UNIVERSIDADE ESTADUAL DE CAMPINAS
Faculdade de Engenharia Elétrica e de Computação

João Victor Calvo Fracasso

Programação genética fundamentada em operadores genéticos semânticos multiobjetivo

Campinas

2019

João Victor Calvo Fracasso

Programação genética fundamentada em operadores genéticos semânticos multiobjetivo

Dissertação apresentada à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia Elétrica, na Área de Engenharia de Computação.

Orientador: Prof. Dr. Fernando José Von Zuben

Este exemplar corresponde à versão final da dissertação defendida pelo aluno João Victor Calvo Fracasso, e orientada pelo Prof. Dr. Fernando José Von Zuben.

Campinas

2019

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca da Área de Engenharia e Arquitetura
Luciana Pietrosanto Milla - CRB 8/8129

F847p

Fracasso, João Victor Calvo, 1991-
Programação genética fundamentada em operadores genéticos semânticos multiobjetivo / João Victor Calvo Fracasso. – Campinas, SP : [s.n.], 2019.

Orientador: Fernando José Von Zuben.
Dissertação (mestrado) – Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Programação genética (Computação). 2. Problema de otimização multiobjetivo. 3. Computação evolutiva. I. Von Zuben, Fernando José, 1968-. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: Genetic programming based on multi-objective semantic genetic operators
Palavras-chave em inglês:
Genetic programming (Computer science)
Multi-objective optimization problem
Evolutionary computation
Área de concentração: Engenharia de Computação
Titulação: Mestre em Engenharia Elétrica
Banca examinadora:
Fernando José Von Zuben [Orientador]
Levy Boccato
Fabrício Olivetti de França
Data de defesa: 09-12-2019
Programa de Pós-Graduação: Engenharia Elétrica

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0002-1109-3316
- Currículo Lattes do autor: http://lattes.cnpq.br/6828517041666640

COMISSÃO JULGADORA - DISSERTAÇÃO DE MESTRADO

Candidato(a): João Victor Calvo Fracasso RA: 190756
Data de defensa: 09 de dezembro de 2019
Titulo da dissertação: "Programação genética fundamentada em operadores genéticos semânticos multiobjetivo"

Prof. Dr. Fernando José Von Zuben (Presidente)
Prof. Dr. Levy Boccato
Prof. Dr. Fabrício Olivetti de França

A Ata de Defesa, com as respectivas assinaturas dos membros da Comissão Julgadora, encontra-se no SIGA (Sistema de Fluxo de Dissertação/Tese) e na Secretaria de Pós-Graduação da Faculdade de Engenharia Elétrica e de Computação.

Agradecimentos

Agradeço minha mãe Marina e meus irmãos Luiz Alfredo e Pedro Henrique, por sempre estarem ao meu lado fornecendo todo o suporte.

À minha namorada Raquel, pelo companheirismo, apoio e por dividir comigo todos os momentos bons e ruins.

À Universidade Estadual de Campinas e a todos os professores que contribuíram para o meu desenvolvimento pessoal e profissional.

Ao professor Fernando José Von Zuben, pela paciência e planejamento apresentado durante o processo de orientação.

A toda a equipe do LBiC pela atenção dispensada durante todo o período da pesquisa, os quais foram fundamentais para a produção desse trabalho.

A todos aqueles que colaboraram, de forma direta ou indireta, para que se tornasse possível a realização deste trabalho.

O presente trabalho foi realizado com apoio do CNPq, Conselho Nacional de Desenvolvimento Científico e Tecnológico – Brasil.

"If people do not believe that mathematics is simple, it is only because they do not realize how complicated life is"

(John Von Neumann)

Resumo

Programação Genética é um ramo da Computação Evolutiva dedicado à evolução de programas geralmente estruturados na forma de árvores sintáticas, embora outras codificações possam ser adotadas. Diversos operadores genéticos têm sido propostos para aumentar o poder da busca, uma vez que o espaço de programas factíveis é desafiador para ser devidamente explorado à procura de soluções de alta qualidade. Operadores genéticos semânticos estão ganhando atenção ultimamente, visto que promovem a natureza local da busca, possivelmente levando a uma evolução mais eficiente. No entanto, a Programação Genética Semântica pode sofrer o fenômeno de *bloating*, caracterizado por um aumento descontrolado no tamanho do programa ao longo das gerações. Algumas tentativas foram feitas na literatura para evitar o *bloating* do código e aqui estamos propondo três novos operadores semânticos para a codificação em árvore. Uma perspectiva multiobjetivo é adotada, onde as subárvores geradas correspondem a instâncias não dominadas em uma biblioteca previamente definida de subárvores candidatas. Vários objetivos conflitantes podem ser incorporados ao processo de tomada de decisão, como precisão, distância semântica a um comportamento de referência e tamanho da subárvore. Resultados experimentais revelam que os novos operadores propostos são eficazes em conter o *bloating*, sem um impacto negativo sobre as outras métricas de desempenho, além de serem competitivos com outras abordagens relevantes disponíveis na literatura.

Palavras-chaves: Programação genética; programação genética semântica; problema otimização multiobjetivo; não-dominância.

Abstract

Genetic Programming is a branch of Evolutionary Computation devoted to the evolution of programs generally structured in the form of syntatic trees, although other codifications can be adopted. Several genetic operators have been proposed to increase the power of the search, given that the space of admissible programs is very challenging to be properly explored toward high quality solutions. Semantically-driven genetic operators are gaining more attention lately, given that they promote the local nature of the search, possibly leading to a more efficient evolution. Nonetheless, Semantic Genetic Programming may undergo the bloat phenomenon, characterized by an uncontrolled increase in the program size along the generations. Some attempts have been made in the literature to mitigate code bloat, and here we are proposing three novel semantic-driven operators for the tree structure codification. A multi-objective perspective is adopted, where generated subtrees correspond to nondominated instances in a previously defined library of candidate subtrees. Several conflicting objectives may be incorporated into the decision making process, such as accuracy, semantic distance to a reference behaviour, and size of the subtree. Experimental results reveal that the new proposed operators are effective in restraining bloating, without a negative impact on the other performance metrics, besides being competitive with other relevant approaches available in the literature.

Keywords: Genetic programming; semantic genetic programming; multi-objective optimization problem; non-dominance.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AE	Algoritmo Evolutivo
AG	Algoritmos Genéticos
CM	*Competent Mutation*
CX	*Competent Crossover*
EAM	Erro Absoluto Médio
EE	Estratégias Evolutivas
EQM	Erro Quadrático Médio
GP	*Genetic Programming*
GRASM	*Greedy Randomized Aware Similarity Mutation*
GRASM	*Greedy Randomized Aware Similarity Crossover*
GRASP	*Greedy Randomized Adaptive Search Procedure*
LGX	*Locally Geometric Semantic Crossover*
MODM	*Multi-objective Desired Mutation*
MODX	*Multi-objective Desired Crossover*
MOO	*Multi-objective Optimization*
MORSM	*Multi-objective Randomized Similarity Mutation*
MORSX	*Multi-objective Similarity Crossover*
PE	Programação Evolutiva
RCL	*Restricted Candidate List*
RHH	*Ramped Half-and-half*
SAM	*Semantically Aware Mutation*
SAX	*Semantically Aware Crossover*
SBKP	*Semantic Backpropagation*

SDI	*Semantically Driven Initialization*
SDM	*Semantically Driven Mutation*
SDX	*Semantically Driven Crossover*
SGM	*Semantic Geometric Mutation*
SGP	*Semantic Genetic Programming*
SGX	*Semantic Geometric Crossover*
SM	*Standard Mutation*
SSM	*Semantic Similarity Based Mutation*
SSX	*Semantic Similarity Based Crossover*
SX	*Standard Crossover*

Sumário

1 Introdução

Algoritmos evolutivos pertencem a uma classe de algoritmos populacionais de busca inspirados na teoria da evolução de Darwin. Tais algoritmos visam uma busca eficiente em um espaço de soluções candidatas para obter soluções efetivas. Um dos subgrupos dos algoritmos evolutivos é a Programação Genética (GP, do inglês *Genetic Programming*) (KOZA, 1992), a qual é aplicada para que uma população de programas seja evoluída, geralmente associada a um espaço de busca de elevada dimensão.

A GP vem ganhando cada vez mais atenção ultimamente devido a uma ampla gama de aplicações bem-sucedidas (VANNESCHI; CASTELLI; SILVA, 2014; SUGANUMA; SHIRAKAWA; NAGAO, 2017) que requerem programação automática e/ou indução de modelos. A funcionalidade do programa pode variar de otimização combinatória para funções lógicas complexas, classificação e regressão, geralmente associadas a técnicas de aprendizado de máquina.

1.1 Motivação

Há uma característica negativa da GP em relação à estrutura do programa (sintaxe) e seu comportamento (semântica). Uma alteração mínima na sintaxe de uma solução candidata, também chamada de indivíduo da população, como, por exemplo, mudar a operação de soma indicada em um nó por uma multiplicação, pode modificar drasticamente a sua semântica. Por outro lado, uma alteração sintática substancial, como a refatoração de código, pode não ter influência no comportamento do programa. Esta característica leva à dificuldade em projetar operadores genéticos eficientes para GP.

Uma área de pesquisa promissora, denominada de Semantic Genetic Programming (SGP) (BEADLE, 2009; NGUYEN, 2011; PAWLAK, 2015; OLIVEIRA, 2016), oferece alguns operadores genéticos dirigidos semanticamente, os quais tentam controlar o comportamento dos programas enquanto realizam alterações em sua estrutura sintática. O monitoramento do efeito dos operadores no espaço fenotípico é interessante para evitar programas descendentes que sejam semanticamente equivalentes e para combinar códigos produzidos por programas semanticamente distintos (NGUYEN, 2011).

Um efeito colateral indesejável da SGP é a tendência de produzir um código complexo, mesmo para tarefas simples, geralmente interpretada como o fenômeno de *bloating* em GP (MORAGLIO; KRAWIEC; JOHNSON, 2012; BLEULER; BADER; ZITZLER, 2008). Esse aumento de complexidade do código influencia negativamente o processo evolutivo, aumentando, por exemplo, o tempo computacional para se realizar a busca por

boas soluções candidatas.

1.2 Objetivos

O objetivo desta pesquisa é examinar o importante papel dos operadores semânticos de GP, incluindo estratégias multiobjetivo que possam orientar modelos de regressão de alto desempenho, enquanto o fenômeno de *bloating* é, de alguma forma, minimizado. Problemas de regressão sintéticos e do mundo real são tomados como casos de estudos e operadores semânticos de alto desempenho e já propostos na literatura são considerados em uma análise comparativa. Em particular, abordamos as seguintes questões:

- Podemos empregar busca multiobjetivo para evitar o crescimento dos programas a partir da aplicação de operadores semânticos?
- Como a utilização de conceitos multiobjetivo pode melhorar a capacidade de generalização da GP?
- Qual o impacto das abordagens multiobjetivo no tempo computacional?

1.3 Organização do texto

O texto está organizado da seguinte maneira: no Capítulo 2, é feita uma breve introdução aos algoritmos evolutivos e uma introdução mais detalhada à programação genética; o Capítulo 3 é dedicado aos principais conceitos da programação genética semântica, além dos principais métodos semânticos existentes em operadores genéticos de cruzamento e mutação; os novos operadores propostos são detalhados no Capítulo 4, representando a principal contribuição da pesquisa; os resultados experimentais são apresentados e analisados no Capítulo 5; finalmente, o Capítulo 6 contém as considerações finais e perspectivas futuras da pesquisa.

2 Programação Genética

O objetivo deste capítulo é apresentar os principais conceitos dos algoritmos evolutivos, para então se concentrar na programação genética. Inicialmente, há uma breve introdução aos algoritmos evolutivos, contendo os pressupostos gerais e a configuração básica de um algoritmo evolutivo. Em seguida, as variantes existentes de algoritmos evolutivos são brevemente descritas. O restante do capítulo focará na programação genética.

2.1 Algoritmos Evolutivos

Um Algoritmo Evolutivo (AE) (BACK; FOGEL; MICHALEWICZ, 2000) é uma técnica de otimização inspirada na evolução natural. Por esta razão, a estrutura geral, bem como as nomenclaturas dos AEs, é em grande parte influenciada por conceitos presentes na biologia. Frequentemente, um algoritmo evolutivo é caracterizado por: (*i*) utilização de uma população de indivíduos, sendo que cada indivíduo representa um ponto no espaço de busca, o qual contém as soluções candidatas para um determinado problema; (*ii*) geração de descendentes a partir da população atual por um processo aleatório que envolve mutação e/ou cruzamento. A mutação corresponde a uma replicação alterada dos indivíduos, enquanto o cruzamento mescla informações derivadas de dois ou mais indivíduos existentes; (*iii*) avaliação de indivíduos por meio de uma medida de qualidade ou valor de aptidão (em inglês, *fitness*). De acordo com a medida de aptidão, o processo de seleção favorece a escolha de indivíduos mais aptos para sobreviverem e gerarem descendentes.

Existem diferentes formas de utilização e aplicação dos algoritmos evolutivos (BACK; FOGEL; MICHALEWICZ, 2000). Entre as mais conhecidas estão: algoritmo genético, estratégias evolutivas, programação evolutiva e programação genética. As principais características desses métodos em suas formas clássicas são:

- Algoritmos Genéticos (AG): propostos por Holland (1975), enfatizam a recombinação como o principal operador de busca e aplicam mutação com baixas probabilidades. Originalmente, utilizavam representação binária de indivíduos e seleção probabilística proporcional ao *fitness*.
- Estratégias Evolutivas (EE): desenvolvidas por Rechenberg (1965), Schwefel (1965) e Bienert na Universidade Técnica de Berlim por volta de 1964, empregam representação em vetores reais e mutações a partir de uma função densidade de probabilidade, sendo este o único operador genético aplicado ao processo de busca. Após a amostragem de novos indivíduos, um processo de seleção é aplicado considerando a população corrente e os novos indivíduos, ou somente os novos indivíduos;

- Programação Evolutiva (PE): proposta por Fogel, Owens e Walsh (1966) e voltada para a evolução de máquinas de estados finitos, operando com mutação e seleção probabilística.

- Programação Genética (PG): adaptação dos algoritmos genéticos para busca em um espaço de programas computacionais feita por Koza (1992). As estruturas de dados são representadas utilizando árvores, empregando operadores específicos de cruzamento e mutação, bem como seleção proporcional ao *fitness*.

O restante deste capítulo se dedica apenas à programação genética, a qual é utilizada ao longo do trabalho.

2.2 Programação Genética

A programação genética é uma técnica simples e poderosa que tem sido aplicada a uma ampla gama de problemas em otimização combinatória, programação automática e indução de modelos. Basicamente, é simulada em computador a seleção natural, onde os indivíduos ou programas são selecionados para sobreviver com base em sua aptidão (*fitness*). Os programas com maior aptidão têm maior chance de contribuir para a próxima geração de indivíduos, que herdarão parte do código de seus progenitores. Os operadores genéticos são, então, aplicados para promover a diversidade e explorar novas possibilidades de código, além de combinar códigos existentes. De geração em geração, o espaço de programas candidatos deve ser devidamente explorado por uma população de soluções candidatas, com a esperança de que as próximas gerações tendam a ser compostas por indivíduos cada vez melhores, em termos do valor de sua aptidão.

Mesmo para grandes espaços de busca, a programação genética demonstrou ser eficiente o suficiente para descobrir indivíduos de alta qualidade após consumir uma quantidade razoável de recursos computacionais, embora sem garantia de obtenção do ótimo global. Portanto, o melhor indivíduo depois de concluir a pesquisa pode ser a solução global ou uma boa aproximação. Geralmente, a melhoria média da aptidão é mais intensa nas gerações iniciais e a probabilidade de encontrar melhores soluções aumenta com o número de gerações e com o tamanho da população.

Em suma, o paradigma da programação genética consiste em (VANNESCHI, 2004):

1. Gerar uma população inicial de programas computacionais candidatos, geralmente a partir de uma codificação em forma de árvore sintática;

2. Executar iterativamente os seguintes passos até que o critério de parada seja atendido:

a) Executar cada programa da população e atribuir um valor de aptidão de acordo com o quão bem ele resolve o problema

b) Criar uma nova população através da aplicação das seguintes operações:

   i. Selecionar probabilisticamente um conjunto de programas de computador para sobreviver para a próxima geração, com base na sua aptidão (seleção);

   ii. Copiar alguns dos indivíduos selecionados, sem modificá-los, na nova população (reprodução);

   iii. Criar novos indivíduos recombinando geneticamente subárvores escolhidas aleatoriamente de dois indivíduos selecionados (cruzamento);

   iv. Criar novos indivíduos, substituindo subárvores existentes por subárvores geradas aleatoriamente (mutação).

3. O programa de computador com maior valor de aptidão que aparecer ao longo das gerações é designado como o resultado final do processo de busca. Este resultado pode ser uma solução ótima ou uma solução aproximada para o problema.

2.2.1 Representação ou codificação dos programas candidatos

A programação genética se distingue de outras aplicações de algoritmos evolutivos por tratar de soluções representadas como programas interpretáveis. A representação mais comum em GP é a estrutura em árvore, para explicar os aspectos sintáticos do programa. Esta será a codificação adotada neste trabalho. No entanto, outros mecanismos de codificação poderiam ter sido considerados, tais como: representação linear (POLI; LANGDON; MCPHEE, 2008), representação baseada em regras de produção (O'NEIL; RYAN, 2003), representação cartesiana (MILLER; THOMSON, 2000) e representação baseada em grafos (LANGDON; POLI, 2002).

