho de Dolado foi inspiração para a realização do experimento conduzido neste trabalho. O experimento aqui proposto também foi o desenvolvimento de um modelo de Programação Genética capaz de gerar funções de esforço para projetos de software. A motivação é, portanto, a comparação dos resultados dos

dois modelos e a possibilidade de melhora nos resultados conhecidos e divulgados na literatura de Engenharia de Software.

1.2 Objetivos

O primeiro objetivo do trabalho é propor um algoritmo de Programação Genética capaz de gerar funções de esforço, minimizando a métrica de aptidão escolhida, que neste caso é a métrica de MMRE (em português, Magnitude Média do Erro Relativo). O algoritmo deve gerar resultados considerando a existência de uma variável independente x e uma variável dependente y, tal que o resultado é uma equação $y = f(x)$.

O conjunto de dados utilizado como entrada é formado por pares de valores x e y. No contexto de aplicação do algoritmo, estes valores representam medidas de tamanho dos projetos de software e o esforço observado no seu desenvolvimento. Nenhuma outra informação sobre os dados de entrada ou características dos resultados desejados é conhecida pelo algoritmo. A função de esforço gerada ao final do ciclo de Programação Genética deverá ser aquela com a melhor métrica de aptidão, ou seja, o menor valor de MMRE.

O segundo objetivo é a comparação dos resultados obtidos pelo algoritmo proposto com os resultados publicados por J.J. Dolado. O algoritmo proposto gerou resultados para os mesmos doze conjuntos de dados (*datasets*) utilizados na pesquisa de Dolado [Dolado, 2001], que fazem parte de pesquisas de diversos autores que propuseram ou testaram modelos de estimativa de esforço para o desenvolvimento de projetos de software baseados em regressão linear.

O algoritmo de Dolado utilizou o erro quadrático médio como métrica de aptidão. As métricas de MMRE e PRED(0.25), apresentadas no Capítulo 2 deste documento, foram utilizadas na avaliação da capacidade preditiva do seu modelo. O objeto de comparação dos dois algoritmos foi a MMRE.

Da comparação entre os resultados dos dois algoritmos pôde-se observar um ganho expressivo da MMRE na maioria dos cômputos gerados pelo algoritmo proposto. O ganho médio, considerando os resultados dos doze conjuntos de dados, foi de 27,5%.

1.3 Organização do texto

No Capítulo 2 está a fundamentação teórica do trabalho. Os conceitos gerais sobre algoritmos genéticos e as várias etapas e elementos da execução da Programação Genética são detalhados e ilustrados. Depois, os conceitos do domínio de estimativas de esforço em projetos de software também são descritos. A interseção entre as duas áreas de estudo é feita ao final do capítulo, apresentando dois trabalhos que abordam o tema e os resultados e conclusões obtidos por eles.

O Capítulo 3 é responsável pelo detalhamento dos elementos e do funcionamento do algoritmo proposto, abordando os parâmetros de controle característicos da Programação Genética, os conjuntos de terminais e de funções, os operadores genéticos utilizados, a estratégia de geração da população inicial e a avaliação da aptidão dos indivíduos componentes da população.

No Capítulo 4 são apresentados os resultados obtidos pela execução do algoritmo proposto sobre os doze conjuntos de dados apresentados por Dolado. Os resultados são comparados aos resultados obtidos por Dolado, ilustrando as diferenças entre os resultados dos dois algoritmos através de gráficos e tabelas.

No capítulo 5 a conclusão do trabalho é elaborada, expondo contribuições atingidas e limitações existentes no algoritmo proposto. O capítulo é concluído com sugestões de trabalhos futuros no sentido de continuação e/ou aprimoramento deste trabalho.

2. Programação Genética

2.1 Introdução

A técnica de Programação Genética foi utilizada neste projeto para possibilitar a formação de funções que relacionem com alta precisão um conjunto de pares de valores que aqui foram denominados por x e y, ou seja, tentar encontrar a lei de formação (função de x) de um *dataset* de entradas (x, y) a partir de seu domínio e imagem. As funções de x, resultados da execução do sistema de Programação Genética, foram representadas em árvores binárias.

As árvores são comparadas e ordenadas através de uma métrica de aptidão e organizadas em populações, na qual são indivíduos. A ação dos operadores genéticos sobre os indivíduos, somada à capacidade de avaliá-los e compará-los, permite a conciliação entre garantir a permanência dos melhores (indivíduos melhor avaliados) e a diversidade ao longo de diversas gerações da população. Esse processo ocorre com elitismo, hereditariedade e aleatoriedade, elementos característicos da Programação Genética.

Este capítulo aborda conceitos sobre os algoritmos genéticos e a Programação Genética, mostrando como aspectos da genética tradicional e da evolução estão presentes nestes modelos e como é possível, com o mínimo de informações iniciais e regras estabelecidas, fazer com que o computador encontre automaticamente uma boa solução para problemas propostos.

2.2 Algoritmos genéticos

O algoritmo genético é o tipo mais comum de algoritmo evolutivo e é o predecessor da Programação Genética. Seu criador, John Holland, se inspirou na

natureza para criar um modelo de computação que se aproxima da evolução natural como a conhecemos [Langdon and Qureshi, 1995]. O algoritmo genético de Holland consiste em um método para avançar de uma população de indivíduos para uma nova população, utilizando uma combinação de seleção e os operadores genéticos de *crossover*, mutação e inversão.

Segundo Banzhaf [Banzhaf et al., 1998], o algoritmo genético original tem duas características principais: indivíduos em uma representação binária de tamanho fixo e a presença muito forte do operador genético *crossover* (ou cruzamento). Os indivíduos do algoritmo de Holland são chamados de cromossomos e sua representação pode ser vista na Figura 2.1. Segundo Mitchell [Mitchell, 1998], o termo cromossomo tipicamente se refere a uma solução candidata para um problema, geralmente codificada como uma sequência de bits. Cada cromossomo é formado por genes (cadeias de bits), que são formados por alelos (instâncias de 0 ou 1).

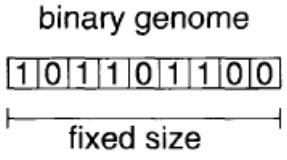

Figura 2.1: Representação binária de um indivíduo da população.
Fonte:[Banzhaf et al., 1998]

O operador de seleção escolhe os cromossomos cuja reprodução será permitida. De maneira geral, os cromossomos mais aptos, que são aqueles que representam uma melhor solução para o problema alvo, terão mais chances para a produção de descendentes do que os menos aptos. O *crossover* é o operador que realiza a troca de partes de dois cromossomos, em uma cópia da recombinação biológica entre dois organismos de conjunto único de cromossomos (haplóides). A mutação altera aleatoriamente os valores de alelos em alguns locais nos cromossomos e a inversão reverte a ordem de uma seção contígua do cromossomo, modificando portanto a ordem com que os genes estão dispostos.

O modelo introduzido por Holland, seus colegas de trabalho e seus alunos foi uma grande inovação e serviu como base para quase todos os trabalhos teóricos subsequentes sobre algoritmos genéticos. Segundo Mitchell [Mitchell, 1998], a expansão da interação entre pesquisadores estudando diversos métodos de computação evolutiva quebrou as barreiras existentes entre algoritmos genéticos, estratégias de evolução, programação evolutiva e outras abordagens evolutivas.

O termo "Algoritmo Genético" é usado hoje para descrever variações muito distantes do conceito original de Holland. Segundo Whitley [Whitley, 1994], um algoritmo genético é qualquer modelo baseado em população que usa os operadores de seleção e *crossover* para gerar novos pontos de amostra em um espaço de busca. No entanto, vários autores indicam também a presença dos operadores de reprodução e mutação até nos algoritmos genéticos mais simples.

A forma mais comum de *crossover* presente nos algoritmos genéticos é o *one-point crossover*. Ele acontece entre dois indivíduos pai de mesmo tamanho, como ilustra a Figura 2.2. Um ponto de cisão é escolhido aleatoriamente e alinhado nos dois pais. O material genético selecionado corresponde a todos os bits à direita do ponto de cisão e a troca é realizada, gerando dois descendentes.

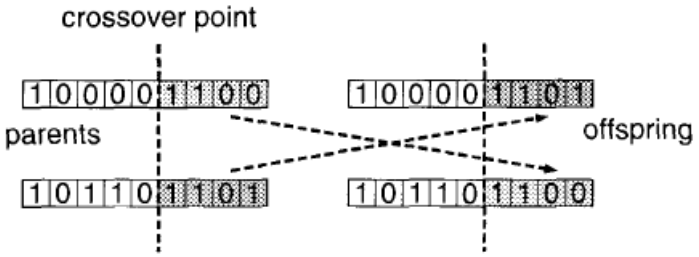

Figura 2.2: *One-point crossover* dos algoritmos genéticos

Foi provado de forma teórica e empírica que os algoritmos genéticos proveem busca robusta em espaços complexos [Goldberg, 1989]. Muitos artigos e Dissertações estabelecem a validade da técnica na otimização de funções e em aplicações de controle. Goldberg [Goldberg, 1989] descreve uma série de estudos e experimentos que envolveram a aplicação de algoritmos genéticos. Um deles é o uso de um algoritmo genético para determinar parâmetros de controle de um algoritmo reconhecedor de padrões em imagens. Apesar dos algoritmos serem computacionalmente simples, eles são poderosos em sua busca por soluções, justificando o crescimento do número de aplicações da técnica nos segmentos dos negócios, científico e da engenharia.

Os indivíduos são organizados em populações de tamanho fixo, nos algoritmos genéticos, e é muito comum que nesses algoritmos exista uma forma de calcular a aptidão de cada indivíduo, ou seja, uma função objetivo que permite compará-los entre si. Esta função objetivo é comumente chamada de função de aptidão. A métrica de aptidão é a forma com a qual é possível avaliar a qualidade dos indivíduos, ou seja, o quão distantes eles estão do resultado desejado. Segundo Langdon et al. [Langdon et al., 2008], dependendo do problema a ser solucionado,

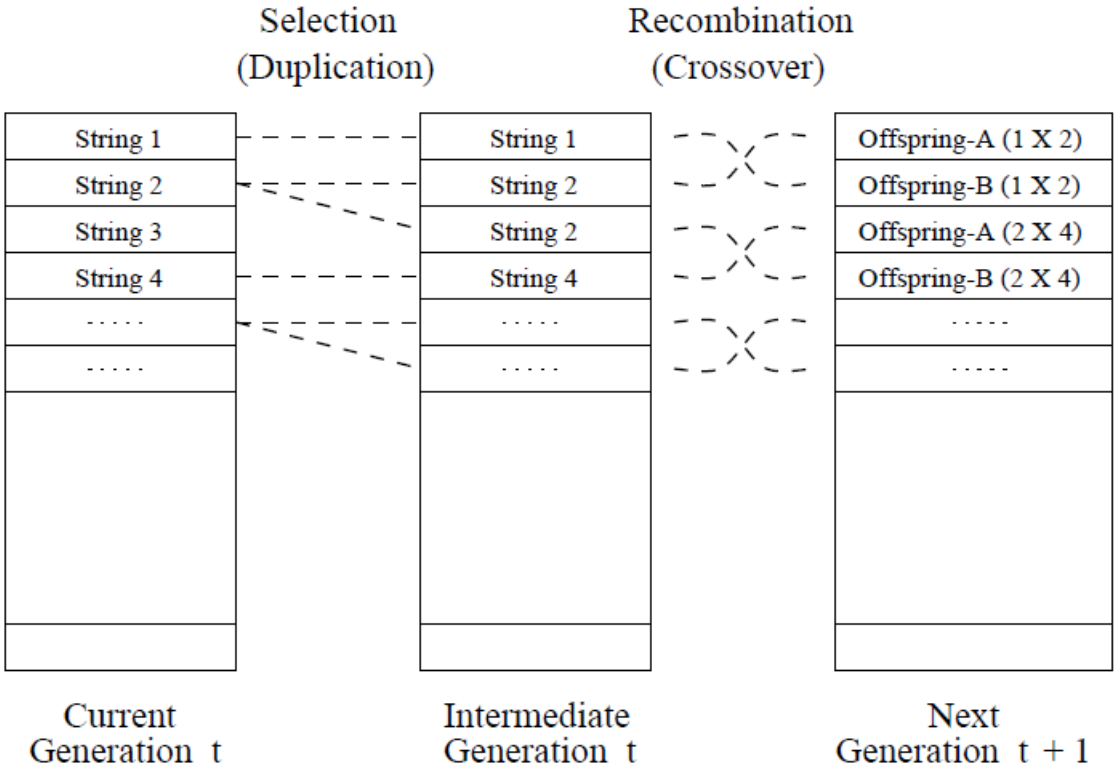

Figura 2.3: Execução de um algoritmo genético, em dois estágios, gerando descendentes somente com o *crossover*. Fonte: [Whitley, 1994]

a aptidão de um indivíduo pode ser medida de diversas formas, entre elas o erro relativo e a quantidade de tempo (ou combustível, dinheiro e similares) necessário para que um sistema chegue a um determinado estado.