Para as estruturas em árvore, é feita a definição do conjunto de símbolos não-terminais $\mathcal{F}$ e de símbolos terminais $\mathcal{T}$ (KOZA, 1992; POLI; LANGDON; MCPHEE, 2008). O conjunto de símbolos terminais $\mathcal{T} = \{t_1, t_2, \ldots, t_m\}$ inclui constantes, que retornam o mesmo valor em cada execução, variáveis nomeadas, que lêem os argumentos do programa durante a execução e funções sem parâmetros, por exemplo, a função `RAND`. Os símbolos não-terminais $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$ são tipicamente dependentes do domínio, como operadores matemáticos ou booleanos. A Figura 2.1 apresenta dois exemplos de indivíduos (programas) utilizando a representação em árvore, para os conjuntos $\mathcal{F} = \{+, -, \times, \div\}$ e $\mathcal{T} = \{x, y\}$.

Os conjuntos de símbolos terminais e não-terminais são muito importantes para o processo de busca, uma vez que definem diretamente o espaço de soluções candidatas. Se esses conjuntos forem pequenos ou não incluírem algum elemento relevante para a solução do problema, as soluções poderão ser insatisfatórias. Por outro lado, se os conjuntos

(a) Indivíduo 1 (b) Indivíduo 2

Figura 2.1 – Indivíduos (programas) utilizando a representação em árvore.

contiverem muitos elementos, o tamanho do espaço de busca aumentará significativamente, dificultando a busca.

Na literatura de GP, é comum representar expressões (indivíduos) utilizando a notação de parênteses presentes em *Lisp* ou *Scheme*. Por exemplo, os indivíduos da Figura 2.1 são representados por: `(+ (-` x `(+` y y`)) x)` e `(`$\times$ `(-` x x`) (`$\div$ y x`))`, respectivamente. Essa notação geralmente facilita a visualização da relação entre (sub) expressões e suas (sub) árvores correspondentes.

2.2.2 Inicialização da População

Inicialmente, é necessário conceber a população inicial que irá ser utilizada ao longo do processo evolutivo. Assim como ocorre para outros algoritmos evolutivos, na GP os indivíduos da população inicial são tipicamente gerados aleatoriamente. Uma boa diversidade de soluções candidatas na população inicial pode ser crucial para o sucesso em obter boas soluções ao final do processo evolutivo. Em geral, há um número muito elevado de indivíduos possíveis. Isso significa que é impossível procurá-los de forma exaustiva. Portanto, qualquer método usado para criar a população inicial terá um viés influenciando a solução final. Existem diferentes métodos para gerar indivíduos aleatoriamente. Os métodos mais utilizados foram propostos por Koza (1992), sendo conhecidos como *full*, *grow* e *ramped half-and-half*.

O método de inicialização *full* consiste em criar árvores com funções aleatórias do conjunto $\mathcal{F}$, tal que o comprimento de cada ramo é igual à profundidade máxima especificada. Posteriormente, escolhem-se aleatoriamente itens do conjunto de terminais $\mathcal{T}$ para cada folha de cada ramo. Para a metodologia *grow*, os ramos das árvores que representam indivíduos podem ter diversos comprimentos, gerando, assim, árvores assimétricas. Este método faz uma seleção aleatória do conjunto $\mathcal{C} = \mathcal{F} \cup \mathcal{T}$ até que seja obtida a profundidade máxima desejada ou sejam completados os nós que possuírem instruções não-terminais com itens terminais. Por fim, o método *ramped half-and-half* (RHH) consiste em uma combinação do método *full* com o método *grow*. RHH inicializa a população com uma profundidade máxima, gerando metade dos indivíduos com a metodologia *full* e a outra metade de indivíduos com a metodologia *grow*.

Embora esses métodos sejam fáceis de implementar e usar, eles geralmente dificultam o controle das distribuições estatísticas de propriedades importantes, como os tamanhos e as formas das árvores geradas. Por exemplo, os tamanhos e formas das árvores geradas pelo método *grow* são altamente sensíveis aos tamanhos das funções e conjuntos de terminais. Se, por exemplo, houver significativamente mais terminais do que funções, o método *grow* quase sempre gerará árvores muito curtas, independentemente do limite de profundidade. Da mesma forma, se o número de funções for consideravelmente maior que o número de terminais, o método *grow* se comportará de maneira bastante semelhante ao método *full*. As aridades das funções no conjunto de símbolos não-terminais também influenciam o tamanho e a forma das árvores produzidas pelo crescimento. Na Seção 3.3, outras técnicas para a construção da população inicial são abordadas, utilizando informação semântica dos indivíduos.

2.2.3 Função de Avaliação

A programação genética consiste em gerar um programa p que maximize (ou minimize) uma função de avaliação $f(\cdot)$. Fazendo analogia com a biologia, a função de avaliação indica o quão apto um indivíduo é para um determinado ambiente, indicando assim, o que são "soluções boas" e "soluções ruins".

A avaliação de um indivíduo depende dos resultados produzidos por sua execução para uma variedade de condições diferentes. Por exemplo, o indivíduo pode ser testado em todas as combinações possíveis de entradas $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$. Alternativamente, em sistemas de controle, por exemplo, tira-se a média dos resultados das ações realizadas por um programa para diferentes condições iniciais.

Para muitos problemas, esta medição produz um único valor numérico explícito, chamado *fitness* ou grau de aptidão. O *fitness* pode ser medido de várias maneiras, por exemplo: a quantidade de erro entre sua saída e a saída desejada; a quantidade de tempo (combustível, dinheiro, etc.) necessária para levar um sistema a um estado-alvo desejado; a precisão do programa em reconhecer padrões ou classificar objetos (POLI; LANGDON; MCPHEE, 2008).

Outra característica comum nas funções de avaliação em GP é que muitos problemas práticos envolvem múltiplos objetivos conflitantes, isto é, eles combinam dois ou mais elementos diferentes que frequentemente são antagônicos. A área de otimização multiobjetivo é uma área complexa e ativa de pesquisa em GP e aprendizado de máquina em geral. A Seção 2.3 aborda diferentes aplicações de otimização multiobjetivo no contexto de GP.

Ao longo deste trabalho, problemas de regressão serão tratados. Assim, as funções de avaliação mais usuais para o cálculo de *fitness* são o erro absoluto médio (EAM)

e o erro quadrático médio (EQM). Dados N pares $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}$, então, as funções de avaliação do indivíduo i, $f(p_i)$, podem ser definidas como:

$$EAM : f(p_i) = \frac{1}{N}\sum_{j=1}^{N} |p(\mathbf{x}_j) - y_j| \quad (2.1) \qquad EQM : f(p_i) = \frac{1}{N}\sum_{j=1}^{N} (p(\mathbf{x}_j) - y_j)^2. \quad (2.2)$$

2.2.4 Operador de Seleção

No processo evolutivo, a seleção de indivíduos para compor a próxima geração e/ou participar de operações genéticas dedicadas a produzir descendentes são procedimentos necessários para implementar a sobrevivência do mais apto. Observe que cada indivíduo da população possui um valor de aptidão, indicando sua capacidade de sobrevivência em relação aos outros indivíduos.

O método de seleção proporcional ao *fitness*, baseada no algoritmo da roleta, (em inglês, *roulette wheel*) (KOZA, 1992) e a seleção por torneio (LANGDON; POLI, 2002) são candidatos muito populares, entre outras possibilidades, como a seleção baseada em *rank* (EBNER et al., 1998). A seleção por torneio geralmente é adotada em GP devido ao melhor gerenciamento da pressão seletiva, não comprometendo a diversidade populacional tão cedo.

Mais indivíduos no torneio implica em uma pressão seletiva mais intensa, uma vez que os indivíduos que compõem o torneio são escolhidos aleatoriamente, sem considerar seu *fitness*. O mesmo indivíduo pode aparecer em multiplicidade no mesmo torneio e o vencedor é o indivíduo com a melhor aptidão.

2.2.5 Operadores Genéticos

A GP emprega diferentes operadores genéticos para modificar a população durante o processo de evolução. Os mais comuns são operadores de cruzamento, mutação e reprodução (KOZA, 1992; KOZA, 1994; LANGDON; POLI, 2002; POLI; LANGDON; MCPHEE, 2008).

O Pseudocódigo 1 apresenta o operador de mutação, o qual escolhe uma subárvore aleatória de um progenitor selecionado e a substitui por uma nova subárvore gerada aleatoriamente (Figura 2.2a). Outra forma comum de mutação é a mutação pontual, selecionando um nó aleatório e substituindo por uma função diferente com a mesma quantidade de argumentos contida no conjunto de não-terminais, $\mathcal{F}$.

Já no caso do operador de cruzamento, presente no Pseudocódigo 2, criam-se dois descendentes trocando o material genético de dois progenitores selecionados. As subárvores p'_1 e p'_2 de cada progenitor são selecionadas aleatoriamente utilizando a função

`RandomNode`$(\cdot)$ e substituídas aplicando a função `Replace` (p_1, p'_1, p'_2), a qual substitui a subárvore p'_1 em p_1 pela subárvore p'_2, resultando em dois indivíduos (possivelmente) diferentes (Figura 2.2b). Nota-se que, dessa forma, indivíduos selecionados várias vezes podem participar da criação de vários programas-filho.

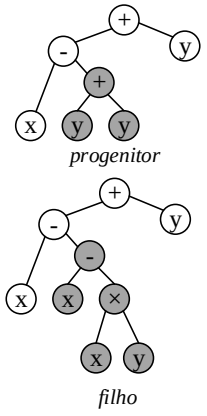
Pseudocódigo 1 Mutação padrão (SM do inglês Standard Mutation)

```
1: procedure SM(p)
2:     p' ← RandomNode(p)
3:     r ← Grow(h, T, F)
4:     return Replace(p, p', r)
```

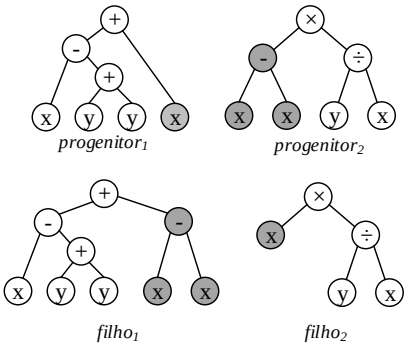
Pseudocódigo 2 Cruzamento padrão (SX do inglês Standard Crossover)

```
1: procedure SX(p1, p2)
2:     p'1 ← RandomNode(p1)
3:     p'2 ← RandomNode(p2)
4:     off1 ← Replace(p1, p'1, p'2)
5:     off2 ← Replace(p2, p'2, p'1)
6:     return off1, off2
```

A execução dos operadores em GP normalmente é exclusiva e sua aplicação na população de indivíduos tem comportamento probabilístico. Geralmente, o operador de cruzamento é aplicado com a maior probabilidade, sendo a taxa de aplicação muitas vezes 90% ou superior. O operador de mutação tem taxas de aplicação menores, tipicamente com valores próximos a 1 %. Quando as taxas de cruzamento e de mutação se somam a um valor menor que 100 %, um operador chamado de reprodução também é usado, com uma taxa de $1 - p$ sendo p o valor da soma das taxas de mutação e cruzamento. A reprodução envolve simplesmente a inserção de uma cópia do indivíduo na próxima geração.



(a) Mutação de um programa.



(b) Cruzamento entre dois programas.

Figura 2.2 – Operadores genéticos no contexto de programação genética.

2.2.6 Bloating

Ao longo do processo evolutivo da população, notou-se o crescimento descontrolado e ilimitado do número de nós de seus indivíduos, geralmente não apresentando melhorias no valor de *fitness* da população. Esse processo foi nomeado na literatura como

fenômeno de *bloating* (LUKE; PANAIT, 2006; POLI, 2003; POLI; LANGDON; MCPHEE, 2008).

O crescimento do tamanho dos indivíduos é prejudicial ao processo evolutivo, tendo como principais prejuízos: uso excessivo de tempo computacional e processamento; baixa capacidade de generalização; e convergência do processo evolutivo em mínimos locais. As principais formas de controle do fenômeno estão divididas em três grupos: métodos limitadores de tamanho de indivíduos, operadores genéticos *anti-bloating* e operadores de seleção *anti-bloating*, apresentados em Poli, Langdon e McPhee (2008).

Os métodos limitadores de tamanho de indivíduos inserem limitantes na profundidade dos indivíduos gerados pelos operadores genéticos. Apresentado em Koza (1992), um filho gerado por uma operação genética, cruzamento ou mutação, só é inserido na próxima geração se tiver uma profundidade menor que um limitante pré-definido. Caso contrário, ele é descartado e um dos progenitores é escolhido para compor a próxima geração. Assim, indivíduos com maior probabilidade de violar a restrição de profundidade tendem a ser copiados com maior frequência. Ou seja, a população tenderá a ser preenchida com indivíduos com profundidade próxima ao limite definido previamente.

Entre os operadores genéticos *anti-bloating* cabe destacar os operadores *size fair crossover*, *size fair mutation* (LANGDON; POLI, 2002; CRAWFORD-MARKS; SPECTOR, 2002) e *shrink mutation* (ANGELINE, 1996). Os operadores *size fair* definem, a partir do comprimento de uma subárvore aleatória de um indivíduo, o tamanho máximo da subárvore candidata não efetuando, assim, trocas que aumentam o tamanho do indivíduo. O operador *Shrink mutation* escolhe aleatoriamente uma subárvore de um indivíduo e a substitui por um valor aleatório do conjunto terminal.

Operadores de seleção *anti-bloating* são divididos em três classes: métodos *tarpeian*, métodos *parsimony pressure* e métodos multiobjetivo. O método *tarpeian* anula o valor de *fitness* de indivíduos que tenham comprimentos acima da média da população, ganhando assim tempo de processamento, uma vez que a função de cálculo de *fitness* não é acionada. O *parsimony pressure* altera as probabilidades de seleção, subtraindo um valor baseado no tamanho de cada indivíduo de seu valor de *fitness*. Assim, a probabilidade de eles estarem na próxima geração se torna menor. Por fim, os métodos multiobjetivo fazem uso dos vários algoritmos de otimização de múltiplos objetivos para descobrir soluções não-dominadas, também chamadas de soluções eficientes. Esse método será detalhado na seção seguinte.

2.3 Programação genética multiobjetivo

Em problemas de otimização com um único objetivo, a solução corresponde a um ou mais pontos factíveis, cujos valores levam a um extremo da função-objetivo. Se o

problema tratado for de minimização, sua solução será o conjunto de pontos factíveis que apresentam o menor valor possível da função-objetivo, sem violar as restrições associadas ao problema.

Já para o caso de problemas de otimização multiobjetivo, aqui denominados de MOO (sigla derivada da denominação em inglês *multi-objective optimization*), este conceito de soluções correspondendo a valores extremos das múltiplas funções-objetivo não pode ser diretamente aplicado, uma vez que os objetivos envolvidos podem ser conflitantes entre si. Um exemplo típico em empresas prestadoras de serviço se refere ao conflito existente entre minimizar o custo de um serviço e maximizar a satisfação do cliente. Abre-se mão, portanto, de soluções que correspondem a extremos das funções-objetivo em prol das chamadas soluções eficientes, que são não-dominadas entre si e que representam os melhores compromissos possíveis entre os múltiplos objetivos do problema. A este conjunto de soluções eficientes para um problema multiobjetivo dá-se o nome de conjunto de Pareto (DEB, 2001).

Sem perda de generalidade, os conceitos e definições associados a problemas de otimização multiobjetivo tratados serão formalizados considerando-se um problema de minimização de todos os objetivos envolvidos. Dessa forma, um problema de otimização multiobjetivo pode ser definido como:

$$\begin{aligned} \min \quad & [f_1(\mathbf{x}) \quad f_2(\mathbf{x}) \quad \cdots \quad f_M(\mathbf{x})] \\ \text{s.a.} \quad & g_j(\mathbf{x}) \leq 0, \ j = 1, 2, \ldots, J. \\ & h_k(\mathbf{x}) = 0, \ k = 1, 2 \ldots, K. \\ & \mathbf{x} = [x_1 \quad x_2 \quad \cdots \quad x_n]^T, \ \mathbf{x} \in \Omega \end{aligned} \tag{2.3}$$

onde $f_i(x)$, $i = 1, ..., M$, são as M funções-objetivo do problema, Ω é o domínio do problema, n é o número de variáveis de decisão e existem J restrições de desigualdade e K restrições de igualdade.

A busca do processo de otimização ocorre no espaço de variáveis de decisão (no nosso caso, espaço dos programas computacionais candidatos), mas a análise da qualidade relativa entre as soluções candidatas ocorre no espaço de objetivos, sendo que a Figura 2.3 ilustra o mapeamento que ocorre entre ambos os espaços, supondo dimensão 2 nos dois espaços, para efeito de visualização.