É importante que a função de aptidão de um algoritmo genético seja graduada, e não binária. Segundo Banzhaf [Banzhaf et al., 1998], se houvesse apenas uma função binária, não haveria diferença suficiente entre indivíduos na população para a realização de um processo de busca evolutiva, dado o caráter da classificação binária. O aspecto diferencial de uma função de aptidão graduada se deve à sua capacidade de distinguir uma solução boa de uma solução ainda melhor.

A população evolui ao longo de várias gerações e os responsáveis por esta evolução são os operadores genéticos. Os indivíduos de uma nova geração são originados da ação dos operadores sobre a geração passada. O processo evolutivo existente na natureza é contínuo. No entanto, no contexto dos algoritmos genéticos é preciso que exista um objetivo a ser alcançado e que estabelece o ponto em que o algoritmo será encerrado. Esse ponto de parada da execução do algoritmo pode ser a geração do indivíduo com as características desejadas ou o melhor

resultado possível dentro de uma janela de gerações. A geração de um indivíduo ideal pode levar muitas gerações para ocorrer.

Ao longo dos anos, a utilidade da representação binária para problemas de otimização foi frequentemente questionada. Como consequência, diversas variações de algoritmo genético que não usavam representação binária surgiram. A representação em árvores é uma das mais famosas e mais utilizadas, em especial no contexto da Programação Genética.

2.3 Programação genética

Na Programação Genética, segundo Langdon [Langdon et al., 2008], a representação dos indivíduos da população geralmente é feita através de árvores sintáticas, nas quais os nós internos abrigam operações e os nós folha abrigam variáveis e constantes. A literatura de Programação Genética chama os nós internos de funções e os nós folha de terminais. Os indivíduos mencionados podem ser *clusters*, partições, expressões matemáticas, programas, entre outros.

O conjunto de terminais pode conter variáveis, como x ou y, funções sem argumentos e constantes. É possível que o conjunto de terminais contenha, por exemplo, todo o conjunto dos números reais. O conjunto de funções varia de acordo com a natureza do problema. Se os indivíduos forem expressões matemáticas, os elementos do conjunto serão operadores, como +, -, /, *, ln, etc. Se o contexto for o desenvolvimento de software, os elementos serão funções de programas de computador, como *openfile*, *save*, *allocate*, *process.kill*, entre outras. Os conjuntos de funções e de terminais permitidos no modelo formam o conjunto primitivo de um sistema de Programação Genética. Isso significa que nenhuma árvore conterá elementos não presentes no conjunto primitivo, independente da mudança que sofra ou do procedimento que a originou.

Além das árvores sintáticas, outros tipos de representação menos comuns podem ser utilizados na Programação Genética, como a representação linear, representação em grafos, representação em termos *booleanos*, entre outros. O uso de cada representação pode estar relacionado a uma série de fatores, como eficiência, conveniência e facilidade de implementação. Neste trabalho, o foco estará na representação em árvores sintáticas. Em Banzhaf [Banzhaf et al., 1998] e Langdon [Langdon et al., 2008], outros tipos de representação são expostos e detalhados.

Os principais operadores utilizados na Programação Genética são: *crossover*, mutação, seleção e reprodução. O *crossover*, descrito na seção 2.2, envolve a geração de um ou dois indivíduos filhos a partir de partes de dois indivíduos pais selecionados na população. A mutação gera um novo indivíduo a partir de uma modificação em um indivíduo da população. O operador de reprodução faz a cópia de parte da geração atual para a próxima. Em Langdon [Langdon and Qureshi, 1995] alguns operadores não tradicionais e incomuns são introduzidos.

A seleção é o operador que elege os candidatos que sofrerão a ação dos outros operadores, tendo grande importância nas mudanças entre gerações. Seguindo o modelo da seleção natural, a seleção da Programação Genética geralmente trabalha com o princípio de sobrevivência do mais apto, pois a métrica de aptidão é a base para a escolha dos indivíduos. Existem diversas formas de realizar seleção em uma geração. Algumas delas serão mencionadas em seções posteriores.

Após a seleção, a frequência do uso dos operadores é determinada pelas probabilidades de ocorrência definidas para cada um deles. Essas probabilidades são importantes parâmetros da execução de um sistema de Programação Genética. Tamanho da população, quantidade de gerações e profundidade máxima das árvores da primeira geração também são parâmetros que podem influenciar na qualidade dos resultados obtidos em uma execução.

A aleatoriedade, outro elemento fundamental de uma execução, está diretamente ligada ao sucesso e eficiência da Programação Genética. Banzhaf [Banzhaf et al., 1998] afirma, baseado em Eigen(1992) e Koza [Koza, 1992], que quando mudanças aleatórias são combinadas com seleção baseada em aptidão, um sistema computacional é geralmente capaz de evoluir soluções mais rápido do que a busca aleatória. A aleatoriedade está presente na geração da população inicial e na ação dos operadores genéticos de *crossover* e de mutação.

2.3.1 População inicial em Programação Genética

A primeira geração da população manipulada por um algoritmo genético é a única que não é gerada através da ação dos operadores genéticos. A população desta geração é formada por árvores de variadas profundidades e organizações de nós e pode ser gerada de forma aleatória ou parcialmente aleatória. Os métodos e parâmetros de métodos utilizados para a construção aleatória das árvores da primeira geração podem ser decisivos para que um algoritmo genético chegue

ao resultado desejado. Na geração parcialmente aleatória, características dos resultados desejados já são conhecidas e julga-se necessário sua introdução em parte da população inicial. Esta maneira de gerar a população inicial está fora do escopo deste trabalho. Na geração aleatória da população inicial, que será o tipo de geração abordado, nenhuma característica desejada é forçada sobre os indivíduos.

Na Programação Genética, os dois métodos mais comuns para a inicialização aleatória de árvores são *Full* e *Grow*. Nestes dois métodos, as árvores geradas estão sujeitas a um valor de profundidade máxima, que serve de parâmetro para o método. Isso significa que nenhuma árvore da população inicial terá profundidade superior a este valor estabelecido.

No método *Full*, somente árvores completas são geradas. Estas são árvores cujas folhas estão todas no último nível de profundidade da árvore, que é igual ao valor máximo de profundidade estabelecido. Nós internos são preenchidos com elementos aleatórios do conjunto de funções até que a árvore alcance a profundidade máxima. A partir daí, os nós folha são preenchidos com elementos aleatórios do conjunto de terminais.

No método *Grow*, a profundidade e o formato das árvores variam de acordo com o tamanho dos conjuntos de funções e de terminais. Isso acontece porque no preenchimento de cada nó, a partir da raiz, um elemento é sorteado da união dos dois conjuntos. Cada novo nó será um elemento aleatório desse conjunto união e, naturalmente, se um terminal for escolhido, o nó será uma folha. Esse aspecto difere este método do método *Full*, pois apesar dos dois respeitarem o parâmetro de profundidade máxima, o método *Grow* é capaz de gerar árvores com profundidade inferior ou igual à máxima e com nós folhas distintos entre si quanto à profundidade.

É importante observar que a relação entre o tamanho dos conjuntos de funções e de terminais afeta a profundidade média das árvores geradas com o método *Grow*. Se o conjunto de funções for bem maior que o conjunto de terminais, a profundidade média das árvores geradas será muito maior do que aquela calculada em uma situação inversa (conjunto de terminais muito maior do que o de funções), pois a probabilidade do sorteio de uma função/operador seria consideravelmente maior do que a de um terminal.

Uma combinação dos dois métodos foi proposta por Koza [Koza, 1992] e foi chamada de *ramped half-and-half*. O método propõe que metade da população seja construída sob o método *Full* e a outra metade sob o método *Grow*. O objetivo é garantir na população inicial uma variedade de formatos e tamanhos que o uso isolado de um dos dois métodos não traria.

2.3.2 Operador de *crossover* na Programação Genética

Segundo Banzhaf [Banzhaf et al., 1998], a Programação Genética é baseada em um modelo do que entendemos ser os mecanismos de aprendizado evolutivo orgânico. Esse modelo inclui a mutação, a seleção natural e a reprodução sexual. É através dessa reprodução que os genes são passados adiante para as próximas gerações. O *crossover* é o responsável, nos modelos dos algoritmos genéticos e da Programação Genética, pela combinação genética de dois indivíduos, reproduzindo os conceitos de hereditariedade e cruzamento cromossômico da biologia. O *crossover* é o operador genético mais frequente na grande maioria dos sistemas de Programação Genética propostos.

Na Programação Genética existem diversos tipos de *crossover*. O mais comum, e aquele que será detalhado neste trabalho, é o chamado *subtree crossover* (ou cruzamento de subárvores). Duas árvores da população são selecionadas para sofrer o *crossover*, sendo os pais neste processo. Um ponto de *crossover* é selecionado aleatoriamente em cada um dos pais e as subárvores geradas da cisão neste ponto são utilizadas para formar árvores filhas, que herdam o "código genético" sob a forma de nós (subárvores). A partir dos dois pontos de *crossover* nas duas árvores pai, diferentes combinações podem ser feitas, como pode ser visto na Figura 2.4.

Langdon [Langdon et al., 2008] afirma que a escolha dos pontos de *crossover* geralmente não é feita com probabilidade uniforme, a não ser em estudos técnicos sobre o comportamento da Programação Genética. Isso porque o fator de ramificação das árvores, que é o número de nós filhos que cada nó interno da árvore possui, é um obstáculo para a eficiência do *crossover* na população. Sendo este fator sempre igual ou maior que dois, a tendência é que as árvores possuam muito mais folhas do que nós internos, ou seja, mais terminais do que funções. Caso o ponto de *crossover* seja escolhido com probabilidade uniforme, a chance do sorteio de um nó folha será muito maior e, como consequência, a troca de material genético tende a ser menor. O problema que se origina dessa situação é o baixo potencial de alterações na população de uma geração para a outra, já

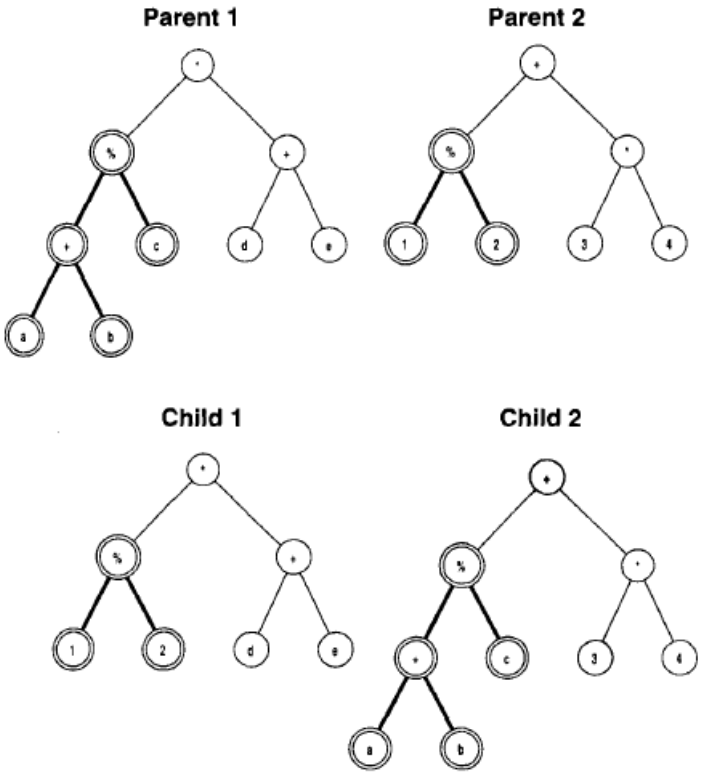

Figura 2.4: Geração de dois descendentes através do *subtree crossover*

que a alta frequência do *crossover* sobre os outros operadores também significa a concentração da responsabilidade pela geração de inovação. Para combater essa limitação, Koza propôs que funções (nós internos) fossem escolhidas como pontos de *crossover* em 90% do tempo e terminais (folhas) em 10% do tempo.

2.3.3 Operador de mutação

Mais simples do que o *crossover*, a mutação é um operador genético que age sobre um indivíduo. Geralmente, a probabilidade de ocorrência de uma mutação após a seleção é inferior à probabilidade de ocorrência de um *crossover*. A mutação pode ocorrer sobre os indivíduos originados do *crossover* ou como um evento isolado. Koza [Koza, 1992] afirma que a mutação desempenha a função de introduzir variedade ocasional e restaurar diversidade perdida em uma população. A ausência de diversidade é um problema que aumenta quanto maior for a distância entre o resultado desejado e a aptidão dos melhores indivíduos, pois nessa situação a troca de material genético do *crossover* tem um potencial muito reduzido de introdução de variedade. Portanto, a presença da mutação é importante em um sistema de Programação Genética para evitar a combinação

de uma população "engessada", ou seja, com baixa diversidade e capacidade de mudança, e métricas de aptidão distantes do resultado desejado.

Na mutação realizada em árvores, um ponto de mutação deve ser escolhido no indivíduo selecionado de forma aleatória, assim como no *crossover*. A subárvore com raiz neste ponto é então substituída por uma subárvore ou nó gerados aleatoriamente. Existem vários tipos de mutação aplicados na Programação Genética com representação em árvores. A mutação de ponto e a permutação estão entre os tipos de mutação mais simples e de menor potencial de introdução de variedade. A mutação de ponto altera a árvore alvo substituindo um único nó por outro da mesma classe. A permutação, por sua vez, faz a troca de posição dos nós filhos de um nó interno.

A mutação de subárvore é um tipo de mutação com alto potencial de introdução de variedade na população. Nela, a subárvore com raiz no ponto de mutação, que fora escolhido aleatoriamente, é substituída por uma subárvore aleatória, cuja geração geralmente respeita a mesma regra de limite de profundidade utilizada na geração da população inicial. É uma mutação com grande potencial de alteração de formato, profundidade e aptidão da árvore escolhida, podendo introduzir em uma população um código genético impossível ou pouco provável de ser alcançado somente com o *crossover*.

A escolha do tipo ou tipos de mutação a serem utilizados em um sistema de Programação Genética depende do poder que deseja-se dar ao operador – seu potencial de introdução de mudança aleatória. A mudança da mutação, seja cirúrgica ou radical, pode trazer a chave para a melhora da aptidão dos indivíduos ou pode ser insignificante, gerando indivíduos de baixíssima ou nenhuma aptidão, que não terão grande impacto na próxima seleção.

2.3.4 Funções de aptidão e seleção

O algoritmo de cálculo de uma função de aptidão possui comportamento recursivo, já que a leitura de uma árvore sintática é naturalmente recursiva. Existem diversos tipos de função de aptidão, atribuindo aos indivíduos diferentes padrões de métrica. Banzhaf [Banzhaf et al., 1998] descreve os tipos básicos de função de aptidão:

- **Aptidão Padronizada:** função em que a melhor aptidão sempre é o mesmo valor, zero;

– **Aptidão Normalizada:** função em que a aptidão está sempre entre zero e um;

– **Aptidão Contínua:** função em que a aptidão assume valores contínuos. O aumento da métrica corresponde, na mesma proporção, no quão bem um programa aprendeu sobre o domínio.

É possível, por exemplo, que uma função de aptidão seja ao mesmo tempo contínua e padronizada. Neste caso, quanto mais próximo o retorno for de zero, melhor a aptidão e, se for zero, obteve-se o resultado ideal.

Após a avaliação de um indivíduo ou população através da função de aptidão, é preciso decidir quais indivíduos sofrerão a ação de operadores genéticos e quais serão descartados. Estas são responsabilidades da técnica de seleção. Existem diversos métodos de seleção e a escolha de um método específico pode ser uma das decisões mais importantes em um sistema de Programação Genética.

A seleção é um algoritmo que, na maioria das vezes, utiliza a métrica de aptidão dos indivíduos como base para a escolha de candidatos que sofrerão a aplicação dos operadores genéticos. Quanto melhor for o valor da métrica, maior será a chance de um indivíduo ser selecionado. Apesar disso, é importante lembrar que um dos princípios da seleção nos algoritmos genéticos é que outros indivíduos além do melhor devem ter chance de reprodução, ou seja, de ser escolhidos. A seleção baseada em aptidão é uma das partes mais importantes do modelo de aprendizado evolutivo implementado pela Programação Genética [Banzhaf et al., 1998].

A seleção trabalha com uma população finita e de tamanho fixo. Apesar disso, a ação dos operadores genéticos injeta uma série de novos indivíduos na população. É preciso determinar, portanto, quais indivíduos farão parte da próxima geração e quais serão descartados. Se o sistema de Programação Genética fizer uso do operador de reprodução, descrito na Seção 2.2, parte dos indivíduos mais aptos serão copiados para a próxima geração e o restante necessário para completar o tamanho da população é preenchido com as árvores resultantes da ação do *crossover* e da mutação. Os demais indivíduos são descartados. Uma nova avaliação da métrica de aptidão dos indivíduos da população torna-se necessária a cada avanço de geração, por causa da inserção de novos indivíduos.

Os dois métodos mais comuns de seleção são a seleção de aptidão proporcional (*Fitness-Proportional Selection*) e a seleção por torneio (*Tournament selection*),

sendo o último o método mais aplicado, segundo Miller [Miller et al., 1995]. A seleção de aptidão proporcional envolve a especificação de uma probabilidade de poder realizar reprodução genética para os indivíduos. Ao indivíduo *i* é dada a probabilidade:

$$p_i = f_i / \sum_{j=1}^{N} f_j$$

, onde N é o tamanho da população. A probabilidade de seleção do i-ésimo indivíduo será a razão entre o resultado da função de aptidão f para i, que é a aptidão de i, e a soma da aptidão de todos os indivíduos da população.

A técnica de seleção de aptidão proporcional mais comum é chamada de seleção de roleta (*Roulette Wheel Selection*) [Dyer, 2008]. Nela, cada membro da população é alocado a uma seção de uma roleta imaginária. Diferente de uma roleta real, esta possui seções de diferentes tamanhos, que são proporcionais à aptidão de cada indivíduo. A roleta é "girada", evento representado por um sorteio aleatório, e o indivíduo associado à seção ganhadora é selecionado.

A seleção de aptidão proporcional foi a ferramenta de escolha da comunidade de algoritmos genéticos por um longo tempo. Um possível problema da técnica da roleta é a seleção de um ou mais indivíduos diversas vezes, graças a atribuições de altas probabilidades decorrentes da qualidade dos indivíduos.

Na seleção por torneio, a competição entre os indivíduos não é realizada no escopo de toda a população, e sim em um subconjunto dela. Um determinado número de indivíduos, chamado de tamanho do torneio e representado na literatura como *k*, é selecionado aleatoriamente da população, com ou sem reposição. A reposição significa que os indivíduos que não vencem o torneio não serão removidos da população; logo, ainda haverá a possibilidade de avançarem para a próxima geração. O melhor indivíduo do torneio é selecionado com probabilidade $P = p$. O segundo melhor é selecionado com probabilidade $P = p \times (1 - p)$, o terceiro com $P = p \times (1 - p)^2$, e assim sucessivamente.

Caso a seleção por torneio seja determinística, o valor de p será sempre 1 e o indivíduo de melhor aptidão da amostra recuperada será selecionado. Caso não seja, o valor de p, que varia entre 0 e 1, pode ser ajustado para que a seleção tenha mais ou menos pressão, que se traduz em um maior ou menor elitismo. Na prática, valores sempre superiores a 0.5 são escolhidos para favorecer candidatos

de melhor aptidão. Dado o valor de p, um sorteio aleatório de um valor entre 0 e 1 é feito. O intervalo do sorteio é dividido em k partes proporcionais aos valores de P e o vencedor do torneio será aquele cuja fatia abrigar o valor sorteado. Este tipo de torneio é chamado de probabilístico. Em ambos os tipos de torneio, o valor mínimo do tamanho k é 2.

Apesar da seleção por torneio dar a chance para que indivíduos fora do topo da hierarquia de aptidão sejam escolhidos, o processo ainda é bastante elitista dentro da amostra k utilizada. Por exemplo, se considerarmos um torneio de tamanho $k = 10$ e $p = 0.7$, qualquer sorteio (0 a 1) maior que zero e menor que 0.7 elegeria o indivíduo de melhor aptidão. O segundo indivíduo seria selecionado caso o valor sorteado fosse maior do que 0.7 e menor do que 0.91. A chance de um dos dois indivíduos de maior aptidão ser o vencedor do torneio é portanto de 91%, o que configura um elitismo forte.

Segundo Miller [Miller et al., 1995], os motivos da preferência pela seleção por torneio são a facilidade de implementação e a possibilidade de ajuste da pressão da seleção, permitindo a sintonia da técnica para diferentes domínios. A pressão da seleção é o grau de favorecimento dos melhores indivíduos, ou seja, o grau de elitismo na escolha dos vencedores do torneio. Quanto maior for a pressão, maior será a probabilidade de indivíduos mais aptos serem os vencedores. A pressão da seleção por torneio é aumentada com o simples aumento do tamanho do torneio (k) e os vencedores de um torneio maior terão aptidão média superior aos vencedores de um torneio menor [Miller et al., 1995].

A taxa de convergência de um algoritmo genético, que é a velocidade com que a solução é encontrada, é em grande parte determinada pela pressão da seleção [Miller et al., 1995]. Entretanto, se a pressão da seleção é muito alta, há uma chance maior do algoritmo convergir prematuramente para uma solução incorreta ou abaixo do ideal. Se a pressão for muito baixa, a taxa de convergência será muito lenta e o algoritmo levará desnecessariamente mais tempo para encontrar a solução ideal.

2.4 Estimativas em projetos de software

Estimativa de software é uma das áreas mais desafiadores da gerência de projetos. Por décadas, os profissionais da área enfrentam os desafios de estimar corretamente o esforço, custo e duração de processos do ciclo de vida dos projetos,

necessários para o estabelecimento de prazos e orçamentos. O desafio é prever estes parâmetros nos estágios iniciais do ciclo de vida, quando ainda existe bastante incerteza a respeito das funcionalidades do produto final [Pospieszny et al., 2017]. As estimativas de recursos em projetos de software não interessam somente ao gerente de projetos, mas também a outros participantes do processo de desenvolvimento, como engenheiros de software, clientes e gerentes de risco.

De acordo com a empresa Parametric Analysis [ISPA, 2008], problemas de estimativa de software geralmente ocorrem pelos seguintes motivos:

- Incapacidade de medir precisamente o tamanho de um projeto de software;
- Incapacidade de especificar precisamente um ambiente de desenvolvimento e suporte de software;
- Avaliação imprópria do tamanho e das habilidades da equipe;
- Falta de requisitos bem definidos para a atividade alvo da estimativa.

Muitas pessoas acreditam que é impossível de projetar precisamente o tamanho de uma aplicação, índices de produtividade ou o tempo necessário para desenvolver ou atualizar um produto. Se devidamente utilizadas, estimativas podem prover restrições que potencialmente limitam escolhas disponíveis no planejamento de um projeto. Elas também identificam as limitações de recursos que devem ser consideradas na montagem do cronograma do projeto, afetando também a escolha de métodos e ferramentas de desenvolvimento.

Segundo Abran [Abran, 2015], a técnica de desenvolvimento comumente utilizada na indústria, conhecida como *Opinião de Especialistas*, não costuma documentar quais parâmetros são levados em consideração ou como eles são combinados. Esse processo de estimativa se baseia no julgamento de um especialista e está sujeito à viés por parte do mesmo, podendo gerar estimativas muito otimistas graças a problemas como inexperiência dos envolvidos, desconhecimento da organização ou pressão por parte da gerência ou dos clientes. Não é viável avaliar o desempenho desse tipo de modelo, pois não há dados quantificados de forma objetiva e padronizados em variáveis de projeto, como o tamanho do software.

Em contraste a esta abordagem, há a estimativa obtida através de métodos algorítmicos, baseados em modelos matemáticos que produzem estimativas de custo e esforço. Esses modelos utilizam fatores geradores de custo do projeto

como parâmetros quantitativos ou categóricos de uma função. As entradas são manipuladas através de equações e o resultado do modelo é uma estimativa de esforço. Qualquer um pode fazer uso dos modelos e a variação na estimativa gerada depende da variação das entradas providas.

Abran [Abran, 2015] afirma que estes modelos são, na verdade, modelos de produtividade, pois são construídos utilizando dados de vários projetos concluídos, baseando-se portanto em um conjunto de fatos conhecidos sobre os quais não há incerteza. Entre estes fatos estão:

- Requisitos do produto: Classificações de software e intervalos esperados de medições de tamanho, baseados em padrões estabelecidos;
- Recursos organizacionais: experiência da equipe em um domínio de negócio, aptidão no desenvolvimento, disponibilidade na duração do projeto, entre outros;
- Aspectos técnicos do processo de desenvolvimento, como metodologia e ambientes de desenvolvimento, homologação e produção;
- Restrições de projeto.

Estes fatos conhecidos são divididos em dados quantitativos (por exemplo, o tamanho do software) e categóricos (como, por exemplo, a linguagem de programação) e serão entradas para a construção dos modelos matemáticos. A saída dos modelos consiste em uma equação e informações sobre a variância dos seus resultados. Entre as maiores vantagens desse tipo de modelo de estimativa está a possibilidade de avaliar o desempenho dos modelos, que descrevem suas variáveis seguindo convenções documentadas e podem ser avaliados contra os fatos observados em uma série de projetos.

Existem modelos matemáticos cujo principal objeto de estimativa é o tamanho do software a ser desenvolvido, como os modelos de Pontos de Função e Linhas de Código. Essa métrica pode ser utilizada como parâmetro de modelos de estimativa de custo e até mesmo como um recurso independente no processo de desenvolvimento de software de uma organização. Os procedimentos de um modelo de estimativa de tamanho de software e de um modelo de estimativa de custo são detalhados a seguir.

2.4.1 Pontos de função

Um dos métodos mais utilizados para estimar e medir o tamanho do software é a Análise de Pontos de Função (APF). O modelo foi introduzido por Allan Albrecht em 1979 e baseia sua medição na quantificação das funcionalidades que o software fornece para o usuário e em um conjunto de fatores individuais do projeto. Segundo o *International Function Point User Group* (IFPUG) [IFPUG, 2005], a APF tem como objetivos medir as funcionalidades que o usuário requisita e recebe e medir o desenvolvimento e manutenção de software, independentemente da tecnologia utilizada para implementação. A unidade de medida de pontos de função se consolidou como um padrão para medir software e é amplamente utilizada no mercado. O método de APF é atualizado pela *IFPUG*, uma organização sem fins lucrativos, desde 1986.