Cabe definir o conceito de dominância. Diz-se que uma dada solução $\mathbf{x}^{(1)}$ domina outra solução $\mathbf{x}^{(2)}$ se $\mathbf{x}^{(1)}$ é melhor ou igual a $\mathbf{x}^{(2)}$ em todos os objetivos do problema e, para pelo menos um dos objetivos, $\mathbf{x}^{(1)}$ é estritamente melhor que $\mathbf{x}^{(2)}$. De uma maneira mais formal, considerando-se a definição adotada na Formulação (2.3), diz-se que $\mathbf{x}^{(1)}$ domina $\mathbf{x}^{(2)}$ (adotaremos a notação $\mathbf{x}^{(1)} \lhd \mathbf{x}^{(2)}$) se e somente se:

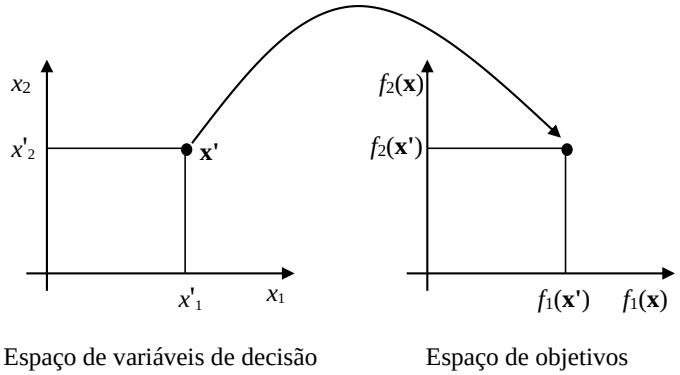


Figura 2.3 – Mapeamento de um ponto no espaço de variáveis de decisão para um ponto no espaço de objetivos.

$$f_i(\mathbf{x}^{(1)}) \leq f_i(\mathbf{x}^{(2)}) \quad \forall i \in \{1, 2, \cdots, M\}$$

e

$$\exists i \in \{1, 2, \cdots, M\} \colon f_i(\mathbf{x}^{(1)}) < f_i(\mathbf{x}^{(2)}).$$

Diz-se, então que uma dada solução $\mathbf{x}$ de um problema multiobjetivo é eficiente, ou seja, pertence à *fronteira* de Pareto, se e somente se não existir nenhuma outra solução $\mathbf{x}'$, factível para o problema em questão, que domine $\mathbf{x}$. A todo este conjunto de soluções eficientes e não-dominadas, dá-se o nome de *conjunto* de Pareto. Formalmente, temos então que o conjunto de Pareto é definido na forma:

$$P^* = \{\mathbf{x} \in \Omega | \neg \exists \mathbf{x}' \in \Omega : \mathbf{x}' \lhd \mathbf{x}\} \tag{2.4}$$

enquanto a fronteira de Pareto é dada por:

$$P_F = \{[f_1(\mathbf{x}) f_2(\mathbf{x}) \cdots f_M(\mathbf{x})] | \mathbf{x} \in P^*\} \tag{2.5}$$

A Figura 2.4 considera dois objetivos a serem minimizados e apresenta o espaço de objetivos. Estão ilustradas a fronteira de Pareto, com destaque para algumas soluções eficientes pertencentes a ela, as quais são não-dominadas entre si, e soluções dominadas por um subconjunto de soluções da fronteira de Pareto.

No contexto de programação genética, é intuitivo considerar $f_1(\mathbf{x})$ como sendo o erro produzido pelo programa frente a um comportamento desejado e considerar $f_2(\mathbf{x})$ como sendo a complexidade (tamanho do código) do programa. Com isso, a fronteira de Pareto é composta por programas candidatos que expressam um compromisso entre dois

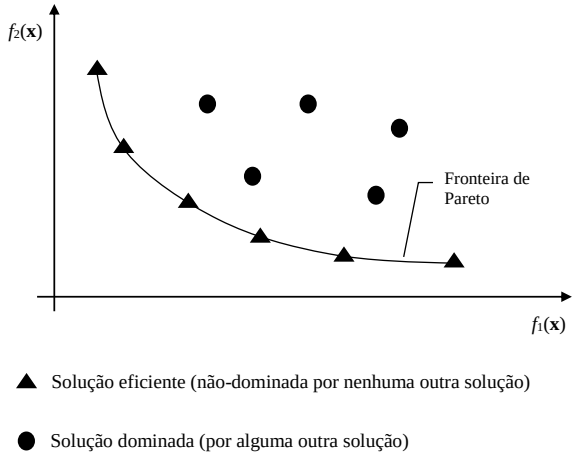


Figura 2.4 – Ilustração do conceito de dominância e de fronteira de Pareto.

objetivos a serem minimizados: erro (inversamente proporcional à acurácia produzida pelo programa) e complexidade (inversamente proporcional à interpretabilidade do programa).

A complexidade das soluções evoluídas é um dos aspectos mais difíceis de controlar em sistemas evolutivos como a GP, onde o tamanho e a forma das soluções evoluídas estão sofrendo variação aleatórias ao longo da evolução. Em alguns casos, por exemplo, o tamanho das soluções desenvolvidas pode crescer rapidamente, sem benefício direto em termos de aptidão.

Existe extensa lista de metodologias multiobjetivo para controle de *bloating* (EKLUND, 2001; BLEULER; BADER; ZITZLER, 2008; BLEULER et al., 2001). Poli, Langdon e McPhee (2008) descrevem as principais metodologias que incluem o comprimento dos indivíduos como um segundo objetivo durante o processo de seleção de indivíduos para compor a próxima geração, visando controlar o crescimento das soluções. Entretanto, essas metodologias não consideram o aspecto semântico, o qual será tratado nesse trabalho. O próximo capítulo aborda um dos ramos atuais de GP, o qual utiliza informações semânticas ao longo do processo evolutivo.

3 Programação Genética Semântica

Neste capítulo, apresentamos a noção da semântica de programas e mostramos como ela pode ser usada na programação genética. Em seguida, descrevemos os principais operadores genéticos que fazem uso desse conceito.

3.1 Definições

Programação Genética Semântica (SGP, do inglês *Semantic Genetic Programming*) é um ramo da GP que utiliza informações semânticas ao longo do processo evolutivo (BEADLE, 2009; NGUYEN, 2011). Ao contrário de outras técnicas de computação evolutiva, o mapeamento entre fenótipo e genótipo em programação genética não é direto, pois a estrutura de um indivíduo (sintaxe) é separada de seu comportamento (semântica) por alguns níveis de abstração.

Basicamente, quando se trabalha com GP, pode-se imaginar a existência de dois espaços diferentes: o primeiro é o espaço genotípico ou sintático, em que indivíduos são representados por estruturas de árvore, enquanto o segundo é o espaço semântico, em que indivíduos são representados por sua semântica, ou seja, seu comportamento de saída em resposta a padrões de entrada. A Figura 3.1 apresenta o espaço genotípico contendo todas as arquiteturas de árvores possíveis, dado o conjunto de símbolos não-terminais $\mathcal{F}$ e o conjunto de símbolos terminais $\mathcal{T}$, e o espaço semântico, considerando a saída obtida para duas instâncias de treinamento. A tarefa do processo evolutivo é obter uma árvore, indivíduo no espaço genotípico, onde sua correspondente semântica é um ponto desejado no espaço semântico (ponto em vermelho).

A Figura 3.1 também mostra que cada árvore no espaço genotípico corresponde um ponto no espaço semântico. Entretanto, essa correspondência nem sempre é biunívoca, uma vez que árvores diferentes podem ter a mesma semântica. A informação semântica pode ser explorada na tentativa de aumentar a localidade da busca, uma vez que a semântica está diretamente relacionada ao fenótipo dos indivíduos. Uma definição formal de semântica, adotada neste trabalho, é apresentada na sequência (Definição 3.2).

As linhas de pesquisa em GP utilizam representação de conjuntos de dados em problemas de aprendizagem supervisionada para descrever semântica de programas (PAWLAK; WIELOCH; KRAWIEC, 2015b; OLIVEIRA, 2016; VANNESCHI; CASTELLI; SILVA, 2014). Formalmente, um conjunto de treinamento é definido como o conjunto de todas as amostras de avaliação (PAWLAK, 2015):

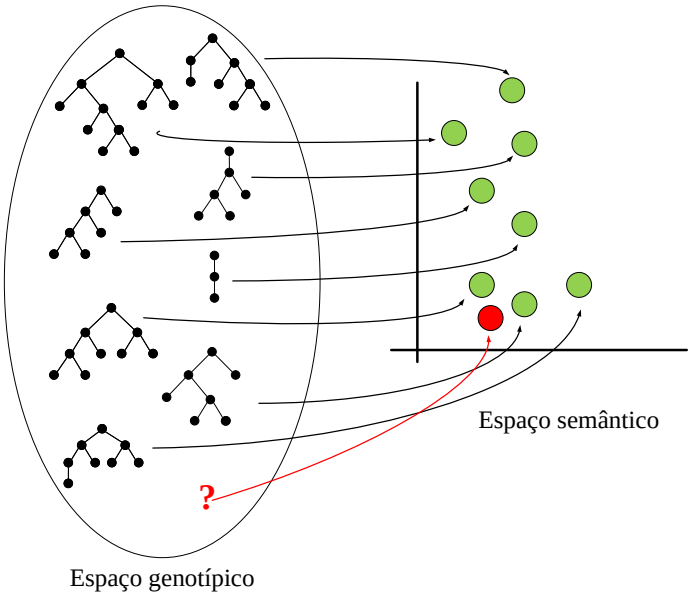


Figura 3.1 – Mapeamento entre o espaço genotípico e o espaço semântico [Adaptado de Schuetze et al. (2016)].

Definição 3.1 *O conjunto de treinamento é formado por n pares entrada-saída na forma* $T = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$*, sendo* $(\mathbf{x}_i,\ y_i) \in \mathcal{X} \times \mathcal{Y}$ $(i = 1, 2, \ldots, n)$*,* $\mathcal{X}$ *o espaço de entrada e* $\mathcal{Y}$ *o espaço de saída.*

Em outras palavras, o conjunto de treinamento é uma amostra de um comportamento desejado para o programa, descrevendo o resultado esperado a partir de padrões de entrada. Da Definição 3.1, temos a definição de semântica:

Definição 3.2 *Seja* $in = (\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\})$ *o conjunto formado pelas entradas associadas ao conjunto de treinamento* T*. A semântica do programa* p *é dada pelo vetor de saídas produzido quando aplicado a in, na forma:* $s(p) = [p(\mathbf{x}_1), p(\mathbf{x}_2)), \ldots\ p(\mathbf{x}_n)]$.

Em suma, a semântica de um programa consiste em uma amostra finita de saídas fornecidas para cada padrão de entrada associado ao conjunto de treinamento. Assim, a semântica geralmente não é uma descrição completa do comportamento de um programa (até porque uma descrição completa acaba se mostrando computacionalmente infactível) e depende da escolha do conjunto de treinamento. Dada a definição de semântica, a distância semântica entre duas árvores ou subárvores geralmente é definida como:

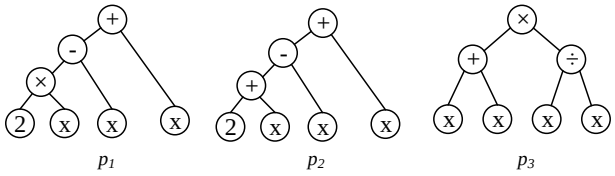
Definição 3.3 *Distância semântica entre duas semânticas é uma função* $d\colon \mathcal{S} \times \mathcal{S} \to \mathbb{R}_{\geq 0}$ *contendo as propriedades:* $d(s_1, s_2) = 0 \iff s_1 = s_2$ *(identidade),* $d(s_1, s_2) = d(s_2, s_1)$ *(simetria),* $d(s_1, s_3) \leq d(s_1, s_2) + d(s_2, s_3)$ *(desigualdade triangular).*

Logo, a distância semântica entre duas árvores (ou subárvores) é uma métrica de distância entre as saídas fornecidas, pelas respectivas árvores, para cada padrão de entrada associado. Neste trabalho, utilizaremos a métrica de distância Euclidiana (NIEDERMEIER; SANDERS, 1996) para calcular a distância semântica entre dois indivíduos.

3.2 Relação entre Sintaxe e Semântica

Embora a semântica determine o que um programa efetivamente faz, a definição tradicional de programação genética (KOZA, 1992; POLI; LANGDON; MCPHEE, 2008; LANGDON; POLI, 2002) implica a manipulação de programas exclusivamente no nível sintático. Com isso, não se tem controle sobre os impactos da variação sintática promovida pelos operadores genéticos sobre o comportamento dos programas gerados. Logo, não fica claro como uma manipulação sintática, que não mensura os seus efeitos semânticos, pode funcionar bem junto a diferentes problemas e em diferentes domínios. Esta é a principal motivação para a inserção de semântica em programação genética, a qual ainda é considerada um tema de pesquisa incipiente e em expansão.

A Figura 3.2 apresenta a relação entre sintaxe e semântica de um indivíduo. Pode-se notar que indivíduos com sintaxes similares podem ter semânticas discrepantes, bem como indivíduos com sintaxes diferentes podem ter semânticas similares.



(a) Sintaxe dos programas.

$\mathbf{x_i}$	$p_1(\mathbf{x_i})$	$p_2(\mathbf{x_i})$	$p_3(\mathbf{x_i})$
1.0	2.0	3.0	2.0
2.0	4.0	4.0	4.0
3.0	6.0	5.0	6.0
4.0	8.0	6.0	8.0

(b) Semântica dos programas.

Figura 3.2 – Relação entre sintaxe e semântica de programas.

A literatura costuma classificar os métodos semânticos da programação genética em duas categorias (VANNESCHI; CASTELLI; SILVA, 2014; NGUYEN et al., 2016): métodos diretos e indiretos. Métodos diretos agem diretamente na semântica dos programas, enquanto métodos indiretos alteram a semântica dos programas através da manipulação da sintaxe dos indivíduos (NGUYEN et al., 2016). Essa taxonomia será utilizada, ao longo deste trabalho, para diferenciar os operadores propostos e compará-los com os principais operadores de cada categoria, que são descritos nos próximos subitens.

3.3 Operadores Semânticos Indiretos

Um conjunto completo de operadores semânticos indiretos foi proposto por Beadle (2009): *Semantically Driven Crossover* (SDX), *Semantically Driven Mutation* (SDM) e *Semantically Driven Initialization* (SDI). A informação semântica foi utilizada para promover a diversidade semântica entre os indivíduos, com um impacto positivo no desempenho em problemas com domínio de valores booleanos.

A SDI examina o comportamento do programa durante o processo de construção da população inicial. Caso for similar ao comportamento de um programa já existente ou tenha um comportamento constante para os padrões de entrada, ele não é inserido na população inicial. Já os operadores genéticos SDM e SDX analisam a equivalência semântica entre os filhos gerados e seus progenitores. Caso os filhos sejam semanticamente semelhantes aos pais, eles são descartados.

Os Pseudocódigos 3 e 4 correspondem aos operadores SDM e SDX, respectivamente, onde na linha 4 é efetuada a operação genética (mutação ou cruzamento) e na linha 6 é verificada a distância semântica entre o filho gerado (p') e o indivíduos progenitor (p para mutação e p_1, p_2 para cruzamento). Se esta distância for maior que um valor ϵ_1 (geralmente definido pelo usuário), a operação genética é efetuada. A busca por subárvores semanticamente distantes é repetida até um valor máximo de tentativas (n_{Trail}), definido pelo usuário, ser atingido.

Pseudocódigo 3 Semantically Driven Mutation (SDM)

```
1: procedure SDM(p)
2:     c ← 0
3:     repeat
4:         p' ← SM(p)
5:         c ← c + 1
6:     until d(p', p) ≥ ε1 ∨ n_Trial < c
7:     return p'
```

Pseudocódigo 4 Semantically Driven Crossover (SDX)

```
1: procedure SDX(p1, p2)
2:     c ← 0
3:     repeat
4:         p'1, p'2 ← SX(p1,p2)
5:         c ← c + 1
6:     until d(p'1, p1) ≥ ε1 ∧ d(p'1, p2) ≥ ε1
       ∧ d(p'2, p1) ≥ ε1 ∧ d(p'2, p2) ≥ ε1 ∨ n_Trial
       < c
7:     return p'1, p'2
```

Análises experimentais (PAWLAK, 2015; BEADLE, 2009) indicaram que os operadores SDM e SDX produziram programas com maior desempenho (melhor valor de *fitness*) quando comparados à utilização da combinação de SX e SM, possuindo uma maior capacidade de generalização quando se considera o desempenho no conjunto de teste. Além disso, os operadores *semantically driven* produzem programas de tamanhos semelhantes aos que foram obtidos pelo processo evolutivo quando utilizados os operadores SX e SM. Por outro lado, o tempo computacional utilizado por essa família de operadores genéticos tende a ser substancialmente maior que aquele associado aos operadores clássicos.

Nguyen (2011) propôs operadores genéticos para o domínio de valor real, chamado *Semantically Aware Crossover* (SAX) e *Semantically Aware Mutation* (SAM). Os operadores foram baseados na verificação da equivalência semântica de subárvores, restringindo o cruzamento para incluir apenas subárvores que não são semanticamente equivalentes.

Os operadores genéticos *semantically aware* (Pseudocódigos 5 e 6) têm comportamento similar aos operadores clássicos (Seção 2.2.5), evitando a troca genética quando a distância semântica entre duas subárvores for menor que um limiar ϵ_1 (geralmente definido pelo usuário). Assim, sua característica primária é promover diversidade semântica na população, evitando substituições de subárvores (espaço genotípico) que tenham comportamentos (elementos do espaço de semânticas) similares.

Pseudocódigo 5 Semantically Aware Mutation (SAM)

```
1: procedure SAM(p)
2:     c ← 0
3:     repeat
4:         p' ← RandomNode(p)
5:         r ← Grow(h, T, F)
6:         c ← c + 1
7:     until d(p', r) ≥ ε1 ∨ n_Trial < c
8:     return Replace(p, p', r)
```

Pseudocódigo 6 Semantically Aware Crossover (SAX)

```
1: procedure SAX(p1, p2)
2:     c ← 0
3:     repeat
4:         p'1 ← RandomNode(p1)
5:         p'2 ← RandomNode(p2)
6:         c ← c + 1
7:     until d(p'1, p'2) ≥ ε1 ∨ n_Trial < c
8:     off1 ← Replace(p1, p'1, p'2)
9:     off2 ← Replace(p2, p'2, p'1)
10:    return off1, off2
```

Uma extensão desses operadores foi proposta por Nguyen (2011), com foco na promoção de mudanças locais na semântica. Eles foram denotados por *Semantic Similarity Based Crossover* (SSX) e *Semantic Similarity Based Mutation* (SSM), permitindo trocas de subárvores caso a distância semântica entre elas esteja entre um valor mínimo e um valor máximo, enfatizando promover trocas locais, ou seja, gerar pouca alteração na semântica dos indivíduos.

O SSM (Pseudocódigo 7) possui as seguintes etapas: (1) selecionar uma subárvore a partir do indivíduo; (2) criar uma subárvore nova; (3) aceitar a subárvore somente se sua distância semântica estiver dentro do intervalo entre ϵ_1 e ϵ_2; caso contrário retornar para a etapa (1) até que o número máximo de tentativas n_{Trail} for atingido. O Pseudocódigo 8 representa o processo para o operador genético de cruzamento, onde é selecionada aleatoriamente uma subárvore de p_1 e uma subárvore de p_2 e é efetuada a troca de material genético, caso a distância semântica entre as duas subárvores esteja dentro do intervalo entre ϵ_1 e ϵ_2.

Experimentos entre os operadores SM, SAM e SSM (NGUYEN; HOAI; NEILL,

Pseudocódigo 7 Semantically Similarity-based Mutation (SSM)

```
1: procedure SSM(p)
2:     c ← 0
3:     repeat
4:         p' ← RandomNode(p)
5:         r ← Grow(h, 𝒯, ℱ)
6:         c ← c + 1
7:     until ε1 ≤ d(p', r) ≤ ε2 ∨ n_Trial < c
8:     return Replace(p, p', r)
```

Pseudocódigo 8 Semantically Similarity-based Crossover (SSX)

```
1: procedure SSX(p1, p2)
2:     c ← 0
3:     repeat
4:         p'1 ← RandomNode(p1)
5:         p'2 ← RandomNode(p2)
6:         c ← c + 1
7:     until ε1 ≤ d(p'1, p'2) ≤ ε2 ∨ n_Trial < c
8:     off1 ← Replace(p1, p'1, p'2)
9:     off2 ← Replace(p2, p'2, p'1)
10:    return off1, off2
```

2009) indicaram que o operador SSM possuiu desempenho superior, seguido pelo operador SM e tendo o SAM com o pior desempenho. O baixo desempenho do operador SAM foi atribuída, por Pawlak (2015), à falta de repetições, assim, não atendendo a condição de distância semântica. Os operadores de cruzamento SAX e SSX superam o operador SX em resultados apresentados em Nguyen, Nguyen e O'Neill (2009) e Nguyen (2011), os quais realizaram análises com diferentes valores de ϵ_2 e ϵ_1, mostrando haver uma alta sensibilidade a esses valores.

3.4 Operadores Semânticos Diretos

O *Semantic Geometric Crossover* (SGX) e a *Semantic Geometric Mutation* (SGM) foram os primeiros operadores semânticos diretos, que agem diretamente na semântica do indivíduo durante a operação genética, para GP propostos por Moraglio, Krawiec e Johnson (2012). Ao contrário dos operadores já listados, no SGX, um descendente é criado com base em uma combinação convexa de seus progenitores. Para problemas de regressão com o domínio de valor real, a operação SGX entre os progenitores p_1 e p_2 consiste em produzir:

$$T_{xo} = p_x \times p_1 + (1 - p_x) \times p_2$$

onde p_x é um valor aleatório no intervalo $[0, 1]$. Essa nova classe de operadores de cruzamento age sobre a sintaxe dos programas pais, com o objetivo de gerar indivíduo cuja semântica está entre a semântica dos progenitores, garantindo que os filhos gerados apresentem um comportamento intermediário. A Figura 3.3 apresenta um exemplo de cruzamento SGX, note que há um aumento no tamanho do indivíduo gerado.

O operador SGM produz um indivíduo:

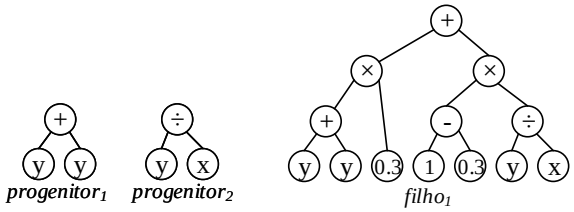


Figura 3.3 – Exemplo do operador *semantic geometric crossover*.

$$T_o = p + ms * (p_1 - p_2)$$

onde p é o indivíduo submetido à mutação, $p_1 - p_2$ são subárvores aleatórias e ms é um valor contínuo, chamado de passo de mutação (exemplificado na Figura 3.4). O objetivo do SGM é transformar a sintaxe do indivíduo visando perturbar localmente a sua semântica. Dado que a semântica dos indivíduos gerados pela mutação pode ser qualquer ponto dentro da região de perturbação, cria-se a possibilidade de gerar um indivíduo cuja semântica seja mais próxima da desejada.

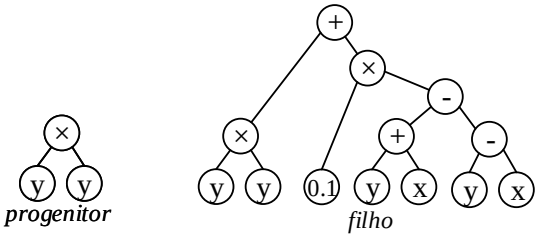


Figura 3.4 – Exemplo do operador *semantic geometric mutation*.

Visto que os operadores SGX e SGM somente adicionam novas subárvores aos indivíduos, promove-se um aumento em seu tamanho ao longo das gerações, aumentando assim o consumo de memória e o custo computacional. Esse crescimento do indivíduo pode ser caracterizado como *bloating*, apresentado em maiores detalhes na Seção 2.2.6.

Outro operador semântico proposto em Krawiec e Pawlak (2013b) e em Krawiec e Pawlak (2012) é denominado *Locally geometric semantic crossover* (LGX), apresentado no Pseudocódigo 9, onde: (1) duas subárvores na forma comum de progenitores são escolhidas aleatoriamente, utilizando as rotinas `CommonRegion` e `RandomNode`; (2) a semântica intermediária é calculada utilizando a operação `Midpoint`; (3) é pesquisado no arquivo de programas (subprogramas) a semântica que mais se aproxima da semântica intermediária por meio da rotina `LibrarySearch`; (4) é realizada a troca da subárvore p_1' por r. Maiores detalhes de sua operação podem ser encontrados em (PAWLAK, 2015; KRAWIEC; PAWLAK, 2012).

Pseudocódigo 9 Locally Geometric Semantic Crossover (LGX)

1: **procedure** LGX(p_1, p_2, *library*)
2: $c \leftarrow$ `CommonRegion`(p_1, p_2)
3: p'_1, $p'_2 \leftarrow$ `RandomNode`(c)
4: $s_m \leftarrow$ `Midpoint`(p'_1, p'_2)
5: $r \leftarrow$ `LibrarySearch`(s_m)
6: $off_1 \leftarrow$ `Replace`(p_1, p'_1, r)
7: $off_2 \leftarrow$ `Replace`(p_2, p'_2, r)
8: **return** off_1, off_2

A operação `Midpoint` utiliza as subárvores p'_1 e p'_2 e calcula o ponto médio entre essas subárvores no espaço semântico, de acordo com o domínio do problema. A equação abaixo apresenta a fórmula para calcular o ponto médio s_m no contexto de regressão:

$$s_m = \frac{s(p'_1) + s(p'_2)}{2}. \tag{3.1}$$

O arquivo de programas pode ser construído a partir de todos os (sub) programas disponíveis na população atual, ou por um conjunto pré-computado de programas, ou, ainda, por uma desses dois. Cabe ressaltar que o número de programas na biblioteca impacta os custos na execução da rotina `LibrarySearch`, podendo ser o principal custo de computação do método.

Aspectos relacionados ao progresso da busca, tamanho da biblioteca de procedimentos, tamanho do indivíduo, capacidade de generalização e complexidade temporal do operador LGX foram apresentados em Krawiec e Pawlak (2013b). Os resultados obtidos demonstram que a utilização do operador LGX pode melhorar a convergência do processo evolutivo, além de reduzir o erro no conjunto de teste. Entretanto, o efeito de *bloating* se mostrou mais frequente e o custo de computação subiu em relação ao encontrado para os operadores SAX, SSX e SX.

A proposição de um conjunto completo de operadores genéticos que adotam conceitos semânticos diretamente na manipulação da população é a estratégia de Pawlak (2015), apresentando os algoritmos *Competent Mutation* (CM) (Pseudocódigo 10) e *Competent Crossover* (CX) (Pseudocódigo 11).

CX consiste em: (1) O ponto médio é calculado pela Equação (3.1) usada no operador LGX, chamada semântica desejada; (2) são selecionados aleatoriamente os pontos de cruzamento (p') ; (3) usa-se o procedimento de *semantic backpropagation* (SBKP) para identificar a semântica necessária na subárvore p' que, quando substituída por uma subárvore r, produzirá um novo descendente que corresponda à semântica desejada; (4) procura-se no arquivo de programas (chamado de *library*) a subárvore r mais relacionada à semântica desejada e a substitui nos pontos de cruzamento.

Pseudocódigo 10 Competent Mutation (CM)

```
1: procedure CM(t, p)
2:     p' ← RandomNode(p)
3:     D ← Sbkp(t, p, p')
4:     D_∅ ← Sbkp(s(p), p, p')
5:     r ← OracleGet(D,D_∅)
6:     return Replace(p, p', r)
```

Pseudocódigo 11 Competent Crossover (CX)

```
1: procedure CX(p_1, p_2)
2:     p' ← RandomNode(p)
3:     a ← Midpoint(p_1, p_2)
4:     D ← Sbkp(a, p_1, p')
5:     D^1_∅ ← Sbkp(s(p_1), p_1, p')
6:     D^2_∅ ← Sbkp(s(p_2), p_1, p')
7:     r ← OracleGet(D,D^1_∅, D^2_∅)
8:     return Replace(p_1, p'_1, r)
```

O objetivo da *semantic backpropagation*, descrita no Pseudocódigo 12, é: dada a semântica desejada D, um programa p e o nó designado r neste programa, determinar a semântica desejada D_r para r tal que, quando um subprograma que possua a semântica D_r substitui a subárvore referente ao nó r, o programa inteiro terá a semântica desejada D como saída.

Um exemplo de execução de *semantic backpropagation* concebido por Pawlak (2015) é apresentado na Figura 3.5. A semântica desejada $D^{(1)} = (\{-2\}, \{0\}, \{0\})$, o programa $p =$ `- (× x 1) (× x (- x 2))` e o nó r em destaque azul são as entradas. A primeira amostra de (-2) é retropropagada em direção ao nó r realizando a primeira operação inversa $3 = 1 - (-2)$, onde 1 é o primeiro componente da semântica do ramo esquerdo c_1. Em seguida, o resultado encontrado, 3, é retropropagado através do nó $\times$ seguindo o sentido da seta em destaque azul, executando a segunda operação inversa $3 = 3 \div 1$. Finalmente, 3 torna-se o primeiro componente da semântica desejada para r. Essas etapas são repetidas para cada amostra da semântica desejada.

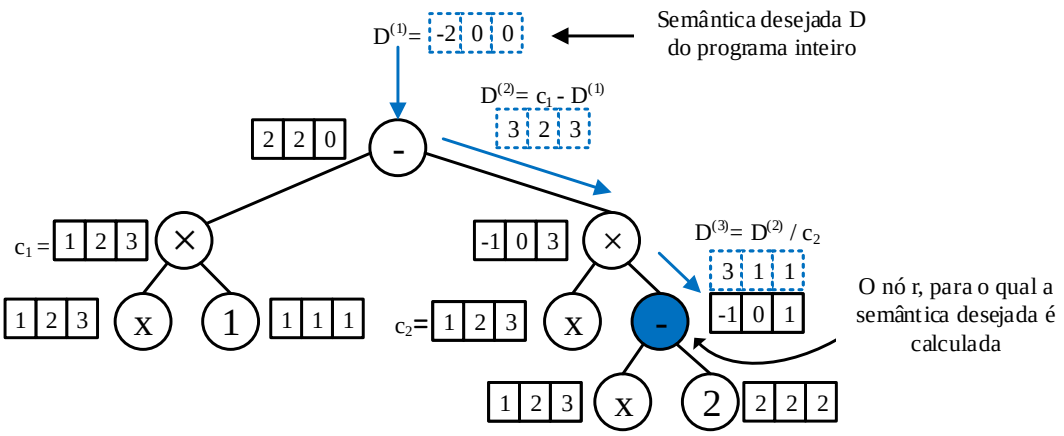


Figura 3.5 – Exemplo de *semantic backpropagation* [Adaptado de Pawlak (2015)].

A rotina `Invert` realiza uma inversão de uma operação a, com relação ao seu k-ésimo argumento e a saída da operação o. A Tabela 3.1 contém a definição para instruções comumente usadas no domínio real.

Pseudocódigo 12 Semantic Backpropagation (SBKP)

```
1: procedure SBKP(D, p, r)
2:     for all D_i ∈ D do
3:         a ← p
4:         while a ≠ r ∧ D_i ≠ ∅ ∧ D_i ≠ ℝ do
5:             k ← Pos(a, r)
6:             D'_i ← ∅
7:             for all o ∈ D_i do
8:                 D'_i ← D'_i ∪ Invert(a, k, o)
9:             D_i ← D'_i
10:            a ← Child(a, r)
11:    return D
```

Tabela 3.1 – Definição do procedimento Invert(a, k, o) para o domínio real

Subprograma a	`Invert`(a, 1, o)	`Invert`(a, 2, o)
$c_1 + c_2$	$o - c_2$	$o - c_1$
$c_1 - c_2$	$o + c_2$	$c_1 - o$
$c_1 \times c_2$	$\begin{cases} o/c_2 & c_2 \neq 0 \\ \mathbb{R} & c_2 = 0 \wedge o = 0 \\ \emptyset & c_2 = 0 \wedge o \neq 0 \end{cases}$	$\begin{cases} o/c_1 & c_1 \neq 0 \\ \mathbb{R} & c_1 = 0 \wedge o = 0 \\ \emptyset & c_1 = 0 \wedge o \neq 0 \end{cases}$
c_1/c_2	$\begin{cases} o \times c_2 & c_2 \neq \pm\infty \\ \mathbb{R} & c_2 = \pm\infty \wedge o = 0 \\ \emptyset & c_2 = \pm\infty \wedge o \neq 0 \end{cases}$	$\begin{cases} c_1/o & c_1 \neq 0 \\ \mathbb{R} & c_1 = 0 \wedge o = 0 \\ \emptyset & c_1 = 0 \wedge o \neq 0 \end{cases}$
$exp(c_1)$	$\begin{cases} \log o & o > 0 \\ \emptyset & \text{caso contrário} \end{cases}$	—
$\log\lvert c_1\rvert$	$\{-e^o, e^o\}$	—
$\sin(c_1)$	$\begin{cases} \{\arcsin o - 2\pi, \arcsin o\} & \lvert o\rvert \leq 1 \\ \emptyset & \text{caso contrário} \end{cases}$	—
$\cos(c_1)$	$\begin{cases} \{\arccos o - 2\pi, \arccos o\} & \lvert o\rvert \leq 1 \\ \emptyset & \text{caso contrário} \end{cases}$	—

A metodologia de *semantic backpropagation* também foi usada no operador de mutação CM. Neste operador, a semântica desejada é o conjunto de saída do conjunto de treinamento, sendo que a metodologia de *semantic backpropagation* novamente é aplicada para identificar a semântica da subárvore que, quando substituída por D, produzirá um novo filho que corresponda à semântica desejada para a aproximação do conjunto de saída. A busca de uma subárvore que tenha a semântica D é realizada pelo procedimento `OracleGet`, que acrescenta outras buscas ao operador `LibrarySearch` presente em LGX, realizando buscas em constantes e evitando semânticas já existentes $D^1_\emptyset$, $D^2_\emptyset$.

Análises de desempenho, capacidade de generalização, custo computacional e tamanho dos indivíduos gerados para os operadores SGX e LGX presentes em Pawlak, Wieloch e Krawiec (2015a) confirmaram o crescimento acentuado no tamanho dos indivíduos, o alto desempenho para o conjunto de treinamento e a baixa capacidade de generalização do operador SGX, sendo superado pelo operador LGX nos quesitos de capacidade de generalização e tamanho dos indivíduos gerados. Análises em Pawlak (2015) demonstram que os operadores competentes possuem melhor convergência e são menos afetados pelo efeito de *bloating* quando comparados aos operadores SGX e LGX. O operador CM possui um desempenho superior ao SGM, conforme demonstrado em Pawlak (2015). Entretanto, os procedimentos de retro-propagação e pesquisa no arquivo influenciam negativamente no custo computacional. Outro fator negativo, apresentado pelo autor, é o fato de o operador não ser utilizável se a semântica desejada do problema não for definida.