A aceitação do método também se deve ao seu uso como ferramenta de medida de produtividade. Quando existe um histórico de dados de produtividade e uma especificação de requisitos funcionais razoavelmente completa, o método pode ser usado para estimar o número de horas de trabalho de desenvolvedores necessário para implementar a especificação.

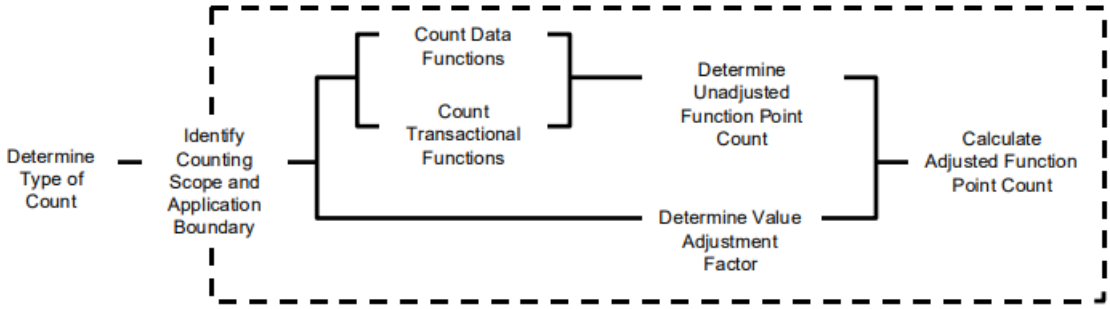

Figura 2.5: Processo de contagem de pontos de função

Como é possível ver na Figura 2.5, o processo de contagem de pontos de função consiste dos seguintes passos:

- Determinar o tipo de contagem de pontos de função;
- Identificar o escopo de contagem e os limites da aplicação;
- Contar as funções de dados para determinar suas contribuições para a contagem de pontos de função não ajustados;

- Contar as funções de transação para determinar suas contribuições para a contagem de pontos de função não ajustados;
- Determinar os fatores de ajuste de valor;
- Calcular a contagem de pontos de função ajustada.

O primeiro passo do procedimento de contagem de pontos de função é identificar o tipo de contagem, que varia de acordo com o tipo de projeto. A contagem pode ser de três tipos:

- Projeto de Desenvolvimento: mede as funções providas aos usuários com a primeira instalação do software entregue quando o projeto é completo;
- Projeto de Melhoria: mede as modificações da aplicação existente que adicionam, alteram ou removem funções oferecidas aos usuários;
- Aplicação: provê a medição das funções que estão sendo providas ao usuário pela aplicação. Esta contagem é iniciada quando a contagem de pontos de função de projeto de desenvolvimento é finalizada. Ela é atualizada cada vez que o término de um projeto de melhoria altera as funções da aplicação.

O próximo passo é identificar o escopo de contagem e os limites da aplicação. A identificação do escopo consiste em definir as partes do software que serão medidas e quais funções serão inclusas na contagem. Por exemplo, o escopo de uma contagem de pontos de função de projeto de melhoria inclui todas as funções que serão adicionadas, alteradas ou removidas. Os limites da aplicação indicam a fronteira entre o software que está sendo medido e o usuário. O limite definido serve para identificar o que é externo à aplicação e age como uma membrana através da qual dados processados por transações entram e saem da aplicação.

A Contagem de Pontos de Função Não Ajustados (PFNA) é determinada avaliando o conjunto de funções providas ao usuário pela aplicação, em termos *do que* é entregue e não *como* é entregue. Somente componentes definidos ou requisitados pelo usuário são contados. As funções contadas se dividem em dois tipos: funções de dados e funções transacionais. Segundo o IFPUG [IFPUG, 2005], esses dois tipos se subdividem em cinco tipos de funções:

Funções de dados:

– Arquivo Lógico Interno (ALI): grupo lógico de dados ou informações de controle reconhecíveis pelo usuário que são gerados, utilizados e mantidos dentro dos limites da aplicação;

– Arquivo de Interface Externa (AIE): grupo lógico de dados ou informações de controle reconhecíveis pelo usuário que são referenciados pela aplicação mas mantidos nos limites de outra aplicação. Isso significa que o AIE de uma aplicação será necessariamente o ALI de outra aplicação.

Funções transacionais:

– Entrada externa (EE): responsáveis por processar dados ou informações de controle que vêm de fora do limite da aplicação. O principal propósito das funções EE é manter um ou mais Arquivos Lógicos Internos e/ou alterar o comportamento do sistema;

– Saída Externa (SE): enviam dados ou informações de controle para fora do limite da aplicação. Apresentam informação ao usuário através de uma lógica de processamento que deve conter pelo menos uma fórmula matemática ou criar dados derivados. Uma SE também pode manter um ou mais Arquivos Lógicos Internos e/ou alterar o comportamento do sistema;

– Consulta Externa (CE): apresentam dados ao usuário pela recuperação de dados ou informações de controle dos Arquivos Lógicos Internos ou Arquivos de Interface Externa.

Cada um dos cinco tipos de função é classificado como 'Simples', 'Regular' ou 'Complexo', de acordo com sua complexidade funcional, que varia de acordo com a quantidade de dados manipulados, a organização destes dados em entidades fracas e o número de agrupamentos lógicos referenciados. Cada um recebe então um número de pontos, baseado no tipo e na complexidade definida. A soma desses pontos configura a Contagem de Pontos de Função Não Ajustados.

A estimativa de tamanho, ou seja, a quantidade de pontos de função (PF), é obtida multiplicando os Pontos de Função Não Ajustados pelo Ajuste de Complexidade Técnica (ACT). Na Figura 2.5,

$$PF = PFNA \times ACT$$

Para obter o ACT, é preciso classificar quatorze características gerais do sistema numa escala de 0 a 5. Essa classificação é o grau de influência da característica

na aplicação a ser estimada, onde zero significa nenhuma influência e 5 significa forte influência. Entre as características listadas pelo IFPUG [IFPUG, 2005] estão o processamento de dados distribuídos, a reusabilidade de código, a facilidade de instalação, a atualização de dados online e o atendimento a critérios rígidos de desempenho. É importante frisar que versões recentes do protocolo de contagem descartaram estas características e sugerem que o ACT seja igual a 1. Nas versões anteriores, o ACT é calculado pela fórmula abaixo:

$$ACT = (GIT \times 0.01) + 0.65$$

GIT é o grau de influência total, somatório do grau de influência das quatorze características. O GIT varia portanto de 0 a 70, e o ACT de 0.65 a 1.35. Percebe-se então que o Ajuste de Complexidade Técnica promove um aumento ou redução dos Pontos de Função Não Ajustados de até 35%. Exemplificando, se as quatorze características forem classificadas como grau 2, grau de influência moderada para a aplicação, o valor do GIT será 28 e o ACT será de 0.93. A contagem de pontos de função, também referenciada no modelo como Pontos de Função Ajustados, será, nesse caso, 93% do valor de PFNA previamente calculado.

Apesar do modelo matemático envolver cálculos simples, o processo de definição do escopo, identificação e classificação de funcionalidades que a aplicação deve ter e classificação das características de ajustes requerem um bom conhecimento do software a ser produzido ou melhorado.

2.4.2 COCOMO II

Um modelo de estimativa de custo de software amplamente conhecido é o COCOMO, sigla para *Constructive Cost Model*. O COCOMO II, versão revisada do COCOMO e objeto de estudo dessa subseção, foi desenvolvido na *University of Southern California* e é um modelo que calcula o tempo e o esforço necessários para o desenvolvimento de um software através de estimativas geradas em seus quatro submodelos. Estes modelos, por sua vez, utilizam diferentes métricas de tamanho e diversos parâmetros para gerar estimativas de diferentes complexidades. Os quatro sub-modelos do COCOMO II, ordenados de forma crescente quanto à complexidade das estimativas geradas, são:

1. Modelo de Composição de Aplicações: baseado na métrica de Pontos de Objeto, que conta as telas, relatórios e módulos de programação de alto nível desenvolvidos na aplicação e aplica a cada um deles um fator de com-

plexidade. Tem seu uso esperado nos estágios iniciais do planejamento de produtos baseados na composição de aplicações, geralmente desenvolvidas por uma equipe pequena;

2. Modelo de Design Inicial: utilizado quando os requisitos de sistema são conhecidos, mas o *design* ainda não começou. Neste modelo são utilizadas a métrica de Pontos de Função e um conjunto de sete parâmetros geradores de custo, que são aplicados sobre o resultado da métrica inicial. **??** lista quais são os sete parâmetros geradores de custo, originados da combinação dos parâmetros geradores de custo do modelo pós-arquitetônico;

3. Modelo de Reuso: estima o esforço para integrar componentes reutilizáveis utilizando um modelo não linear. Por se tratar de reengenharia e conversão de software, a métrica utilizada é a de número de linhas de código, considerando apenas o código que for reutilizado ou gerado. A estimativa também utiliza três parâmetros relacionados ao código: estrutura, autodescritividade e clareza da aplicação, que são classificados em uma escala de 5 níveis (de muito baixo a muito alto);

4. Modelo pós-arquitetônico: utilizado quando a arquitetura do sistema foi desenhada e se conhece uma quantidade maior de informações sobre o sistema. Um total de dezessete geradores de custo, listados em **??**, são classificados em uma escala de 6 níveis (de muito baixo a extra alto) para ajustar a estimativa de esforço do modelo (medido em meses de trabalho) para o desenvolvimento do produto. Os dezessete geradores de custo estão agrupados em quatro categorias: produto, plataforma, pessoal e projeto. A métrica de tamanho utilizada é a de número de linhas do código-fonte do produto.

A fórmula geral do COCOMO II está apresentada abaixo, na qual PM (*person-months*) é o esforço para desenvolver o projeto de software em meses de trabalho de uma pessoa, GC representa os dezessete geradores de custo do modelo, T o tamanho estimado do software em milhares de linhas de código-fonte e *W* representa os cinco fatores de escala do modelo [Musilek et al., 2002].

$$PM = 2.94 \times \prod_{i=1}^{17} GC_i \times T^{0.91+0.01\times\sum_{i=1}^{5} W_i}$$

2.4.3 Desempenho de modelos de estimativa

O desempenho de um modelo de estimativa pode ser medido de diversas formas. A mais popular é a Magnitude Média do Erro Relativo (MMRE), onde o Erro Relativo (ER) é:

$$ER = \frac{Valor\ Real - Valor\ Estimado}{Valor\ Real}$$

A magnitude do erro relativo (MRE) é o seu valor absoluto e a MMRE é a média deste valor para n projetos, conforme a equação abaixo:

$$MMRE = \frac{1}{n} \times \sum_{i=1}^{n} abs(ER)$$

Outra métrica relevante é o nível de previsão de um modelo, que é dado por $PRED(l) = \frac{k}{n}$, onde k é o numero de projetos de uma amostra de tamanho n para os quais a MRE é inferior a l. Por exemplo, os k projetos para calcular o PRED(0.25) de um modelo serão aqueles cuja MRE for inferior a 0.25. É importante ressaltar que a análise da previsibilidade de um modelo de produtividade deve ser feita através de testes com projetos diferentes daqueles que foram utilizados no processo de construção do modelo.

2.5 Programação Genética aplicada a modelos de estimativa

Segundo Abran [Abran, 2015], pesquisadores tipicamente constroem modelos utilizando dados de projetos completos, sobre os quais não há incerteza. Diferentes abordagens e técnicas podem ser utilizadas para gerar as saídas do modelo, que são a função responsável por calcular custo/esforço/tempo e a variância do resultado. Entre estas técnicas está a Programação Genética, objeto de estudo deste trabalho.

Dolado [Dolado, 2001] utiliza a Programação Genética e a regressão linear clássica para gerar funções de custo de software, tendo como base diferentes *datasets* públicos com dados de projetos da indústria de software finalizados. Os dados conhecidos e registrados nestes *datasets* eram tempo e tamanho. Logo, as funções resultantes buscavam relacionar as duas variáveis, utilizando tamanho como a variável independente. As melhores equações foram selecionadas das várias execuções e seus valores de MMRE e PRED e foram comparados com os resultados de modelos lineares, gerados a partir dos mesmos *datasets*. O objetivo da comparação foi verificar se os resultados da técnica de Programação Genética

apresentava desvios significativos em relação àqueles apresentados pelos modelos lineares.