No próximo capítulo, apresentaremos três novas família de operadores, contendo operadores de mutação e cruzamento. A primeira família se enquadra em operadores indiretos e com característica de localidade. A segunda é uma extensão da primeira família, utilizando aspectos semânticos e multiobjetivo para que se controle o crescimento do programa e a robustez de seu resultado. Para a terceira família, foram utilizados os conceitos presentes nos operadores *Competent Mutation* e *Competent Crossover*, assim como conceitos de otimização multiobjetivo, se configurando num método direto.

4 Propostas de operadores genéticos semânticos

Este capítulo apresenta três novas famílias de operadores genéticos, propondo uma nova maneira de construir o arquivo de programas para uso no contexto de cruzamento. O conceito de lista restrita de candidatos (RCL, do inglês *Restricted Candidate List*) foi abordado em uma das etapas dos operadores, podendo alterar seu comportamento. Por último, são utilizados aspectos semânticos e multiobjetivo para que se controle o crescimento do programa.

4.1 Construção do arquivo de programas

A utilização de um arquivo de programas faz parte da proposta de diversos operadores genéticos (PAWLAK, 2015; KRAWIEC; PAWLAK, 2013a; PAWLAK; KRAWIEC, 2017; PAWLAK; WIELOCH; KRAWIEC, 2015b). Ele tem como principal propósito fornecer subárvores que possam substituir um ramo de um indivíduo escolhido aleatoriamente, modificando a população durante o processo de evolução.

A fonte de itens que compõem um arquivo de programas pode ser arbitrária. Dois tipos de construção de arquivo de programas foram abordados em Pawlak, Wieloch e Krawiec (2015b): arquivo dinâmico, construído a partir de todos os indivíduos da população, e arquivo estático, criado a partir de programas pré-computados antes da inicialização do processo evolutivo. O arquivo estático tem como vantagem executar métodos de inicialização somente uma única vez, não requerendo processamento computacional ao longo do processo evolutivo. Já arquivos dinâmicos são construídos a cada geração, requisitando um custo computacional extra ao longo das gerações. Como vantagem, arquivos dinâmicos são capazes de se adaptar encontrando blocos promissores, enquanto arquivos estáticos são imutáveis, podendo conter itens com baixo desempenho, que não contribuem para a busca.

A diversidade semântica de um arquivo está diretamente relacionada ao número de itens contidos, bem como ao tamanho dos itens. Segundo Pawlak, Wieloch e Krawiec (2015b), programas com tamanhos pequenos tendem a exibir distâncias semânticas pequenas, enquanto programas maiores geralmente apresentam uma distribuição semântica maior. Assim, um arquivo com um grande número de itens de diferentes tamanhos é desejado. Entretanto, a pesquisa exaustiva se torna mais custosa computacionalmente, assim como a avaliação dos itens.

Duas formas de construção de arquivo foram empregadas pelos novos operadores propostos neste trabalho. A primeira forma de construção foi para a concepção de um

arquivo estático, empregado a eliminação de itens com semânticas idênticas e alterando a quantidade de nós dos itens. O arquivo elaborado a partir primeira forma de construção foi utilizada nos operadores de mutação e se mantendo estático ao longo de todo processo evolutivo. Para operadores de cruzamento, o arquivo é construído pelo seguinte processo (Pseudocódigo 13): dado um indivíduo p, o arquivo associado a ele terá todas as subárvores existentes nesse indivíduo.

Pseudocódigo 13 Construção do arquivo de programas a partir de um indivíduo

```
1: procedure BUILDLIB(p)
2:     for node ∈ p do
3:         item ← getSubtree(node)
4:         library ← library ∪ item
5:     return library
```

Por exemplo (Figura 4.1), o primeiro item do arquivo será o próprio indivíduo, o segundo item (em verde) será o primeiro ramo, o terceiro item conterá o segundo ramo, repetindo até que se tenham todas as subárvores existentes. Essa forma de construção de arquivo tem como principal função preservar parte da sintaxe do indivíduo após a troca de material genético, mantendo subárvores promissoras que possam contribuir para a semântica do indivíduo.

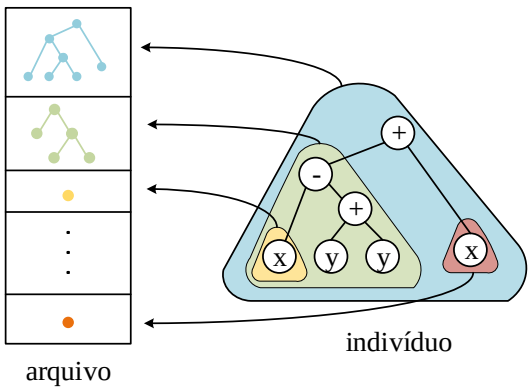


Figura 4.1 – Processo de construção de arquivo de programas a partir de um indivíduo.

4.2 Greedy randomized aware similarity

A primeira família de operadores a ser proposta é denominada *Greedy Randomized Aware Similarity*, e se baseia na heurística construtiva do algoritmo *Greedy Randomized Adaptive Search Procedure* (GRASP), descrito em Feo e Resende (1995), Burke e Kendall (2013). A principal característica desta família de operadores é a criação de uma RCL formada pelos melhores elementos. No contexto de SGP, são os itens que possuem as

menores distâncias semânticas, sendo o aspecto "ganancioso" (*greedy*) do operador. O aspecto probabilístico da heurística é a escolha aleatória do elemento a ser utilizado na substituição do operador. Além disso, os operadores propostos fazem uso do arquivo de subárvores, descrito em Krawiec e Pawlak (2013a), Krawiec e Pawlak (2013b), Pawlak e Krawiec (2017), Pawlak (2015) e abordado na seção anterior. O arquivo de programas será insumo para a criação da RCL, que será utilizada para gerar diversidade nos indivíduos.

O algoritmo de mutação proposto tem cinco etapas, apresentadas no Pseudocódigo 14: (1) selecionar uma subárvore p' a partir do indivíduo p; (2) calcular uma distância semântica das subárvores contidas em *library* com a subárvore p', armazenando a maior distância (c^{max}) e a menor distância (c^{min}); (3) construir a RCL, onde são empregados c^{max}, c^{min} e um parâmetro, $\alpha \in [0, 1]$, que ajusta o comportamento desejado para o operador; (4) escolher aleatoriamente um item da RCL; (5) substituir a subárvore p' pelo item escolhido.
.

Pseudocódigo 14 Greedy randomized aware similarity mutation (GRASM)

1: **procedure** GRASM(p, *library*, α)
2: $p' \leftarrow$ `RandomNode`(p)
3: $c^{min} \leftarrow \min\{d(p', i) \mid i \in library\}$
4: $c^{max} \leftarrow \max\{d(p', i) \mid i \in library\}$
5: $RCL \leftarrow \{\ i \in library \mid d(p', i) \leq c^{min} + \alpha(c^{max} - c^{min})\}$
6: $r \leftarrow$ `RandomItem`(RCL)
7: **return** `Replace`(p, p', r)

Essas etapas estão exemplificadas na Figura 4.2, dado um progenitor e um arquivo composto por 6 itens. É selecionada aleatoriamente uma subárvore do progenitor (em destaque) e calculada a distância semântica entre subárvores contidas no arquivo e a subárvore do progenitor. Após, é concebida a RCL (itens em destaque do arquivo) e realizada a troca da subárvore do progenitor por um item aleatório da RCL.

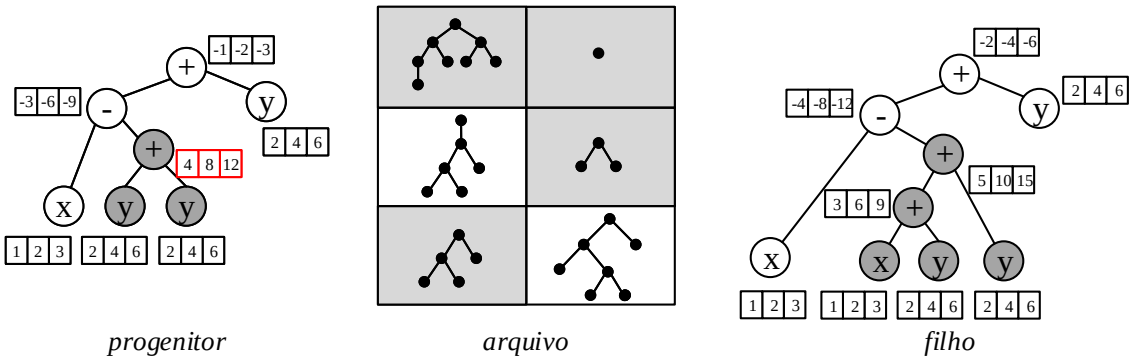


Figura 4.2 – Exemplo do operador genético GRASM.

O Pseudocódigo 15 apresenta a versão de cruzamento da família de operadores *Greedy Randomized Aware Similarity*, que consiste em, dados os progenitores p_1 e p_2 e

o parâmetro de controle α, construir o arquivo de itens l_2 a partir do indivíduo p_2. Em seguida, é utilizado o operador de mutação anteriormente apresentado, adotando como entrada o arquivo l_2 e α para ajuste de comportamento do operador. Assim, as informações sintáticas de p_1 e p_2 farão parte do filho gerado, sendo esta geração guiada por informações semânticas ao longo do processo.

O controle do comportamento dos operadores é feito pelo parâmetro de controle α. Quanto mais perto de 1, maior o número de candidatos na RCL e o operador tende a exibir uma característica aleatória, com comportamento semelhante àquele exibido pelos operadores SX e SM (Seção 2.2.5). Para valores próximos de zero, a RCL terá uma quantidade menor de elementos, tendo subárvores com comportamento semântico semelhante (menor distância semântica) ao nó escolhido.

Pseudocódigo 15 Greedy randomized aware similarity crossover (GRASX)

```
1: procedure GRASX(p1, p2, α)
2:     l1 ← BuildLibrary(p1)
3:     l2 ← BuildLibrary(p2)
4:     p1 ← GRASM(p1, l2, α)
5:     p2 ← GRASM(p2, l1, α)
6:     return p1, p2
```

A família de *Greedy Randomized aware Similarity* é categorizada como sendo composta por operadores indiretos com características locais, enfatizando trocas locais, ou seja, gerar pouca alteração na semântica dos indivíduos a partir de substituições nas arquiteturas das subárvores (espaço genotípico) que tenham comportamento (semântica) individual muito discrepante da semântica já existente. Diferentemente de seu concorrente, *Semantic Similarity Based*, os operadores propostos possuem somente um parâmetro de controle. Já seu concorrente contém limitantes inferior e superior. Além disso, os limitantes dos operadores SSM e SSX variam entre zero e infinito, contrapondo ao parâmetro de controle dos operadores propostos, que varia entre zero e um, facilitando seu ajuste. Por fim, a família de operadores apresentada serviu como base para as vertentes multiobjetivo a serem descritas nas seções seguintes.

4.3 Indivíduos não dominados

Na Seção 2.3, foram apresentados os principais conceitos de otimização multiobjetivo, os quais vêm sendo utilizados para controle do problema de *bloating* (Seção 2.2.6). Entretanto, a maior parte dessas estratégias (BLEULER; BADER; ZITZLER, 2008; FERARIU; PATELLI, 2009) controla o fenômeno de crescimento dos indivíduos a partir da utilização de conceitos multiobjetivo nos operadores de seleção, e não empregam aspectos semânticos durante o processo de evolução. Além disso, existem vertentes da programação

genética semântica que tendem a produzir programas mais complexos, portanto, menos interpretáveis, o que é indesejado, pois prejudica a capacidade de generalização e o custo computacional (Seção 3.3).

Combinando as estratégias multiobjetivo e a RCL, torna-se viável a proposição de uma lista de candidatos eficientes (não-dominados), conduzindo a uma nova estratégia de controle de *bloating*. Explora-se, então, o conceito de não-dominância entre as subárvores que compõem um arquivo de programas disponível para os operadores genéticos. O processo consiste em conceber a RCL a partir de itens que são não-dominados para um determinado conjunto de objetivos. Os objetivos para o critério de não-dominância podem ser a distância semântica da subárvore na lista de candidatos com a subárvore aleatória p', comprimento ou parcimônia. A Figura 4.3 exemplifica o procedimento de construção da RCL. Dado o arquivo de itens, com tamanhos e semânticas distintas, e a semântica desejada D, é computada a distância semântica entre os itens do arquivo e a semântica desejada D. Em seguida, são avaliados os itens não-dominados para os objetivos de minimizar a distância semântica e o comprimento do item. O gráfico à direita expõe a disposição dos itens no espaço de objetivos, com destaque em vermelho para os itens não-dominados (itens 2 e 3). Esses itens irão compor a RCL e serão utilizados futuramente pelos operadores propostos.

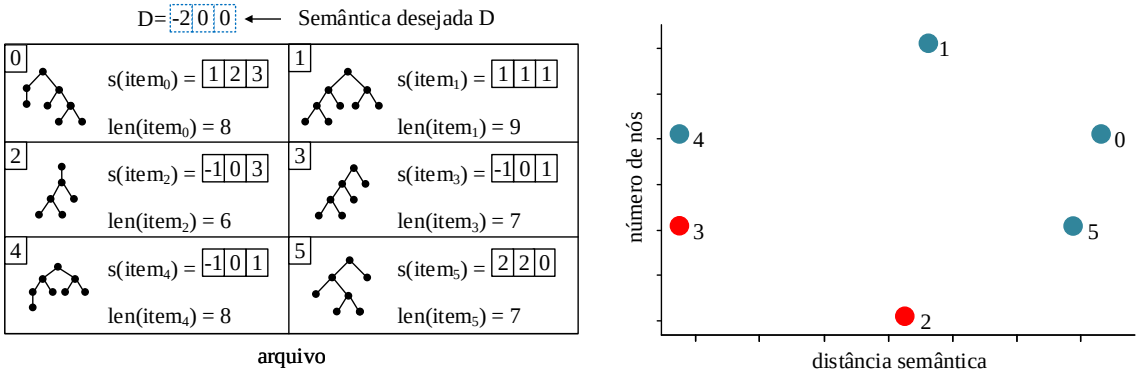


Figura 4.3 – Ilustração da construção da RCL a partir dos itens não-dominados do arquivo de programas. $s(.)$ é a saída do programa e $len(.)$ é o seu comprimento.

Podemos fazer uso dessa estratégia em diferentes operadores genéticos existentes (LGX, CM, CX). Substituindo a estratégia de busca no arquivo de programas pela estratégia multiobjetivo, ao invés de selecionar o item mais próximo semanticamente da semântica desejada, são selecionados os itens não-dominados, podendo adicionar itens mais distantes semanticamente, mas com um comprimento menor. Por essa razão, são apresentados novos operadores semânticos que podem ser incorporados ao processo evolutivo para reduzir a taxa de crescimento da complexidade dos programas. Por fim, vamos mostrar que evitar sub-domínios dominados em operadores genéticos é benéfico para o processo evolutivo, reduzindo o tamanho do programa final gerado.

4.4 Multi-objective randomized similarity

A primeira família de operadores que faz uso de conceitos multiobjetivo é denominada *Multi-objective randomized similarity mutation* (MORSM) para mutação e *Multi-objective randomized similarity crossover* (MORSX) para cruzamento, sendo uma extensão multiobjetivo dos operadores genéticos GRASM e GRASX, respectivamente. Os operadores propostos possuem estratégias diferentes de seleção de subárvores para RCL, contendo os itens não-dominados para os objetivos de números de nós e distância semântica. A ideia desta família de operadores é evitar o efeito de *bloating*, abordado em Langdon e Poli (2002) e O'Neill et al. (2010), e ter modificações locais em indivíduos ao longo das gerações.

Os detalhes do MORSM estão presentes no Pseudocódigo 16. O procedimento consiste em: (1) selecionar uma subárvore p' a partir do indivíduo p; (2) calcular uma distância semântica das subárvores contidas em *library* com a subárvore p'; (3) computar o número de nós dos itens presentes em *library*; (4) obter os itens não-dominados utilizando a distância semântica e o número de nós previamente calculados; (5) escolher aleatoriamente um item da RCL; (6) substituir a subárvore p' pelo item escolhido.

Pseudocódigo 16 Multi-objective randomized similarity mutation (MORSM)

1: **procedure** MORSM(p, $library$)
2: $p' \leftarrow$ `RandomNode`(p)
3: $o_{dist} \leftarrow d(s(p'), d(i))$ for all $i \in library$
4: $o_{len} \leftarrow len(i)$ for all $i \in library$
5: RCL $\leftarrow$ `nonDominated`($library$, o_{dist}, o_{len})
6: $r \leftarrow$ `RandomItem`(RCL)
7: **return** `Replace`(p, p', r)

Essas etapas estão exemplificadas na Figura 4.4, dado um progenitor e um arquivo composto por 6 itens. É selecionada aleatoriamente uma subárvore do progenitor (em destaque) e calculada a distância semântica entre subárvores contidas no arquivo e a subárvore do progenitor. Após, é concebida a RCL a partir de itens não-dominados, tendo como objetivos a distância semântica e o número de nós previamente calculados, e realizada a troca da subárvore do progenitor por um item aleatório da RCL.