Dolado utiliza a implementação de Programação Genética vista em Mc Kay et al. [McKay et al., 1997] e Willis et al. [Willis et al., 1997], baseada na estrutura geral de um algoritmo genético. Os indivíduos evoluídos pelo algoritmo são árvores de equações matemáticas, que compõem a população que sofrerá a ação dos operadores genéticos. Os nós folha das árvores são constantes ou variáveis e os nós internos podem ser diversas funções, desde soma e subtração à logaritmo e exponencial. O número de rodadas do algoritmo de Programação Genética variou de 100 a 200 e, na maioria das vezes, o modelo obteve convergência muito rapidamente, gerando as melhores funções em poucas gerações. Entre as rodadas houve mudança nos principais parâmetros de execução do algoritmo. Duas funções de aptidão foram adotadas no uso das duas técnicas. O erro quadrático médio e o coeficiente de correlação. Dolado fez uma adaptação na implementação para o seu experimento, utilizando somente o erro quadrático médio como função de aptidão, pois o uso do coeficiente de correlação comumente se mostrava incapaz de distinguir equações.

Da comparação entre Programação Genética e regressão clássica, foi concluído por Dolado que o uso do tamanho como variável independente na estimativa de custo dos dois métodos não produz bons resultados preditivos. Esta avaliação foi feita sob a ótica de economias de escala, ou seja, ganhos de produtividade, que as estimativas de custo relacionando tamanho e esforço possam trazer para o processo de desenvolvimento. Os resultados o levaram a crer que talvez o tamanho, por si só, não seja o gerador de custo chave no desenvolvimento de software.

Olhando somente para as métricas preditivas dos dois modelos, os resultados do algoritmo de Programação Genética obtiveram melhores valores de PRED(0.25) em onze dos doze *datasets*, mas sob o custo de ter um valor um pouco pior de MMRE em vários casos. Somente em três *datasets* o algoritmo de Programação Genética obteve resultados significativamente melhores do que a regressão clássica. Nos demais, as variações entre os resultados foram pequenas.

No trabalho de Ferrucci [Ferrucci et al., 2010], o artigo de Dolado e os artigos de outros três autores são descritos e avaliados como os mais significantes estudos que compararam os resultados de estimativa de software de algoritmos genéticos (AG) com o de outras técnicas. Todas os trabalhos utilizaram o Erro Quadrático

Médio ou a MMRE como função de aptidão e a maioria usou MMRE e PRED(0.25) para avaliar os resultados. Nestes três outros trabalhos, a performance dos AG variou. Foi inferior em um deles, não apresentou ganhos ou perdas em outro e obteve resultados expressivamente melhores no último.

Também são apresentados e resumidos outros trabalhos onde algoritmos genéticos foram usados para melhorar o desempenho de técnicas de estimativa existentes. Eles já foram combinados às técnicas de Raciocínio Baseado em Casos, Redes Neurais Artificiais, Regressão por Vetores de Suporte e Análise Relacional Grey. O trabalho descrito como o primeiro a fazer uma combinação desta natureza foi feito por Shuckla(2000) e utilizou um algoritmo genético em conjunto com Redes Neurais artificiais. As redes neurais foram responsáveis por encontrar uma correlação existente entre as características do projeto e esforço e também alguma correlação existente entre as variáveis independentes. O algoritmo genético ficou responsável por minimizar o valor do Erro Quadrático Médio, que foi a função de aptidão adotada. A conclusão obtida através de testes de significância foi de que a abordagem da combinação Redes Neurais e Algoritmos Genéticos apresentou resultados superiores a métodos orientados a inteligência artificial comumente usados, como o CARTX.

[Ferrucci et al., 2010] descreve outros dois trabalhos que combinaram algoritmos genéticos à técnica de Raciocínio Baseado em Casos (RBC). Em um deles, o AG foi utilizado para ajustar o esforço reutilizado obtido considerando medidas de similaridade entre pares de projeto de software. No outro, atuou para otimizar a seleção de pesos de atributos e projetos. Nos dois casos, a estimativa gerada pela combinação foi significantemente melhor se comparada à estimativa gerada pela RBC.

2.6 Considerações finais

A partir do que foi observado em Dolado [Dolado, 2001] e Ferrucci [Ferrucci et al., 2010] pode-se concluir que é possível construir um modelo de estimativa de custo de software usando a abordagem evolutiva dos algoritmos genéticos. A geração de funções de custo que maximizem ou minimizem determinadas métricas de capacidade preditiva com base em dados históricos pode ser definida como um problema de otimização. Como os algoritmos genéticos atuam justamente sobre

esse tipo de problemas, modelos que os utilizam geram soluções comparáveis aos resultados de técnicas de estimativa amplamente utilizados.

A Programação Genética também pode ser aplicada em um modelo que gera funções de custo. A calibragem do modelo de Programação Genética utilizado pode ser feita modificando a função de aptidão para maximizar ou minimizar métricas de capacidade preditiva e alterando outros parâmetros de funcionamento do algoritmo. O benefício de seu uso é a exploração de um vasto espaço de funções e todos estes mecanismos de controle dos resultados obtidos ao longo das gerações.

Apesar do cenário geral aparentemente nebuloso das estimativas de software, onde nem sempre a relação custo-tamanho das soluções existentes fornece resultados satisfatórios, o uso de modelos que mantenham um histórico de uso, que usem variáveis bem documentadas e que sejam facilmente utilizados é uma poderosa ferramenta para o gerente de projetos e do tomador de decisões.

No próximo capítulo, o uso da Programação Genética para gerar fórmulas que relacionam tamanho do software e esforço/tempo/custo de desenvolvimento a partir de dados conhecidos de projetos será abordado na implementação.

3. Um Algoritmo de Programação Genética para Estimativas em Projetos de Software

3.1 Introdução

Neste capítulo, será detalhada a implementação de um algoritmo de Programação Genética visando melhorar estimativas de tempo de desenvolvimento em projetos de software. O algoritmo recebe um conjunto de pares de valores x e y como entrada e seu objetivo é gerar uma expressão $y = f(x)$ que minimize a MMRE (Seção 2.4.3) para essa entrada ao longo das gerações.

Os indivíduos que formam a população manipulada pelo algoritmo são expressões matemáticas. Estas expressões são representadas como árvores binárias, nas quais os nós internos são funções e nós terminais são números ou a variável x. A representação destas expressões será chamada de *árvore de expressão*. A população inicial é gerada sem que haja conhecimento sobre as características dos dados de entrada. O resultado obtido ao final da última geração é a função de x com o menor valor de MMRE para os dados de entrada selecionados.

Os elementos necessários para a execução do algoritmo serão detalhados nas próximas seções, incluindo parâmetros de controle (como o número de gerações e o tamanho da população), conjuntos de terminais e de funções, tipos de operadores genéticos, mecanismos para geração da população inicial e avaliação de indivíduos componentes da população.

3.2 Parâmetros e características do algoritmo

Esta seção apresenta a configuração do algoritmo de Programação Genética, incluindo os seus parâmetros e as possibilidades de manipulação das árvores de expressão.

3.2.1 Conjunto de terminais

O conjunto de terminais representa os valores que podem ser atribuídos aos nós folha das árvores de expressão pelo algoritmo de Programação Genética. Na implementação proposta, ele é formado pelo conjunto dos números racionais ($\mathbb{Q}$) menos o zero e pela variável x, como pode ser visto na Figura 3.1. O valor zero não foi considerado porque as operações selecionadas sempre podem ser simplificadas quando usam zero como um dos seus operandos.

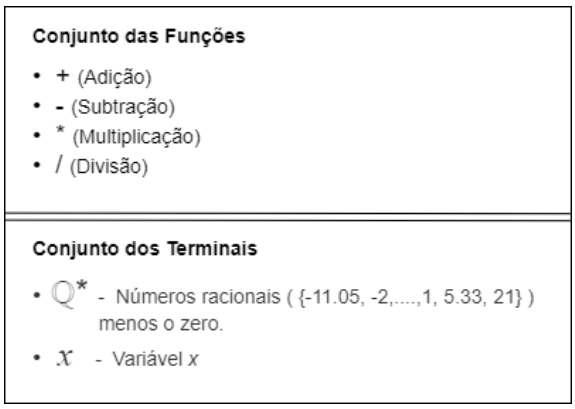

Figura 3.1: Conjuntos dos terminais e das funções utilizados pelo algoritmo.

3.2.2 Conjunto de funções

O conjunto de funções representa os valores que podem ser atribuídos aos nós internos (não folha) das árvores de expressão pelo algoritmo de Programação Genética. O conjunto das funções utilizado na implementação proposta é formado por adição ($+$), subtração ($-$), multiplicação ($\times$) e divisão ($\div$).

3.2.3 Geração da população inicial

A geração da população inicial é o primeiro passo da execução do algoritmo de Programação Genética. O método utilizado para gerar as árvores iniciais é o *Ramped Half-and-Half*, apresentado na Seção 2.3.1. Este método é a combinação dos métodos *Grow* e *Full*, cada um gerando metade da população inicial. A geração

semi-aleatória da população inicial não foi escolhida pois não se sabia nenhuma informação à respeito da estrutura dos resultados desejados.

Ao gerar as árvores de expressão, seja com *Grow* ou *Full*, os nós são preenchidos de forma aleatória. Toda função possui a mesma probabilidade de ser escolhida para um nó interno. Já no preenchimento dos nós folha, existem restrições. Para a geração da população inicial, o conjunto dos terminais foi limitado ao conjunto dos números inteiros ($\mathbb{Z}$) entre -9 e 9 menos o zero e à variável x.

As probabilidades de um nó folha ser um inteiro e de ser o terminal x são, respectivamente, de 82% e 18%, em valores aproximados. Caso a escolha seja um inteiro, um novo sorteio aleatório é feito para decidir se ele será positivo ou negativo. A probabilidade individual de um inteiro pertencente ao intervalo [-9, -8,...,8, 9] menos o zero é de aproximadamente 4,5%, cerca de quatro vezes menor do que a probabilidade de escolha do terminal x.

A escolha de uma probabilidade maior para o x se deve ao fato de não haver exponenciação no conjunto das funções utilizadas pelo algoritmo. Para que o algoritmo seja capaz de gerar polinômios de grau 2 e 3 com frequência e, consequentemente, explorar um espaço de resultados maior, a incidência de x nas árvores precisou ser estimulada. Apenas árvores de expressão que contenham pelo menos um terminal x são consideradas válidas.

A profundidade limite das árvores da população inicial foi estabelecida como quatro. A profundidade das árvores *Full* é sempre quatro e a das árvores *Grow* é quatro ou menos. Exemplos de árvores *Full* e *Grow* geradas na população inicial podem ser observados nas Figuras 3.2 e 3.3. A escolha deste limite de profundidade buscou alcançar dois objetivos: (i) aumentar a probabilidade da formação de polinômios de graus 2 e 3; e (ii) facilitar a geração de números fracionários nos terminais. Os dois objetivos contribuem para a geração de expressões complexas já na população inicial. No restante da execução do algoritmo, nenhuma restrição de profundidade é imposta às árvores.

As escolhas de utilizar um conjunto de funções pequeno, com apenas quatro funções, e de restringir o conjunto de terminais na geração da população inicial tiveram o propósito de simplificar as árvores geradas, facilitando sua leitura e ajudando no processo de implementação.

A restrição imposta na geração da população inicial não impede que números racionais não inteiros venham a ocupar os nós folha. Durante a execução do algo-

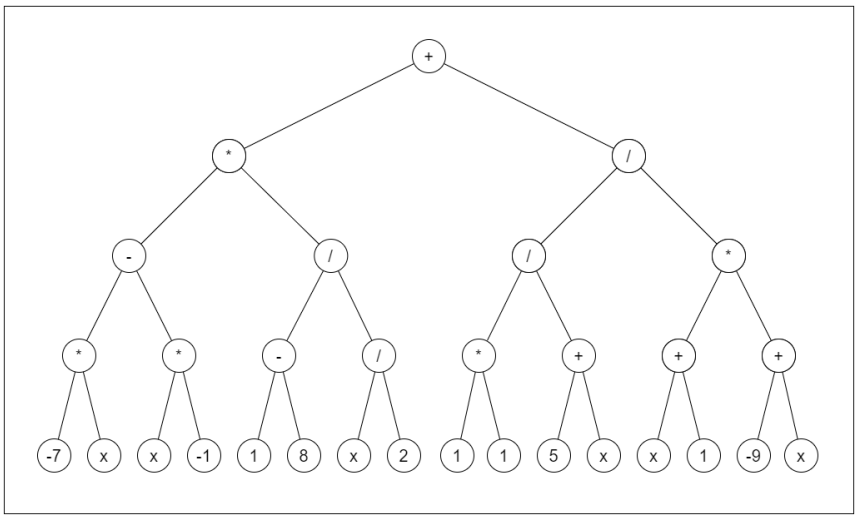

Figura 3.2: Exemplo de árvore Full de profundidade 4 gerada na população inicial.

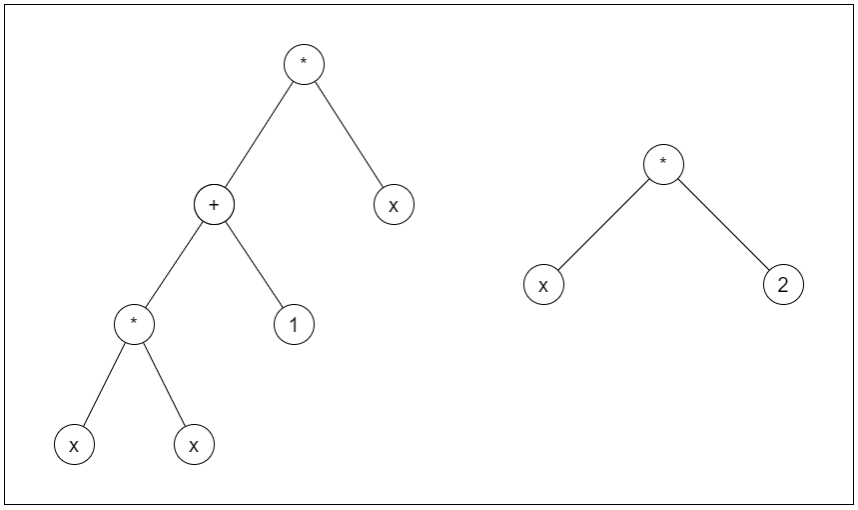

Figura 3.3: Exemplo de árvores Grow, de profundidades 3 e 1, geradas na população inicial.

ritmo, as árvores passam por um processo de simplificação. Nele, operações com resultado conhecido entre nós terminais são resolvidas, diminuindo a ramificação das árvores. Por exemplo, se um nó interno de divisão (÷) possui nós filhos constantes (digamos, 3 e 4), respectivamente à esquerda e à direita, a simplificação eliminará os nós filhos e substituirá o operador de divisão por um nó terminal com o valor $0,75$. Como os nós da população inicial são inteiros, a geração dos primeiros números fracionários depende da presença da função de divisão.

3.2.4 Número de gerações e tamanho da população

O número de gerações foi estabelecido como cinquenta e a população manipulada pelo algoritmo é formada por setenta árvores. Ambos os números são fixos e não existe outro critério de parada da execução do algoritmo senão o fim das cinquenta gerações. Durante a implementação, números de gerações e tamanhos de população menores foram testados. Na primeira execução do algoritmo, o tamanho da população era 20, o número de gerações era 10 e o limite de profundidade das árvores da população inicial era 3. Uma melhoria nos resultados foi percebida ao aumentar os valores desses parâmetros, pois árvores mais complexas eram necessárias para atingir bons valores de MMRE em algumas execuções, que nem sempre convergiam nas gerações iniciais. O aumento dos parâmetros também respeitou os recursos computacionais disponíveis à época.

3.3 Operadores genéticos

A cada avanço de geração, uma nova população é gerada. Alguns dos indivíduos com maior aptidão permanecem na população, inalterados. Os operadores genéticos de seleção, mutação e *crossover* são utilizados para gerar o restante dos indivíduos, até que a população alcance setenta indivíduos. Após a mudança de geração, a MMRE de cada árvore de expressão é avaliada e a população é ordenada da melhor árvore para a pior.

3.3.1 Reprodução e seleção

A reprodução copia os melhores indivíduos para a população da próxima geração. Ela deve ocorrer a cada geração, após a avaliação da aptidão dos indivíduos e antes da seleção. O nível de elitismo escolhido para esta implementação, ou seja, a parcela dos melhores indivíduos copiados para a próxima geração, é de 30%. Como o potencial de alteração das árvores no algoritmo proposto é alto, a taxa de elitismo de 30% buscou evitar que bom material genético fosse perdido com o avanço de gerações, sem prejudicar a natureza exploratória do algoritmo. A Figura 3.4 apresenta os efeitos do elitismo no processo evolutivo.

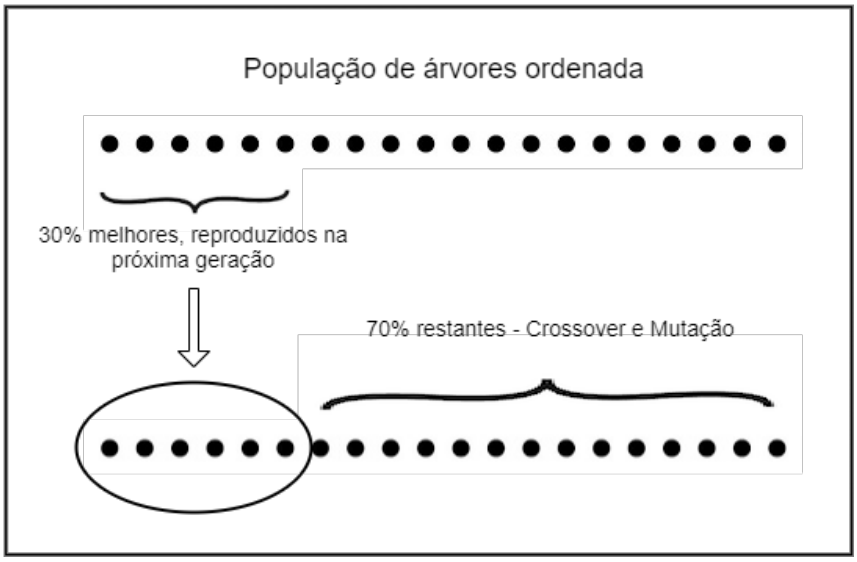

Figura 3.4: Reprodução dos 30% melhores indivíduos para a próxima geração.

3.3.2 Seleção

Os 70% de indivíduos restantes na população da próxima geração são criados através da ação dos operadores de mutação e *crossover*. A seleção é responsável por decidir quais árvores de expressão serão alvo de qual operador. Para cada novo indivíduo na população da nova geração, primeiro é feito um sorteio para decidir qual operador será responsável por gerá-lo. O *crossover* é escolhido com 75% de probabilidade e a mutação é escolhida com 25%. Esta taxa de mutação é considerada alta e também representa um esforço para gerar diversidade na população. A aplicação dos operadores só origina um novo indivíduo.

O tipo de seleção utilizado é o torneio clássico, mencionado no Capítulo 2. A escolha deste operador de seleção teve base na literatura de referência. O torneio seleciona um ou dois indivíduos para sofrer a ação do operador genético sorteado, sendo um para mutação e dois para *crossover*. O tamanho do torneio (k) foi estabelecido como cinco e a probabilidade p foi estabelecida como 70%. Assim, cinco indivíduos da população são selecionados aleatoriamente para participar do torneio. Indivíduos copiados para a próxima geração também podem ser sorteados para participar nos torneios. O indivíduo com menor MMRE será selecionado com a probabilidade 70%, o segundo com probabilidade 21%, o terceiro com 6,3%, e assim por diante. No caso do *crossover*, após a retirada do primeiro vencedor, o torneio é repetido com os quatro participantes restantes, impedindo que o mesmo indivíduo seja selecionado duas vezes.

3.3.3 *Crossover*

O operador de *crossover* utilizado no algoritmo é o *crossover* de subárvore. Ele gera um descendente pela combinação de duas subárvores. Para obter estas subárvores, um nó interno é sorteado aleatoriamente em cada uma das duas árvores vencedoras do torneio e ele se torna o ponto de quebra, originando uma subárvore de cada árvore. Um novo sorteio aleatório é feito para determinar qual das duas raízes de subárvore terá seu filho esquerdo substituído pela outra subárvore. A árvore resultante da combinação é simplificada pela redução de operações sobre constantes e validada: caso ela não possua um nó x, ela é descartada e um novo torneio é realizado para selecionar os participantes do novo *crossover*. Na Figura 3.5, é possível ver o funcionamento do *crossover* de subárvore.

3.3.4 Mutação

O operador de mutação utilizado no algoritmo é a mutação de ponto. Este operador possui um alto potencial de inserção de material genético novo na população. Um nó aleatório, incluindo nós folha, é selecionado na árvore alvo do operador e este nó se torna o ponto de mutação.

Optou-se por utilizar o método *Grow* de geração de árvores como parte do operador de mutação. Em vez de somente substituir o terminal ou função do nó selecionado como ponto de mutação por outro terminal ou função, a escolha foi substituir a subárvore com raiz no ponto de mutação por uma árvore *Grow*.

A geração da árvore *Grow* segue o mesmo padrão utilizado na população inicial, mas a profundidade limite das árvores *Grow* é igual à profundidade da árvore que sofreu a mutação. A subárvore que terá raiz no ponto de mutação sempre terá profundidade igual ou menor à da árvore original, logo o potencial de crescimento da árvore que sofre mutação corresponde ao seu tamanho.

O uso das árvores *Grow* para gerar a subárvore aleatória necessária para a mutação não se embasou na literatura e se trata da reutilização de uma funcionalidade já implementada. Na Figura 3.6 é possível ver uma mutação de ponto onde a árvore *Grow* gerada possui a mesma profundidade da árvore inicial.

A execução do algoritmo gera árvores cada vez maiores e a mutação tem seu papel nisto. A árvore resultante da mutação pode ter sua profundidade p

aumentada para até $2 \times p$ quando o ponto de mutação é uma folha no nível mais baixo da árvore e a árvore *Grow* resultante tem profundidade p.

3.4 Considerações finais

O algoritmo proposto reúne diversos elementos da literatura clássica de Programação Genética e consegue gerar expressões matemáticas minimizando a métrica de aptidão. Diversas estratégias para contornar a baixa diversidade da população inicial foram descritas e garantem a exploração de um espaço maior de terminais e soluções e a geração de árvores de tamanhos variados.

A adição de mais funções no conjunto de funções, a expansão do conjunto de terminais e a geração de árvores com maior diversidade na população inicial são possíveis melhorias na capacidade exploratória do algoritmo proposto. No próximo capítulo, o algoritmo é utilizado no contexto de estimativa de software para gerar funções de custo a partir de dados conhecidos de projetos.

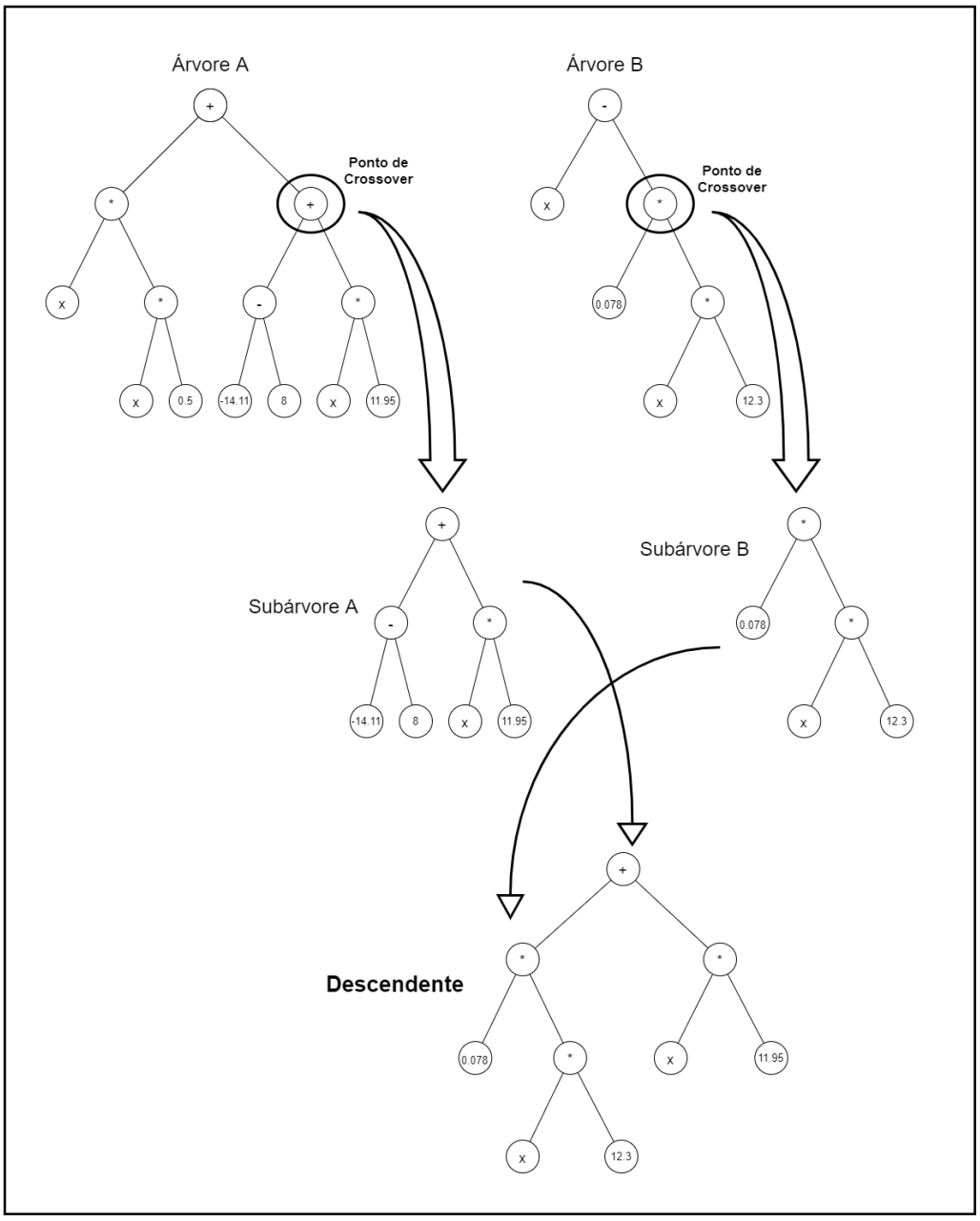

Figura 3.5: *Crossover* de subárvore.