O MORSX promove a troca de subárvores semanticamente equivalentes ou com tamanhos pequenos no cruzamento. Em outras palavras, é efetuada a troca de um ramo aleatório do progenitor p_1 por um dos itens da RCL, obtido a partir dos não-dominados do arquivo de items l_2 concebido a partir do progenitor p_2. Os detalhes do algoritmo MORSX são fornecidos no Pseudocódigo 17.

Diferentemente dos operadores GRASM e GRASX, os operadores multiobjetivo não possuem parâmetros de controle, sendo seus comportamentos controlados a partir

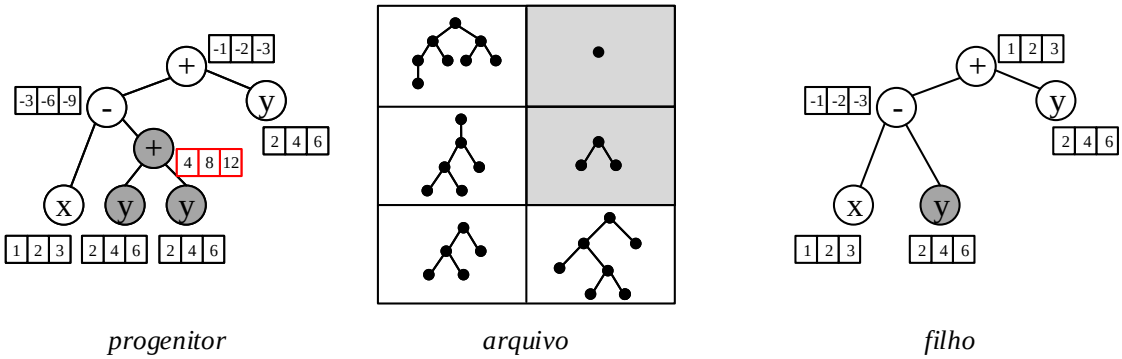


Figura 4.4 – Exemplo do operador genético MORSM.

Pseudocódigo 17 Multi-objective similarity crossover (MORSX)

```
1: procedure MORSX(p1, p2)
2:     l1 ← BuildLibrary(p1)
3:     l2 ← BuildLibrary(p2)
4:     p1 ← MORSM(p1, l2)
5:     p2 ← MORSM(p2, l1)
6:     return p1, p2
```

dos objetivos escolhidos, quantidade de objetivos e do arquivo de itens utilizado. Nota-se que o número de objetivos influenciará na quantidade de itens não-dominados que compõem a RCL. Os itens presentes no arquivo tendem a apresentar diversidade semântica e diversidade no tamanho.

4.5 Multi-objective desired operator

No contexto dos métodos diretos, propõe-se uma extensão do operador *Competent Mutation* (CM) (Seção 3.4), visando controlar o crescimento de indivíduos ao longo das gerações, sem prejudicar o seu desempenho. Este operador consiste em selecionar uma subárvore p' do indivíduo p, calculando a semântica desejada d_s no ramo selecionado usando o procedimento `Sbkp` (Seção 3.4). Então, a distância entre a semântica desejada e a semântica das subárvores contidas no arquivo de programas é calculada. Computadas todas as distâncias, o procedimento `nonDominated` é empregado para compor a RCL com subárvores não-dominadas, com o objetivo de minimizar a distância e o comprimento da subárvore. Finalmente, uma subárvore da RCL é escolhida aleatoriamente e inserida no lugar de p', conforme apresentado no Pseudocódigo 18.

Essas etapas estão exemplificadas na Figura 4.5, dado um progenitor e um arquivo composto por 6 itens. É selecionada aleatoriamente uma subárvore do progenitor (em destaque), calculada a semântica desejada no ramo selecionado (em azul) e a distância semântica entre subárvores contidas no arquivo e a semântica desejada. Após, é concebida a RCL a partir de itens não-dominados, tendo como objetivos a distância semântica e o

Pseudocódigo 18 Multi-objective desired mutation (MODM)

```
1: procedure MODM(p, t, library)
2:     p' ← RandomNode(p)
3:     D ← Sbkp(t, p, p')
4:     o_dist ← d(D, d(i)) for all i ∈ library
5:     o_len ← len(i) for all i ∈ library
6:     RCL ← nonDominated(library, o_dist, o_len)
7:     r ← RandomItem(RCL)
8:     return Replace(p, p', r)
```

número de nós previamente calculados de cada item presente no arquivo, e realizada a troca da subárvore do progenitor por um item aleatório da RCL.

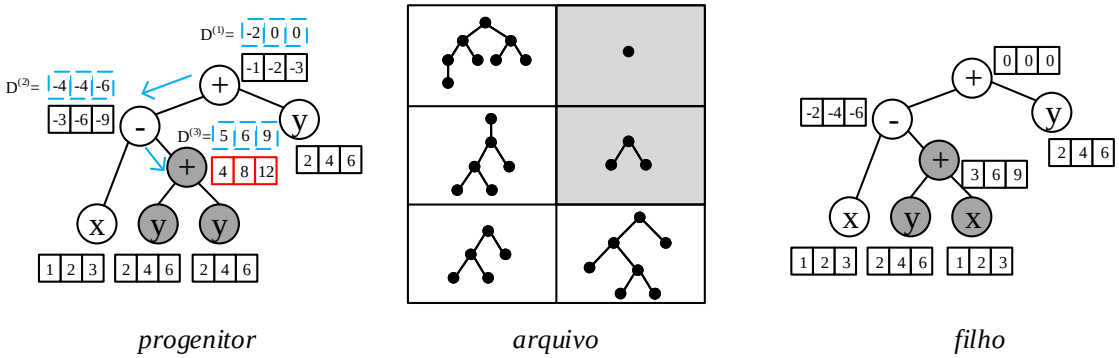


Figura 4.5 – Exemplo do operador genético MODM.

O cruzamento explorando conceitos diretos é inspirado no cruzamento aproximadamente geométrico, *Competent Crossover*. Este cruzamento, chamado de *Multi-objective desired crossover* (MODX), possui as tarefas (Pseudocódigo 19): (1) selecionar um nó p'_1 do indivíduo progenitor p_1; (2) calcular o ponto médio entre a semântica dos progenitores p_1 e p_2, usando a o procedimento `Midpoint`, análogo ao procedimento presente no operador LGX; (3) executar as etapas presentes no operador de mutação MODM, tendo como argumento, um nó p'_1, a semântica calculada a e o arquivo de programas *library*, podendo ser um arquivo dinâmico ou um arquivo estático, retornando o filho p_1.

Pseudocódigo 19 Multi-objective desired crossover (MODX)

```
1: procedure MODX(p_1, p_2, library)
2:     p'_1 ← RandomNode(p_1)
3:     a ← Midpoint(p_1, p_2)
4:     p_1 ← MODM(p_1, a, library)
5:     p_2 ← MODM(p_2, a, library)
6:     return p_1, p_2
```

Cabe ressaltar que os operadores multiobjetivo propostos aqui diferem dos operadores propostos em Galván-López et al. (2016) e Galván-López, Vázquez-Mendoza e Trujillo (2017), que realizaram duas adaptações simples com o NSGA-II, incorporando

semântica na escolha de indivíduos para cruzamento e modificações do *crowding distance* do NSGA- II para que a distância seja de natureza semântica, aplicando em conjuntos de dados de classificação binária desbalanceados e não visando o controle de crescimento das soluções.

Além disso, a metodologia proposta aplica o aspecto multiobjetivo somente nos operadores genéticos, diferindo dos métodos tradicionais multiobjetivo para controle de *bloating*, amplamente apresentados por Poli, Langdon e McPhee (2008), que empregam aspectos multiobjetivo em operadores de seleção e não utilizam informações semânticas ao longo do processo evolutivo. Por fim, os operadores propostos visam, principalmente, aplicar informações semânticas para guiar a busca de soluções (conforme apresentado na Capítulo 3) e incluir conceitos multiobjetivo com foco em atenuar o efeito de *bloating*, sendo essa a principal contribuição do trabalho. Na próxima seção, apresentaremos uma análise dos algoritmos e comparações com operadores semânticos existentes.

5 Análise experimental

Este capítulo é dedicado à análise dos algoritmos e comparação com operadores semânticos existentes. Uma visão geral da metodologia experimental é apresentada. O capítulo contém quatro experimentos separados, observando análise de desempenho, capacidade de generalização, tamanho dos programas e custo computacional.

5.1 Configuração dos experimentos

Para investigar o desempenho dos algoritmos propostos, eles foram aplicados a nove problemas de regressão. Para promover uma extensa análise comparativa, esses mesmos problemas foram tratados por técnicas semânticas de GP presentes na literatura. Entre os problemas testados, existem seis problemas de *benchmark* GP de McDermott et al. (2012) e dois problemas de valor real considerados em Oliveira (2016). As descrições detalhadas dos problemas são apresentadas na Tabela 5.1.

Existem quatro problemas com um parâmetro de entrada, sendo funções trigonométricas, exponencial-trigonométrica, racional e logarítmica. Para problemas com dois parâmetros de entrada, foram empregadas três funções, uma trigonométrica, uma polinomial e a última exponencial racional, os quais irão avaliar a capacidade dos algoritmos em conceber equações similares. A base *Airfoil Self-noise*, concebida pela Nasa em 1989, consiste de dados referentes a aerofólios de tamanhos diferentes, com diferentes ângulos e velocidades em um túnel de vento, tendo como saída o nível de ruído gerado (medido em decibéis), avaliando desempenho dos algoritmos em bases ruidosas. Por fim, a base *Concrete Comp Str* possui como saída dados de resistência à compressão do concreto quando são consideradas oito variáveis de entrada, que servirão para avaliar o comportamento dos algoritmos para um maior número de variáveis.

Os parâmetros utilizados para configurar o processo evolutivo são retratados na Tabela 5.2. As funções de divisão e logaritmo foram modificadas para sua versão protegida, de modo que o resultado é definido como 1 sempre que temos uma divisão por zero ou um argumento não positivo na função logaritmo. Essas funções protegidas são adequadamente combinadas com as outras funções aritméticas usuais, geralmente adotadas na programação genética (PAWLAK; WIELOCH; KRAWIEC, 2015a; NGUYEN et al., 2016). A seleção por torneio de três indivíduos é usada para selecionar um subconjunto de indivíduos como progenitores da próxima geração. A função de aptidão e a distância semântica foram o erro quadrático médio e a norma Euclideana, respectivamente, para o conjunto de dados de treino. Para a *library* de subárvores candidatas, 200 indivíduos foram construídos aleatoriamente e impedidos de exibir comportamentos semânticos semelhantes.

Tabela 5.1 – Problemas de regressão utilizados para avaliação dos operadores. U[a, b, c] são c amostras aleatórias de uma distribuição uniforme entre a e b e E[a, b, c] são amostras igualmente espaçadas entre a e b com distância c entre as amostras. Os conjuntos de teste e treinamento são independentes

Nome	Definição	Conjunto de treinamento	Conjunto de teste
a) Problemas de regressão Benchmarks de GP			
Keijzer-1	$0.3\ x\ \sin(2\pi x)$	E[-1, 1, 0.01]	E[-1, 1, 0.001]
Keijzer-4	$x^3 e^{-x} \cos(x) \sin(x) (\sin^2(x) \cos(x)\ -1)$	E[0, 10, 0.05]	E[0.05, 10.05, 0.05]
Keijzer-6	$\sum_i^x \frac{1}{i}$	E[1, 50, 1]	E[1, 120, 1]
Keijzer-7	$\ln(x)$	E[1, 100, 1]	E[1, 100, 0.1]
Nguyen-9	$\sin(x_1) + \sin(x_2^2)$	U[-1, 1, 100]	U[-1, 1, 100]
Nguyen-12	x_1^4 - $x_1^3 + \frac{x_2^2}{2}$ - x_2	U[-1, 1, 100]	U[-1, 1, 100]
Vladislavleva-1	$\frac{\exp -(x_1-1)^2}{1.2+(x_2-2.5)^2}$	E[-1, 1, 100]	E[0, 1, 100]

Nome	Números de atributos	Conjunto de treinamento	Conjunto de teste
b) Problemas de regressão da UCI			
Airfoil Self-noise	5	752	751
Concrete Comp Str	8	515	515

Finalmente, a profundidade máxima da árvore de qualquer programa foi definida como 13, de modo que os programas que violam esse limite são descartados.

Tabela 5.2 – Valores de parâmetros do processo evolutivo.

Parâmetro	Valor
Número de indivíduos	200
Número de gerações	100
Seleção	Torneio
Tamanho do torneio	3
Profundidade máxima inicial	6
Profundidade máxima	13
Função de aptidão	erro quadrático médio
Distância semântica	Norma L_2
Tamanho da *Library*	200
Conjunto de funções	$+$, $-$, $\times$, $\div$, sin, cos, exp, log

A configuração experimental foi concebida para completar três propósitos principais: (1) Estabelecer uma análise comparativa envolvendo as três famílias de operadores propostos e operadores semânticos relevantes da literatura, sendo divididos nos grupos indiretos e diretos. (2) Determinar o impacto dos operadores multiobjetivo sobre a evolução do tamanho do programa. (3) Obter o comportamento da evolução ao longo das gerações, em termos de desempenho, tamanho do programa e tempo computacional. Os experimentos foram divididos em duas análises: aplicação dos operadores de mutação de maneira isolada

e aplicação dos operadores de cruzamento de maneira isolada.

Com o objetivo de comparar a eficácia dos operadores já disponíveis na literatura com os propostos neste trabalho, 30 execuções independentes foram realizadas para cada operador, coletando informações de média, mediana e fornecendo intervalos de confiança de 95%. Para análises de comportamento do processo evolutivo, dados do melhor indivíduo ao longo das gerações foram apresentados em um gráfico.

Os algoritmos foram comparados em um ambiente Intel Core i5-6400, 16 GB de RAM, usando como linguagem padrão *Python* 3.6 e o pacote de algoritmos evolutivos *Distributed Evolutionary Algorithms in Python* (DEAP).

5.2 Análise comparativa dos operadores de mutação

Esta seção se destina a comparar o desempenho dos operadores de mutação propostos com outros existentes na literatura. Foram comparadas estatísticas de treinamento e generalização do conjunto de testes, tamanho dos programas produzidos e tempo de processamento, considerando como propostas alternativas os operadores SDM, SAM e SSM, para métodos indiretos, e CM, para métodos diretos. Os parâmetros internos foram mantidos fixos para todas as execuções e todos os problemas de *benchmark*.

5.2.1 Avaliação do desempenho em termos de aptidão

A Figura 5.1 apresenta a média do melhor indivíduo ao longo das geração e, nas Tabelas 5.3 e 5.4, do melhor *fitness* ao final do processo evolutivo para métodos indiretos e diretos, respectivamente.

Entre os métodos diretos, o operador multiobjetivo apresentou um desempenho pior que a de seu concorrente mono-objetivo, em 4 dos 9 problemas. A queda de desempenho se justifica, pois durante o processo evolutivo não é analisada somente a similaridade com a semântica desejada no processo de busca na *library*, mas sim a combinação entre similaridade semântica e comprimento da subárvore. Cabe ressaltar que a probabilidade de encontrar um programa complexo (em termos de número de nós da solução) que tenha uma alta aptidão é maior do que a de encontrar um programa curto com o mesmo comportamento (LANGDON; POLI, 1998).

Para os métodos indiretos, nota-se um comportamento de convergência similar em todos os operadores, com exceção do operador multiobjetivo, que possui uma evolução menos acelerada. Entretanto, em 5 dos 9 problemas, o melhor desempenho foi alcançado pelo operador MORSM, ficando empatado com SDM. Pode-se afirmar que a mutação genética MORSM acrescenta maior diversidade nas modificações locais, o que indica que essa variabilidade semântica foi benéfica na exploração do espaço de busca.

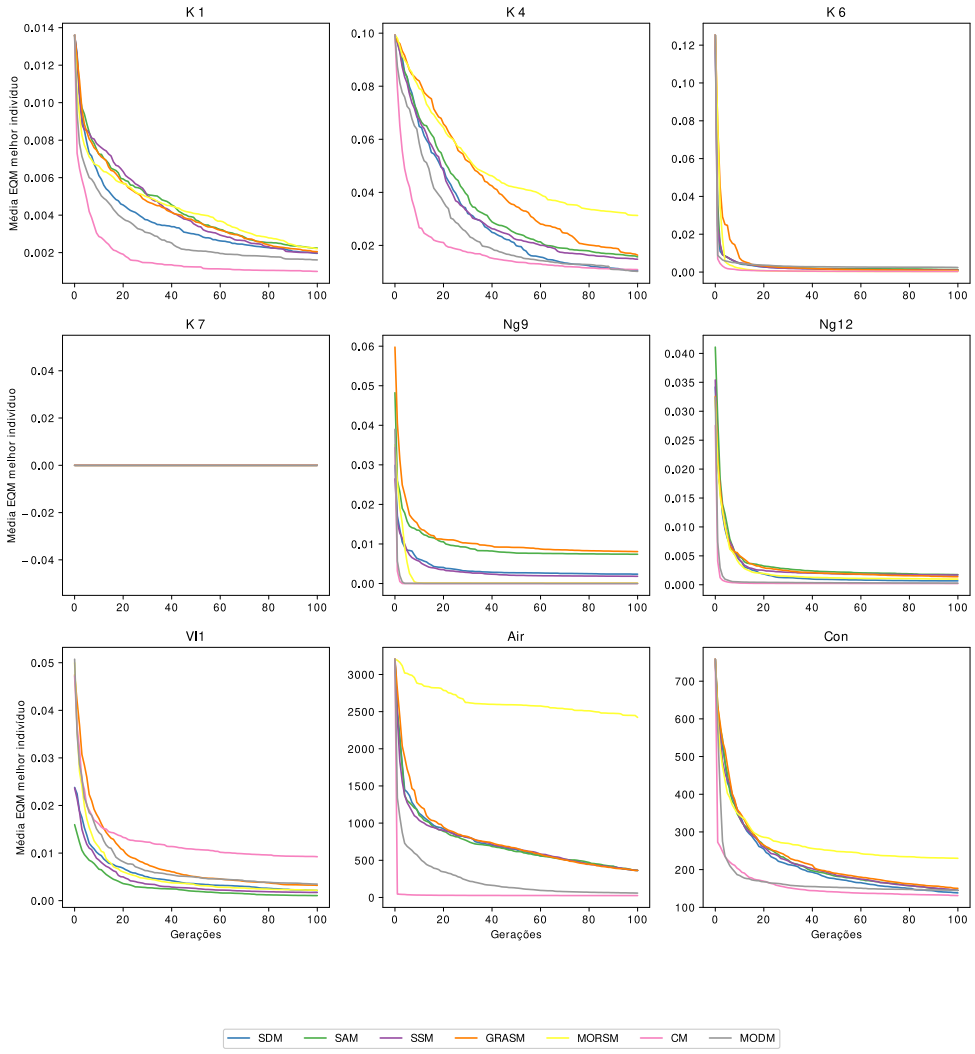


Figura 5.1 – Média da melhor aptidão ao longo das gerações no caso de operadores de mutação.