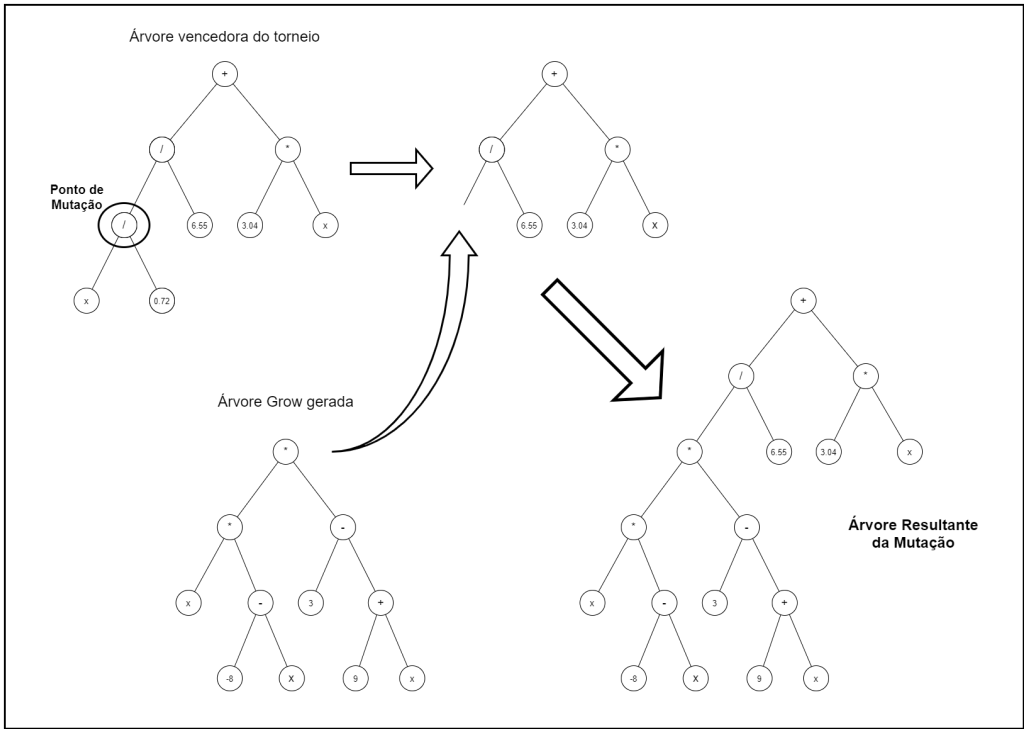

Figura 3.6: Mutação de ponto.

4. Resultados

Neste capítulo serão apresentados os resultados obtidos pela aplicação do algoritmo proposto no Capítulo 3 em doze conjuntos de dados utilizados por Dolado [Dolado, 2001] no artigo que serviu de inspiração para a realização deste trabalho de pesquisa (Seção 2.5). A MMRE dos resultados da aplicação do algoritmo proposto, que é minimizada pela otimização, é comparada com os resultados do algoritmo de Programação Genética apresentado no artigo original [Dolado, 2001]. Como resultado das análises realizadas, foram obtidos valores de MMRE em média 27,5% menores do que aqueles obtidos por Dolado. Outros dados referentes à distribuição dos valores de MMRE também foram analisados.

4.1 Conjuntos de dados utilizados

As observações utilizadas na avaliação do algoritmo proposto no Capítulo 3 consistem em doze conjuntos de dados com dados de projetos de desenvolvimento de software, cada qual utilizado por autores diferentes para fins de análise ou teste de modelos de estimativas de esforço em projetos de software.

Os conjuntos de dados são formados por várias observações, cada qual descrita como um par de tamanho e esforço. A variável independente é uma métrica de tamanho de projeto de software. Várias métricas são utilizadas, incluindo pontos de função, linhas de código, número de arquivos, entre outras. A métrica mais comum é a de pontos de função, utilizada em sete dos doze conjuntos de dados. A variável dependente é o esforço necessário para completar o projeto.

Dolado [Dolado, 2001] apresenta uma breve descrição dos conjuntos de dados. O conjunto de dados reunidos por *Barry Boehm* é um dos mais analisados na literatura de Engenharia de Software, pois foi utilizado para a construção do

modelo COCOMO [Boehm, 1981]. Ele é o único entre os doze conjuntos de dados que utiliza a métrica de tamanho "Instruções de Fonte Entregues Ajustadas" e possui a maior quantidade de observações (63). O conjunto de dados com a menor quantidade de observações é o coletado por *Kemerer*, com 15 pares de tamanho e esforço. Ele foi utilizado para testar diferentes métodos de estimativa e utiliza pontos de função como métrica de tamanho. Os demais conjuntos de dados são descritos a seguir:

- O conjunto de dados de *Albrecht and Gaffney*, com 24 observações, corresponde a projetos desenvolvidos pela IBM *Data Processing Services*;
- O conjunto de dados de *Abran and Robillard* contém 21 observações e é proveniente uma grande organização financeira do Canadá. A métrica de tamanho das observações é a de Pontos de Função Ajustados, que é uma das etapas do modelo de Análise de Pontos de Função, descrita na Seção 2.4.1;
- O conjunto de dados de *Bailey and Basili* tem 18 observações. A métrica de tamanho é a de milhares de linhas de código;
- O conjunto de dados de *Belady and Lehman* tem 33 observações. O tamanho é medido em linhas de código e o esforço em meses de trabalho;
- O conjunto de dados de *Heiat and Heiat* tem 35 observações. O artigo propõe um modelo de estimativa de software que calcula esforço em horas de trabalho e avalia as métricas de Pontos de Função e Linhas de Código;
- O conjunto de dados *Academic Environment* tem 48 observações. Os dados são uma combinação de vários projetos cujo tamanho e esforço foram medidos em linhas de código e horas de trabalho, respectivamente;
- O conjunto de dados de *Miyazaki et al.* tem 47 observações. A métrica de tamanho é a de milhares de linhas de código e os dados foram usados para testar o método de "regressão robusta";
- O conjunto de dados de *Shepperd and Schofield* tem 18 observações. A métrica de tamanho é a de número de arquivos e o conjunto de dados foi utilizado para testar o método de estimativa por analogia;
- O conjunto de dados de *Desharnais* tem 61 observações. O tamanho é medido em pontos de função e os conceitos de "entidade" e "transação" são utilizados para identificar os pontos de função;

– O conjunto de dados de *Kitchenham and Taylor* tem 33 observações. Se trata de uma combinação de dados de 33 projetos desenvolvidos na mesma linguagem, a S3. O tamanho do software é medido em linhas de código e o esforço em meses de trabalho.

4.2 Execução e análise do algoritmo

Para analisar a distribuição da MMRE das funções de custo geradas pelo algoritmo proposto, foi necessário analisar os resultados de diversas execuções devido ao uso de números pseudo-aleatórios pelo algoritmo. Foi decidido que os dados de 30 execuções do algoritmo para cada um dos doze conjuntos de dados seriam salvos e utilizados para a análise estatística da MMRE, em especial para a comparação com os resultados previamente publicados por Dolado [Dolado, 2001].

A Tabela 4.1 contém os resultados das 30 execuções realizadas para cada conjunto de dados, representados pelos nomes dos autores que colheram os dados, conforme apresentado na Seção 4.1. Na tabela, estão os valores da média (μ), da mediana (*Med.*), do desvio padrão (σ), do mínimo e do máximo MMRE encontrado entre as 30 execuções para cada conjunto de dados.

Dataset	**MMRE**					
	Min	**Max**	**Med.**	μ	σ	**J.J. Dolado**
Abran and Robillard	0.1989	0.2187	0.2061	0.2070	0.0051	0.256
Albrecht and Gaffney	0.4470	0.6071	0.4849	0.5071	0.0517	0.548
Bailey and Basili	0.2343	0.2635	0.2593	0.2560	0.0083	0.269
Belady and Lehman	0.4498	0.7569	0.4598	0.5671	0.1425	0.710
Boehm	0.5976	0.8014	0.6143	0.6765	0.0930	1.781
Heiat and Heiat	0.0849	0.1016	0.0895	0.0909	0.0040	0.087
Academic environment	0.3429	0.3822	0.3618	0.3604	0.0115	0.423
Kemerer	0.3902	0.3991	0.3935	0.3946	0.0022	0.584
Miyazaki et al.	0.3372	0.3727	0.3579	0.3575	0.0060	0.506
Shepperd and Schofield	0.3751	0.4718	0.4696	0.4580	0.0233	0.456
Desharnais	0.4734	0.5053	0.4996	0.4991	0.0062	0.623
Kitchenham and Taylor	0.6101	0.6364	0.6330	0.6324	0.0055	1.143

Tabela 4.1: Resultados da execução do algoritmo de programação genética proposto e resultados publicados pelo autor do artigo que deu origem a esta pesquisa.

Na última coluna da Tabela 4.1 está o valor da MMRE obtida pela função de custo de Dolado [Dolado, 2001] para cada conjunto de dados. É importante salientar que o autor publicou apenas uma função de custo para cada conjunto de

dados – a melhor função obtida depois de várias execuções – e que seu algoritmo buscava minimizar o erro quadrático médio como detalhado na Seção 2.5. O número de execuções do algoritmo de Dolado variou de 100 a 200.

Se compararmos o MMRE mínimo do algoritmo proposto, ou seja, a melhor função de custo obtida, com o MMRE da função de custo de Dolado [Dolado, 2001], é possível observar que o algoritmo implementado gera melhores valores de MMRE em todos os doze casos. Se a média das trinta execuções for comparada em vez do melhor valor, o algoritmo proposto gera melhor MMRE em dez dos doze casos, perdendo apenas nos conjuntos de dados *Shepperd and Schofield* e *Heiat and Heiat*.

O percentual médio de ganho da MMRE, comparando os valores mínimos do algoritmo proposto e a MMRE de Dolado, foi de 27,5%. O ganho foi calculado usando a Equação 4.1. Os maiores ganhos foram observados nos conjuntos de dados *Boehm* (66,4%) e *Kitchenham and Taylor* (46,6%). Nestes dois casos, a MMRE da função de custo de Dolado é muito maior (ou seja, pior) do que a MMRE calculada da função de custo do algoritmo proposto. Os menores ganhos foram observados nos conjuntos de dados *Heiat and Heiat* (2,4%) e *Bailey and Basili* (12,9%).

$$\text{Ganho} = \frac{\text{MMRE}_{\text{Dolado}} - \text{MMRE}_{\text{Min}}}{\text{MMRE}_{\text{Dolado}}} \tag{4.1}$$

A correlação (coeficiente de Pearson) encontrada entre o tamanho dos conjuntos de dados e o ganho observado na MMRE foi de 0.44, indicando que não há uma relação forte entre o tamanho do conjunto de dados e o erro médio da equação encontrada pelo algoritmo de programação genética.

É possível observar que em três dos doze conjuntos de dados (*Belady and Lehman*, *Albrecht and Gaffney* e *Boehm*) os valores de MMRE encontrados nas 30 execuções do algoritmo proposto espalham-se a uma distância maior do que a média. Os três conjuntos de dados possuem os maiores valores de desvio padrão entre os conjuntos observados. Nos nove demais, observa-se uma variação muito pequena nos valores de MMRE, que se encontram bastante concentrados em torno da média. Os dados utilizados para a construção da Tabela 4.1 também são apresentados nos gráficos de *boxplot* da Figura 4.1.

No gráfico, o limite superior da MMRE é 1 (100%) e os triângulos vermelhos sinalizam a MMRE encontrada por Dolado para cada conjunto de dados. Os

valores de Dolado para os conjuntos de dados *Boehm* e *Kitchenham and Taylor* são superiores a 1 e não são exibidos no gráfico.

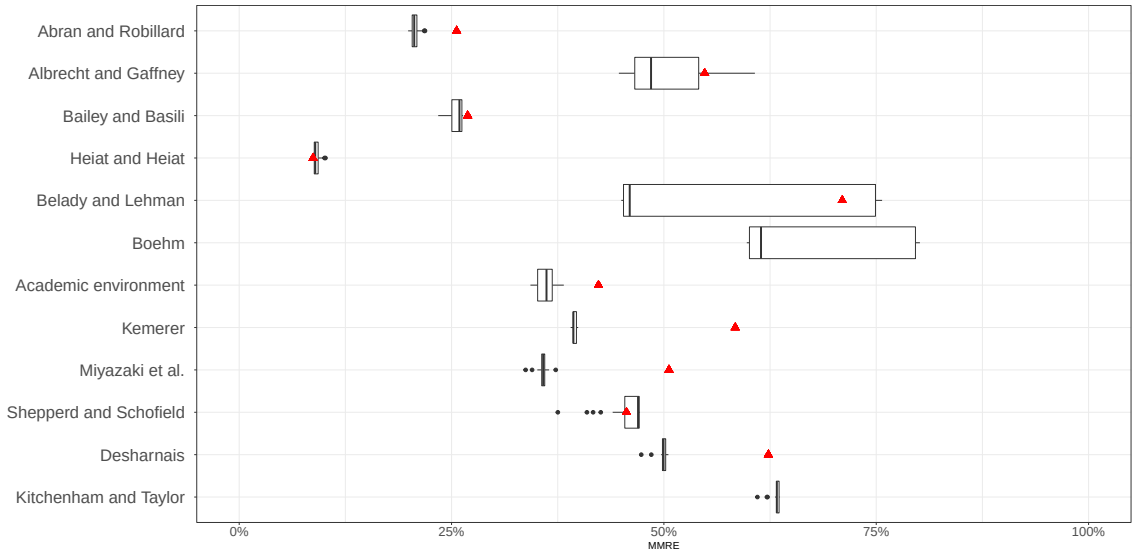

Figura 4.1: Boxplots gerados com os dados das 30 execuções do algoritmo proposto sobre os doze conjuntos de dados apresentados por Dolado [Dolado, 2001].

A Tabela **??** foi preenchida com as funções de custo obtidas para cada conjunto de dados, com o objetivo de ilustrar as saídas do algoritmo. Uma simplificação das funções encontradas pelo algoritmo foi feita para aumentar a legibilidade. Com exceção dos conjuntos de dados *Kemerer* e *Desharnais*, a melhor função de custo, ou seja, aquela com a menor MMRE das 30 rodadas, foi exibida na tabela. A segunda melhor função do conjunto *Kemerer* foi utilizada para a exibição na tabela, pois o número de termos das melhores equações é muito grande. Já no conjunto *Desharnais*, somente a quinta melhor função possuía um número satisfatório de termos para exibição na tabela.

Comparando-se os dois conjuntos de equações, percebe-se que as equações geradas pelo algoritmo proposto são mais complexas do que as geradas pelo algoritmo de Dolado, mesmo depois da etapa de simplificação aplicada sobre o resultado do algoritmo. Percebe-se também a falta de alguns operadores, notavelmente o operador de logaritmo neperiano, que está presente em algumas equações encontradas por Dolado. Por fim, com raras exceções (como o conjunto de dados de *Boehm*), percebe-se que as equações são muito diferentes entre si: linearidades em uma equação não são encontradas na sua contrapartida, assim como coeficientes similares nas potências.

Dataset	Expressão	Expressão Dolado
Abran and Robillard	$2.5 \cdot x - 21.48 + \frac{7.71 \cdot x^2 + 127.29 \cdot x - 2673}{x^2 - 99 \cdot x}$	$6.435 \cdot x^{0.8245}$
Albrecht and Gaffney	$\frac{9 \cdot x^2 - 50 \cdot x + 6}{272 \cdot x} - 6$	$1.06 \cdot 10^{-8} \cdot (1.775 \cdot 10^6 \cdot x + x^3)$
Bailey and Basili	$\frac{-2.14 \cdot x^2 + 19.22 \cdot x + 7.78}{10 \cdot x - x^2} + \frac{x + 1.5}{2.5 \cdot x^2 + 2.5 \cdot x} + x$	$x + \sqrt{1.416 \cdot x}$
Belady and Lehman	$\frac{0.064 \cdot x^2 - 0.64 \cdot x}{19 \cdot x - 130} - 4$	$3.83 \cdot 10^{-5} \cdot x^{3/2} \cdot (1 - 1.2 \cdot 10^{-6} \cdot x)$
Boehm	$2.157 \cdot 10^{-3} \cdot x + 2.5$	$9.432 \cdot 10^{-3} \cdot x$
Heiat and Heiat	$6.8x + \frac{272}{x} + 47.6 - \frac{40}{7} \frac{(8 \cdot x - 11)(x - 13)}{-8 \cdot x^3 + 187 \cdot x^2 + 726 \cdot x - 1326.5}$	$25.21 \cdot x^{0.6959}$
Academic environment	$0.055 \cdot x + 115$	$0.08843 \cdot x + \frac{824.5}{\ln x}$
Kemerer	$0.12 \cdot x + 11.18$	não disponibilizou
Miyazaki et al.	$0.66 \cdot x + \frac{5}{5.57x - 96} + 1.265$	$x + 5.3 \cdot 10^{-11} \cdot x^5$
Shepperd and Schofield	$\frac{999.54 \cdot x^2 - 1701 \cdot x}{1.93 \cdot x^2 + 225.17 \cdot x - 535.5}$	$54.93 \cdot x^{0.6621}$
Desharnais	$8x - \frac{-2x^3 - 14x^2 - 6x}{-3x^2 - 20x - 6} + 1012$	$32.9 \cdot x^{0.8854}$
Kitchenham and Taylor	$1 - \frac{63}{x} - \frac{17x^2 - 97x + 120}{(x+1) \cdot 0.00083x^2} + 14$	$0.03453 \cdot \frac{x}{\ln x}$

Tabela 4.2: Exemplos de funções de custo geradas pelo algoritmo proposto.

Quanto ao desempenho, foi observada uma grande variação no tempo de processamento necessário para concluir uma rodada. O tempo de uma execução de uma rodada variou da faixa dos 100 ms para a faixa dos 5.000 ms, considerando todos os conjuntos de dados. Valores superiores à 6.000 ms não foram registrados. O algoritmo foi executado em uma máquina pessoal e com capacidade de processamento limitado.

4.3 Considerações finais

O algoritmo implementado cumpriu o seu objetivo de gerar funções de custo através da técnica de Programação Genética, minimizando a métrica de MMRE. Foi possível verificar que o algoritmo é capaz de gerar resultados similares ou superiores (com menor MMRE) na maior parte dos casos através da tabela resumo e dos gráficos de *boxplot*. Também é importante ressaltar que nenhuma informação sobre os dados de entrada foi inserida como parâmetro do algoritmo.

Na comparação do algoritmo proposto com o algoritmo de Dolado, observou-se um ganho considerável na minimização da MMRE. Mesmo sabendo que a função de aptidão utilizada por Dolado buscava minimizar o erro quadrático médio e usava as métricas de MMRE e PRED(0.25) para determinar a capacidade

preditiva do modelo, consideramos que o desempenho superior ao gerar funções de custo com menor MMRE foi satisfatório.

5. Conclusão

5.1 Contribuições

Este trabalho descreveu os principais conceitos das áreas de Programação Genética e estimativas de esforço em projetos de software, além da interseção entre estas áreas. O artigo e experimento publicados por J.J. Dolado foram utilizados tanto como objeto de estudo quanto de comparação. O algoritmo proposto neste trabalho buscou gerar o mesmo tipo de resultados do algoritmo de Dolado, ou seja, funções que relacionassem o tamanho de um projeto de software com o esforço necessário para o seu desenvolvimento.

O algoritmo de Programação Genética foi desenvolvido para alcançar este objetivo. Conceitos clássicos da literatura, como a inicialização da população com o método *ramped half-and-half* (metade das árvores *Full* e metade *Grow*), *crossover* de subárvore e seleção por torneio, foram utilizados na implementação do algoritmo proposto. Além disso, ideias inovadoras, como o uso de árvores *Grow* geradas de forma pseudo-aleatória na mutação de subárvore, foram introduzidas. Ao final da implementação, o algoritmo conseguiu gerar funções de esforço com uma variável independente que minimizassem a MMRE para conjuntos de pares (x, y) recebidos como entrada.

A realização de trinta execuções do algoritmo para cada conjunto de dados permitiu que as características da distribuição dos valores de MMRE obtidos fossem analisadas. Pôde-se observar que a maioria dos valores de MMRE de diferentes execuções eram bastante próximos da média, o que indicava estabilidade de resultados entre execuções distintas. Além disso, o coeficiente de correlação entre o tamanho dos conjuntos de dados e a MMRE obtida pelo algoritmo, considerando o melhor resultado entre as trinta execuções, foi de 0.44, indicando que não existe relação forte entre o tamanho dos conjuntos e a MMRE.

Da comparação entre os valores de MMRE obtidos pelo algoritmo proposto com os valores obtidos por Dolado observou-se um ganho expressivo na maioria dos conjuntos de dados. Nos doze conjuntos o melhor resultado de MMRE obtido pelo algoritmo proposto foi menor (melhor) do que aquele obtido por Dolado. O percentual de ganho médio na comparação dos resultados dos doze conjuntos de dados foi de 27,5%. Foi possível, portanto, gerar funções de esforço melhores com o algoritmo de Programação Genética proposto.

5.2 Limitações do trabalho

Uma das limitações do trabalho foram as simplificações utilizadas em alguns pontos da implementação. O uso de apenas quatro operadores matemáticos (soma, subtração, produto e divisão) é um exemplo. Outros dois pontos são o uso de apenas números inteiros na geração de árvores *Full* e *Grow*, utilizadas na geração da população inicial e na mutação de subárvores, e o uso de um subconjunto limitado de operadores de simplificação de árvores. Não se sabe o impacto dessas duas limitações na qualidade dos resultados.

A forma como é feita a comparação entre os resultados do algoritmo proposto e os resultados de Dolado também é uma limitação: o algoritmo de Dolado utiliza tanto a métrica de MMRE quanto a métrica de PRED(0.25) para avaliar a capacidade preditiva dos seus modelos. Por outro lado, o algoritmo proposto neste trabalho considerou apenas a MMRE como objeto de comparação.

5.3 Trabalhos futuros

Uma possibilidade de trabalho futuro envolve o aprimoramento do algoritmo proposto, melhorando a simplificação de árvores e a verificação do impacto de melhoria nos resultados obtidos. Na mesma linha, a introdução de números com casas decimais na população inicial e a inclusão de mais operadores matemáticos (como potência e logaritmo neperiano) ao conjunto de funções também poderiam ser implementadas e testadas.

A mudança na função de aptidão do algoritmo proposto, minimizando a MMRE e maximizando o PRED(0.25), e uma nova comparação com os resultados de Dolado configuram uma possibilidade para futuros experimentos. A aná-

lise consideraria as duas métricas e muito provavelmente geraria um cenário de comparação diferente do observado neste trabalho.

Por fim, a aplicação do algoritmo proposto em outros cenários de geração de expressões matemáticas e a implementação de alterações que permitam ao algoritmo trabalhar com mais de uma variável independente também são sugestões para trabalhos futuros.

Bibliografia

[Abran, 2015] Abran, A. (2015). *Software Project Estimation: The Fundamentals for Providing High Quality Information to Decision Makers*. Wiley-IEEE Computer Society Pr, 1st edition.

[Banzhaf et al., 1998] Banzhaf, W., Francone, F. D., Keller, R. E., and Nordin, P. (1998). *Genetic Programming: An Introduction: on the Automatic Evolution of Computer Programs and Its Applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Boehm, 1981] Boehm, B. W. (1981). *Software Engineering Economics*. Prentice Hall PTR, USA, 1st edition.

[Dolado, 2001] Dolado, J. (2001). On the problem of the software cost function. *Information Software Technology*, 43.

[Dyer, 2008] Dyer, D. W. (2008). *Evolutionary Computation in Java A Practical Guide to the Watchmaker Framework*.

[Ferrucci et al., 2010] Ferrucci, F., Gravino, C., Oliveto, R., and Sarro, F. (2010). Using evolutionary based approaches to estimate software development effort. *Evolutionary computation and optimization algorithms in software engineering: applications and techniques*, page 13.

[Goldberg, 1989] Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition.

[IFPUG, 2005] IFPUG (2005). *Function Point Counting Practices Manual*. IFPUG.

[ISPA, 2008] ISPA (2008). *Parametric Estimating Handbook*. International Society of Parametric Analysts, 527 Maple Avenue East, Suite 301.

[Koza, 1992] Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

[Langdon et al., 2008] Langdon, W. B., Poli, R., McPhee, N. F., and Koza, J. R. (2008). *Genetic Programming: An Introduction and Tutorial, with a Survey of Techniques and Applications*. Springer, Berlin, Heidelberg, Berlin, 1 edition.

[Langdon and Qureshi, 1995] Langdon, W. B. and Qureshi, A. (1995). Genetic programming - computers using "natural selection"to generate programs. page 45.

[McKay et al., 1997] McKay, B., Willis, M., and Barton, G. (1997). Steady-state modelling of chemical process systems using genetic programming. *Computers Chemical Engineering*, 21(9):981 – 996.

[Miller et al., 1995] Miller, B. L., Miller, B. L., Goldberg, D. E., and Goldberg, D. E. (1995). Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, 9:193–212.

[Mitchell, 1998] Mitchell, M. (1998). *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, USA.

[Musilek et al., 2002] Musilek, P., Pedrycz, W., Sun, N., and Succi, G. (2002). On the sensitivity of cocomo ii software cost estimation model. pages 13 – 20.

[Pospieszny et al., 2017] Pospieszny, P., Czarnacka-Chrobot, B., and Kobyliński, A. (2017). An effective approach for software project effort and duration estimation with machine learning algorithms. *Working Paper*.

[Whitley, 1994] Whitley, D. (1994). A genetic algorithm tutorial. *Statistics and Computing*, 4(2):65–85.

[Willis et al., 1997] Willis, M., Hiden, H., Hinchliffe, M., McKay, B., and Barton, G. W. (1997). Systems modelling using genetic programming. *Computers Chemical Engineering*, 21:S1161 – S1166. Supplement to Computers and Chemical Engineering.