Tabela 5.3 – Média e 95% do intervalo de confiança para o desempenho no conjunto de dados de treinamento no caso de operadores de mutação indiretos.

	SDM		SAM		SSM		GRASM		MORSM	
kj1	**0.002**	± 0.001	**0.002**	± 0.001	**0.002**	± 0.001	**0.002**	± 0.001	**0.002**	± 0.001
kj4	**0.010**	± 0.006	0.016	± 0.017	0.015	± 0.016	0.016	± 0.018	0.031	± 0.025
kj6	0.001	± 0.001	0.001	± 0.001	0.001	± 0.001	0.001	± 0.001	**0.000**	± 0.0
kj7	**0.000**	± 0.0	**0.000**	± 0.0	**0.000**	± 0.0	**0.000**	± 0.0	**0.000**	± 0.0
ng9	0.002	± 0.004	0.007	± 0.01	0.002	± 0.003	0.008	± 0.01	**0.000**	± 0.001
ng12	**0.001**	± 0.001	0.002	± 0.001	0.002	± 0.001	**0.001**	± 0.001	**0.001**	± 0.002
vl1	0.002	± 0.001	**0.001**	± 0.001	0.002	± 0.001	0.003	± 0.002	0.002	± 0.001
air	364.9	± 215.9	**361.8**	± 182.7	363.1	± 154.8	363.6	± 215.8	2421	± 1574
con	**138.3**	± 28.1	145.1	± 22.0	145.0	± 23.1	150.2	± 30.5	230.1	± 81.3
Rank	2.55		2.94		2.83		3.55		3.11	

Tabela 5.4 – Média e 95% do intervalo de confiança para o desempenho no conjunto de dados de treinamento no caso de operadores de mutação diretos.

	CM		MODM	
kj1	**0.001**	± 0.002	0.002	± 0.001
kj4	0.011	± 0.008	**0.010**	± 0.008
kj6	**0.000**	± 0.0	0.002	± 0.003
kj7	**0.000**	± 0.0	**0.000**	± 0.0
ng9	**0.000**	± 0.0	**0.000**	± 0.0
ng12	**0.000**	± 0.0	**0.000**	± 0.0
vl1	0.009	± 0.007	**0.003**	± 0.005
air	**24.844**	± 8.247	58.784	± 29.108
con	**131.14**	± 62.85	145.246	± 20.98
Rank	1.38		1.61	

5.2.2 Capacidade de generalização

Para uma segunda análise de desempenho, é avaliada a capacidade de generalização dos operadores, onde são aplicados os dados do conjunto de teste como entrada, para o indivíduo mais apto do processo evolutivo. A Figura 5.2 apresenta a mediana da aptidão alcançado pelo melhor programa junto aos dados de teste ao longo do processo evolutivo, e as Tabelas 5.5 e 5.6 mostram os mesmos dados do melhor programa do processo evolutivo para o conjunto de treinamento. Foi utilizada a mediana para melhor análise devido à presença de *overfitting*.

Analisando a mediana das 30 execuções contidas na Tabela 5.5, constatou-se, em geral, que o SDM possui a melhor capacidade de generalização, desempenhou melhor em 7 vezes dos 9 problemas. Observa-se que o operador de inserção de conceitos multiobjetivo pode influenciar negativamente na capacidade de generalização, como presente no problema Kj6. Além disso, os operadores SAM, SSM e GRASM obtiveram desempenho similares entre eles, possuindo também convergências similares.

Observando a Tabela 5.6, destaca-se o operador MODM, sendo que sua capacidade de generalização foi superior ao de seu concorrente CM em 4 dos 9 problemas, demostrando sua competitividade perante o outro operador. Por fim, destaca-se a capacidade de generalização dos métodos diretos, os quais atingiram desempenhos superiores aos métodos indiretos, possuindo, também, maiores velocidades de convergência.

5.2.3 Tamanho do programa

A média e o intervalo de confiança de 95% do número de nós no melhor programa estão contidos nas Tabelas 5.7 e 5.8, e seu comportamento ao longo do processo evolutivo está presente na Figura 5.3.

Nota-se que, para ambos os métodos, diretos e indiretos, as extensões multi-

Tabela 5.5 – Mediana e 95% do intervalo de confiança para o desempenho no conjunto de dados de teste no caso de operadores de mutação indiretos.

	SDM			SAM			SSM			GRASM			MORSM	
kj1	0.002 ≤	**0.002**	≤ 0.003	0.002 ≤	0.003	≤ 0.003	0.002 ≤	**0.002**	≤ 0.004	0.002 ≤	0.003	≤ 0.009	0.001 ≤	**0.002**	≤ 0.003
kj4	0.005 ≤	**0.010**	≤ 0.013	0.004 ≤	0.011	≤ 0.018	0.006 ≤	0.011	≤ 0.018	0.006 ≤	0.012	≤ 0.016	0.012 ≤	0.024	≤ 0.042
kj6	0.001 ≤	**0.004**	≤ 0.006	0.001 ≤	**0.004**	≤ 0.007	0.003 ≤	**0.004**	≤ 0.005	0.002 ≤	**0.004**	≤ 0.076	0.13 ≤	1.689	≤ 2.684
kj7	0.0 ≤	**0.000**	≤ 0.0	0.0 ≤	**0.000**	≤ 0.0	0.0 ≤	**0.000**	≤ 0.0	0.0 ≤	**0.000**	≤ 0.0	0.0 ≤	**0.000**	≤ 0.0
ng9	0.001 ≤	0.004	≤ 0.011	0.0 ≤	0.003	≤ 0.008	0.001 ≤	0.002	≤ 0.006	0.002 ≤	0.010	≤ 0.021	0.0 ≤	**0.000**	≤ 0.0
ng12	0.001 ≤	**0.001**	≤ 0.002	0.002 ≤	0.002	≤ 0.003	0.002 ≤	0.002	≤ 0.003	0.001 ≤	**0.001**	≤ 0.003	0.0 ≤	**0.001**	≤ 0.001
vl1	0.002 ≤	**0.003**	≤ 0.005	0.01 ≤	0.020	≤ 0.069	0.004 ≤	0.005	≤ 0.01	0.004 ≤	0.008	≤ 0.015	0.017 ≤	0.033	≤ 0.101
air	216.3 ≤	338.3	≤ 496.0	219.4 ≤	380.5	≤ 506.2	225.0 ≤	**334.1**	≤ 486.1	172.8 ≤	336.8	≤ 463.3	1280 ≤	1471	≤ 3759
con	118.2 ≤	**128.6**	≤ 145.0	119.1 ≤	132.0	≤ 147.2	124.3 ≤	137.5	≤ 147.6	116.1 ≤	138.2	≤ 167.0	154.8 ≤	204.3	≤ 263.5
Rank		2.16			3.33			2.50			3.33			3.66	

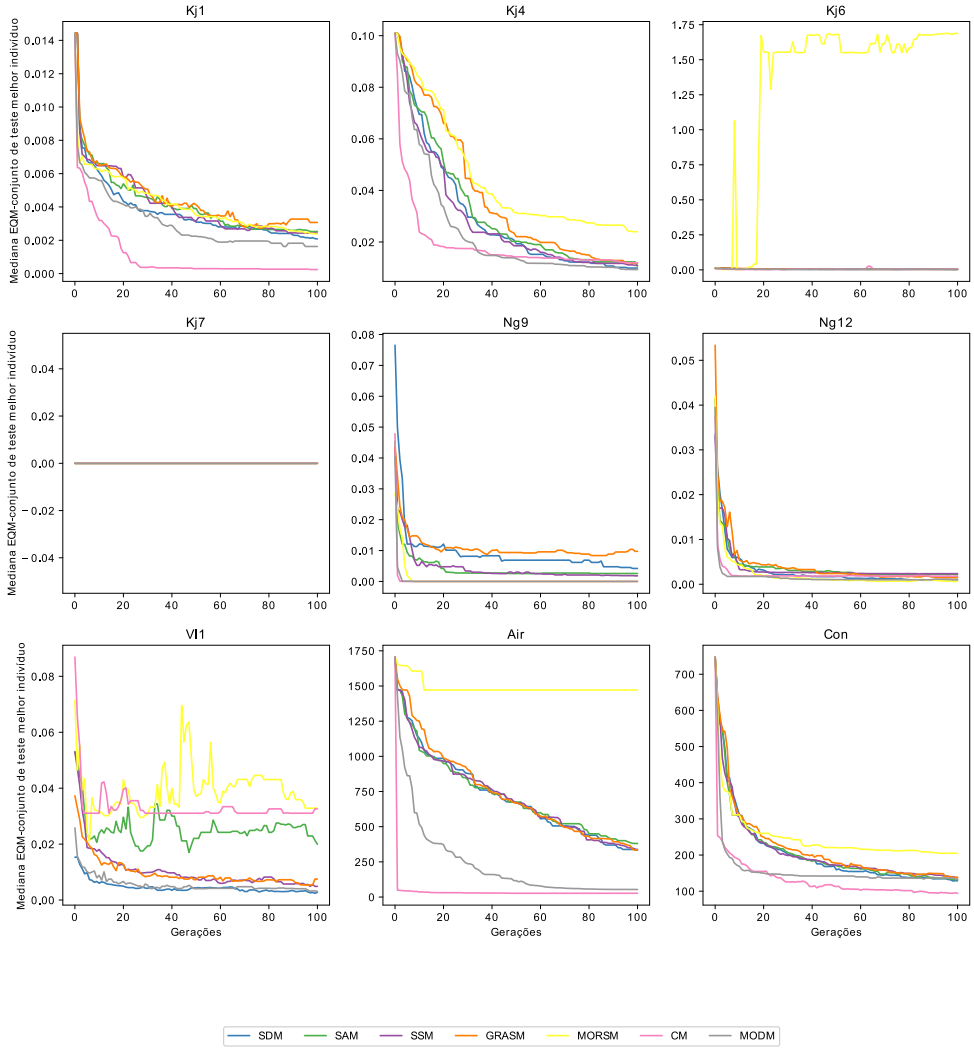


Figura 5.2 – Mediana do desempenho do melhor indivíduo para o conjunto de dados de teste ao longo das gerações no caso de operadores de mutação.

Tabela 5.6 – Mediana e 95% do intervalo de confiança para o desempenho no conjunto de dados de teste no caso de operadores de mutação indiretos.

	CM			MODM		
kj1	0.0 ≤	**0.000**	≤ 0.002	0.001 ≤	0.002	≤ 0.003
kj4	0.002 ≤	0.012	≤ 0.015	0.003 ≤	**0.009**	≤ 0.013
kj6	0.002 ≤	0.007	≤ 19.461	0.001 ≤	**0.004**	≤ 0.056
kj7	0.0 ≤	**0.000**	≤ 0.0	0.0 ≤	**0.000**	≤ 0.0
ng9	0.0 ≤	**0.000**	≤ 0.0	0.0 ≤	**0.000**	≤ 0.0
ng12	0.001 ≤	0.002	≤ 0.003	0.001 ≤	**0.001**	≤ 0.002
vl1	0.023 ≤	0.033	≤ 0.062	0.002 ≤	**0.003**	≤ 0.005
air	23.294 ≤	**26.658**	≤ 33.64	42.502 ≤	53.362	≤ 71.625
con	81.671 ≤	**94.155**	≤ 152.451	119.554 ≤	133.470	≤ 142.949
Rank		1.55			1.44	

objetivo se mostraram válidas, com uma diminuição significativa no tamanho da solução, quando comparado aos demais métodos, evidenciando, assim, os benefícios da abordagem multiobjetivo para o controle de crescimento da solução. O operador GRASM atingiu o maior crescimento das soluções, com um tamanho médio maior que os demais operadores, além de possuir a maior amplitude de tamanho da população. Isso se deve à estratégia de construção do arquivo utilizada no processo, a qual somente observou aspectos semânticos.

Destaca-se o comportamento do operador CM, que superou o operador multiobjetivo em 3 dos 9 problemas. Isso se deve à metodologia de busca na *library* apresentada em Pawlak (2015), a qual calcula uma constante que mais se aproxima da semântica desejada, fazendo com que se reduza o tamanho da árvore, mas aumentando seu custo computacional associado à busca da constante.

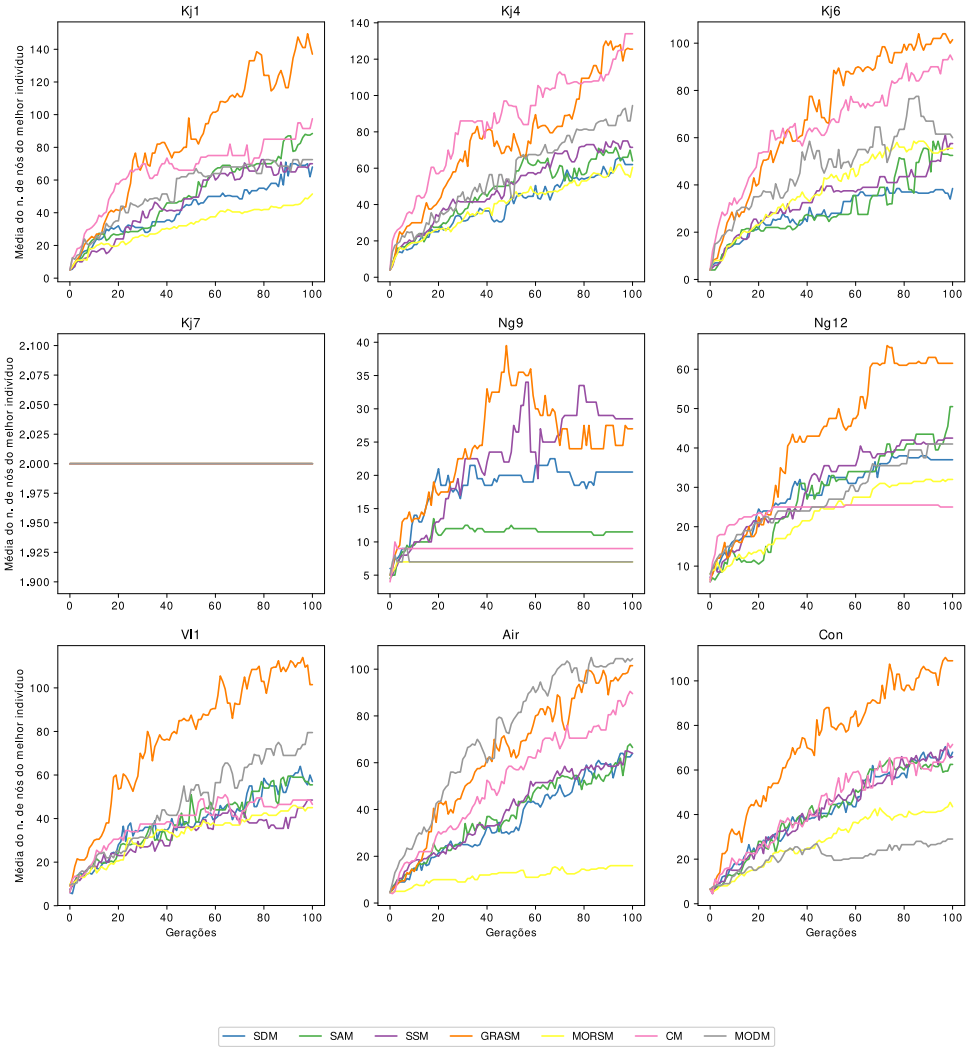


Figura 5.3 – Média do tamanho do melhor indivíduo ao longo das gerações no caso de operadores de mutação.

Tabela 5.7 – Média e 95% do intervalo de confiança para o tamanho do melhor indivíduo no caso de operadores de mutação indiretos.

	SDM		SAM		SSM		GRASM		MORSM	
kj1	64.07	± 25.59	84.60	± 28.42	70.23	± 27.87	136.43	± 58.24	**51.60**	± 13.53
kj4	63.43	± 30.26	72.13	± 31.71	71.43	± 31.97	121.30	± 54.2	**59.10**	± 23.65
kj6	**43.63**	± 27.25	55.73	± 31.28	61.13	± 32.07	105.03	± 51.59	63.73	± 29.91
kj7	**2.00**	± 0.0	**2.00**	± 0.0	**2.00**	± 0.0	**2.00**	± 0.0	**2.00**	± 0.0
ng9	27.43	± 21.05	21.93	± 19.27	35.73	± 25.68	42.07	± 35.94	**7.50**	± 2.69
ng12	40.90	± 23.69	52.83	± 28.36	42.10	± 22.64	69.60	± 37.53	**34.63**	± 13.14
vl1	60.83	± 32.64	56.57	± 26.77	51.60	± 25.16	114.20	± 50.6	**47.23**	± 15.4
air	65.13	± 21.18	61.90	± 25.07	66.37	± 24.07	102.67	± 51.72	**17.77**	± 15.57
con	68.47	± 17.78	64.77	± 23.85	66.93	± 16.14	107.87	± 36.33	**48.0**3	± 25.09
Rank	2.66		2.88		3.11		4.77		1.55	

Tabela 5.8 – Média e 95% do intervalo de confiança para o tamanho do melhor indivíduo no caso de operadores de mutação diretos.

	CM		MODM	
kj1	98.83	± 60.98	**88.10**	± 58.02
kj4	128.17	± 55.59	**93.67**	± 43.12
kj6	93.20	± 34.59	**71.47**	± 45.81
kj7	**2.00**	± 0.0	**2.00**	± 0.0
ng9	9.53	± 2.17	**7.20**	± 0.79
ng12	**31.33**	± 19.42	50.70	± 27.07
vl1	**71.97**	± 64.12	78.07	± 44.94
air	**76.50**	± 40.79	108.93	± 35.92
con	78.73	± 53.24	**36.87**	± 21.45
Rank	1.61		1.38	

5.2.4 Tempo de execução

A Figura 5.4 apresenta a média do tempo total consumido ao longo das gerações; Por sua vez, as Tabelas 5.9 e 5.10 apresentam os mesmos números no final da execução, juntamente com o intervalo de confiança de 95%.

As aplicações multiobjetivo impactaram negativamente no tempo de processamento, com um acréscimo significativo, quando comparado aos demais métodos. Nota-se que o tempo de processamento se manteve constante ao longo das gerações em boa parte dos problemas, enquanto os competidores levaram um tempo computacional proporcional ao número de nós dos indivíduos presentes naquela geração.

O operador SAM atingiu o menor tempo computacional, quando comparado aos demais operadores, em virtude de executar o cálculo da distância semântica apenas uma vez, enquanto seus competidores realizam esse mesmo cálculo diversas vezes. Entre os operadores diretos, o operador CM obteve menor tempo computacional em 5 dos 9 problemas.

Nota-se que o tempo computacional está relacionado com o tamanho dos indivíduos da população, aumentando ao longo das gerações. Entretanto, o controle multiobjetivo não se mostrou benéfico para o tempo de execução. Acredita-se que o controle multiobjetivo possa ter impacto positivo no tempo computacional para processos evolutivos com um maior número de gerações, quando o efeito de *bloating* se torna mais presente.

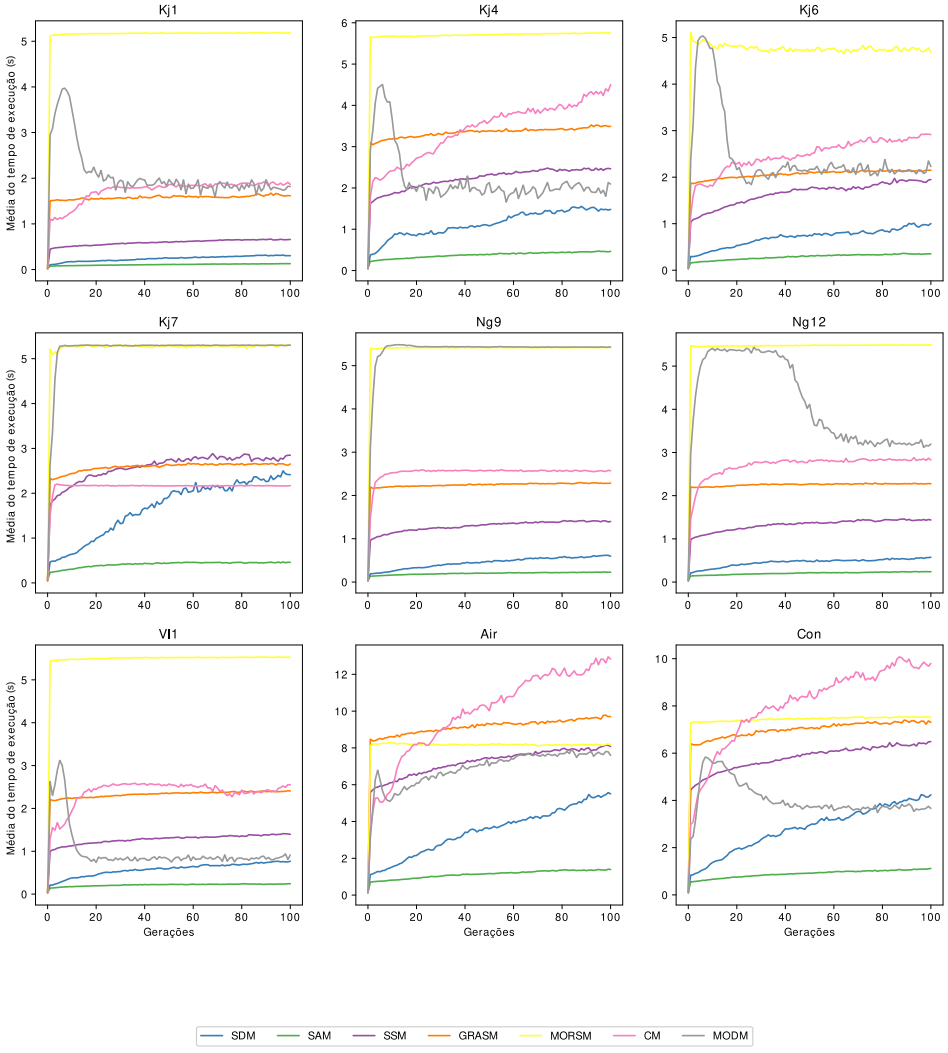


Figura 5.4 – Média do tempo computacional consumido ao longo das gerações no caso de operadores de mutação.

5.3 Análise comparativa dos operadores de cruzamento

Esta seção tem como objetivo comparar as características de operadores de cruzamento. Assim como foi realizado durante a análise dos operadores de mutação,

Tabela 5.9 – Média e 95% do intervalo de confiança para o tempo computacional consumido no caso de operadores de mutação indiretos.

	SDM		SAM		SSM		GRASM		MORSM	
kj1	24.16	± 4.76	**10.94**	± 0.73	60.07	± 3.51	162.64	± 8.56	516.83	± 3.11
kj4	116.76	± 29.17	**38.00**	± 3.7	225.99	± 17.17	339.91	± 14.5	571.05	± 3.45
kj6	76.55	± 22.29	**29.63**	± 3.77	166.87	± 12.31	206.86	± 6.31	477.25	± 16.06
kj7	180.26	± 47.52	**41.87**	± 3.4	263.99	± 25.65	260.08	± 7.09	526.60	± 3.33
ng9	45.71	± 7.6	**20.10**	± 1.88	129.92	± 6.68	225.20	± 4.07	541.55	± 0.49
ng12	49.20	± 16.45	**20.64**	± 1.77	134.88	± 8.73	226.27	± 4.45	547.83	± 1.13
vl1	60.22	± 16.3	**21.96**	± 2.81	129.28	± 5.88	234.41	± 5.78	551.12	± 1.77
air	359.39	± 49.84	**115.07**	± 7.24	730.39	± 31.5	920.30	± 36.12	816.13	± 18.54
con	291.16	± 53.18	**89.31**	± 7.21	588.33	± 39.44	704.16	± 27.06	740.37	± 16.28
Rank	2.00		1.00		3.11		4.00		4.88	

Tabela 5.10 – Média e 95% do intervalo de confiança para o tempo computacional consumido no caso de operadores de mutação diretos.

	CM		MODM	
kj1	**158.38**	± 46.52	224.39	± 99.97
kj4	349.08	± 65.6	**236.99**	± 102.05
kj6	**250.63**	± 48.17	289.57	± 128.53
kj7	**216.60**	± 1.0	523.47	± 2.17
ng9	**248.52**	± 20.45	539.98	± 2.26
ng12	**271.20**	± 20.03	416.93	± 42.74
vl1	219.95	± 61.55	**154.47**	± 125.3
air	1006.17	± 263.44	**681.22**	± 79.16
con	793.62	± 277.42	**419.31**	± 73.3
Rank	1.44		1.55	

foram executados separadamente os operadores CX e LGX (métodos diretos) e SDX, SAX e SSX (métodos indiretos) para compará-los em termos de adequação ao conjunto de treinamento e de teste, tamanho dos programas produzidos e custo computacional associado aos operadores propostos.

5.3.1 Avaliação do desempenho em termos de aptidão

Nas Tabelas 5.11 e 5.12, apresentamos a média e o intervalo de confiança de 95% da melhor aptidão alcançada. A melhor aptidão ao longo das gerações é apresentada na Figura 5.5.

O CX possui a convergência mais acelerada na regressão simbólica, pois em poucas gerações o valor de *fitness* diminui de maneira acentuada e mantém a redução ao longo das gerações. Em contrapartida, o operador MORSX possui a convergência mais retardada, muitas vezes ficando preso em mínimos locais, conforme apresentado no problema Kj1.

O GRASX atingiu a melhor adequação em 6 dos 9 problemas, tendo sido superado pelos operadores SDX, SSX e MORSX nos demais problemas, respectivamente. Devido à maior quantidade de itens no arquivo de programas, criou-se uma perturbação semântica na população suficiente para percorrer espaços promissores, evitando mudanças semânticas abruptas nos indivíduos.

Para os operadores diretos, o desempenho dos operadores LGX e CX é bastante estável em relação ao operador multiobjetivo, que apresentou um desempenho pior que seus concorrentes mono-objetivo em 4 dos 9 problemas. Nota-se que a inserção de conceitos multiobjetivo influenciou negativamente na convergência do operador, deixando-a mais lenta.

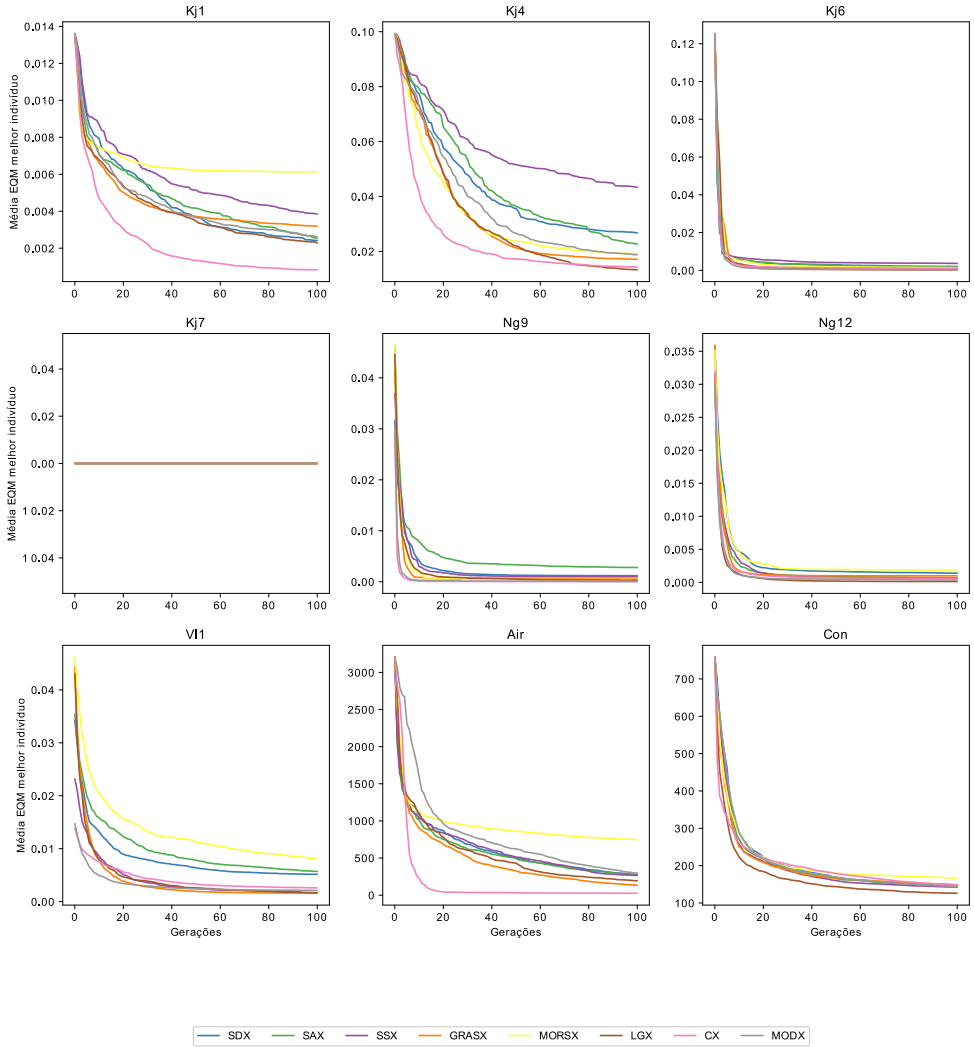


Figura 5.5 – Média da melhor aptidão ao longo das gerações no caso de operadores de cruzamento.

Tabela 5.11 – Média e 95% do intervalo de confiança para o desempenho no conjunto de dados de treinamento no caso de operadores de cruzamento indiretos.

	SDX		SAX		SSX		GRASX		MORSX	
kj1	**0.002**	± 0.002	0.003	± 0.001	0.004	± 0.003	0.003	± 0.002	0.006	± 0.001
kj4	0.027	± 0.029	0.023	± 0.024	0.043	± 0.037	**0.017**	± 0.021	0.019	± 0.017
kj6	0.002	± 0.002	0.002	± 0.002	0.004	± 0.003	**0.000**	± 0.001	0.001	± 0.001
kj7	**0.000**	± 0.0	**0.000**	± 0.0	**0.000**	± 0.0	**0.000**	± 0.0	**0.000**	± 0.0
ng9	0.001	± 0.003	0.003	± 0.007	0.001	± 0.002	0.001	± 0.002	**0.000**	± 0.001
ng12	**0.001**	± 0.001	**0.001**	± 0.0	**0.001**	± 0.001	**0.001**	± 0.0	0.002	± 0.001
vl1	0.005	± 0.004	0.006	± 0.007	**0.002**	± 0.001	**0.002**	± 0.001	0.008	± 0.009
air	279.1	± 305.0	269.0	± 247.6	266.0	± 299.7	**134.1**	± 117.7	749.3	± 210.7
con	148.5	± 29.6	146.1	± 26.5	**142.5**	± 31.0	149.0	± 32.6	166.7	± 48.9
Rank	3.00		3.16		3.00		2.16		3.66	

Tabela 5.12 – Média e 95% do intervalo de confiança para o desempenho no conjunto de dados de treinamento no caso de operadores de cruzamento diretos.

	LGX		CX		MODX	
kj1	0.002	± 0.002	**0.001**	± 0.001	0.003	± 0.003
kj4	**0.013**	± 0.019	0.014	± 0.017	0.019	± 0.026
kj6	**0.000**	± 0.0	0.001	± 0.002	**0.000**	± 0.0
kj7	**0.000**	± 0.0	**0.000**	± 0.0	**0.000**	± 0.0
ng9	**0.000**	± 0.0	**0.000**	± 0.0	**0.000**	± 0.0
ng12	**0.000**	± 0.0	0.001	± 0.0	**0.000**	± 0.0
vl1	**0.002**	± 0.001	0.003	± 0.002	**0.002**	± 0.003
air	192.563	± 161.091	**25.052**	± 5.573	295.971	± 155.006
con	**126.074**	± 24.159	147.607	± 31.806	143.666	± 31.39
Rank	1.61		2.22		2.16	

5.3.2 Capacidade de generalização

A Figura 5.6 mostra a mediana do desempenho do melhor indivíduo para o conjunto de dados de teste ao longo das gerações e as Tabelas 5.13 e 5.14 apresentam, respectivamente, a mediana e o intervalo de confiança de 95% para o desempenho do melhor indivíduo, considerando o conjunto de dado de testes para os métodos indiretos.

Em 7 de 9 problemas, a melhor adequação ao conjunto de teste é obtida pelo GRASX. Também não foram notados sinais de sobreajuste de nenhum operador dentre SDX, SAX e SSX. O operador MORSX obteve o pior desempenho de generalização, possuindo convergência lenta, assim como verificado para o conjunto do treinamento.

Observa-se no problema Kj6 que os operadores diretos mono-objetivo apresentaram dificuldades na generalização do problema, possuindo um acréscimo no nível de aptidão ao longo das gerações, enquanto a vertente multiobjetivo possuiu uma convergência suave e decrescente. Verifica-se, no problema *air* uma diferença significativa na capacidade de generalização do operador CX, devido ao procedimento `OracleGet`, que considera

constantes numéricas durante o processo de busca da semântica mais próxima à semântica desejada, tornando presente contantes em todos os melhores indivíduos das 30 execuções independentes.

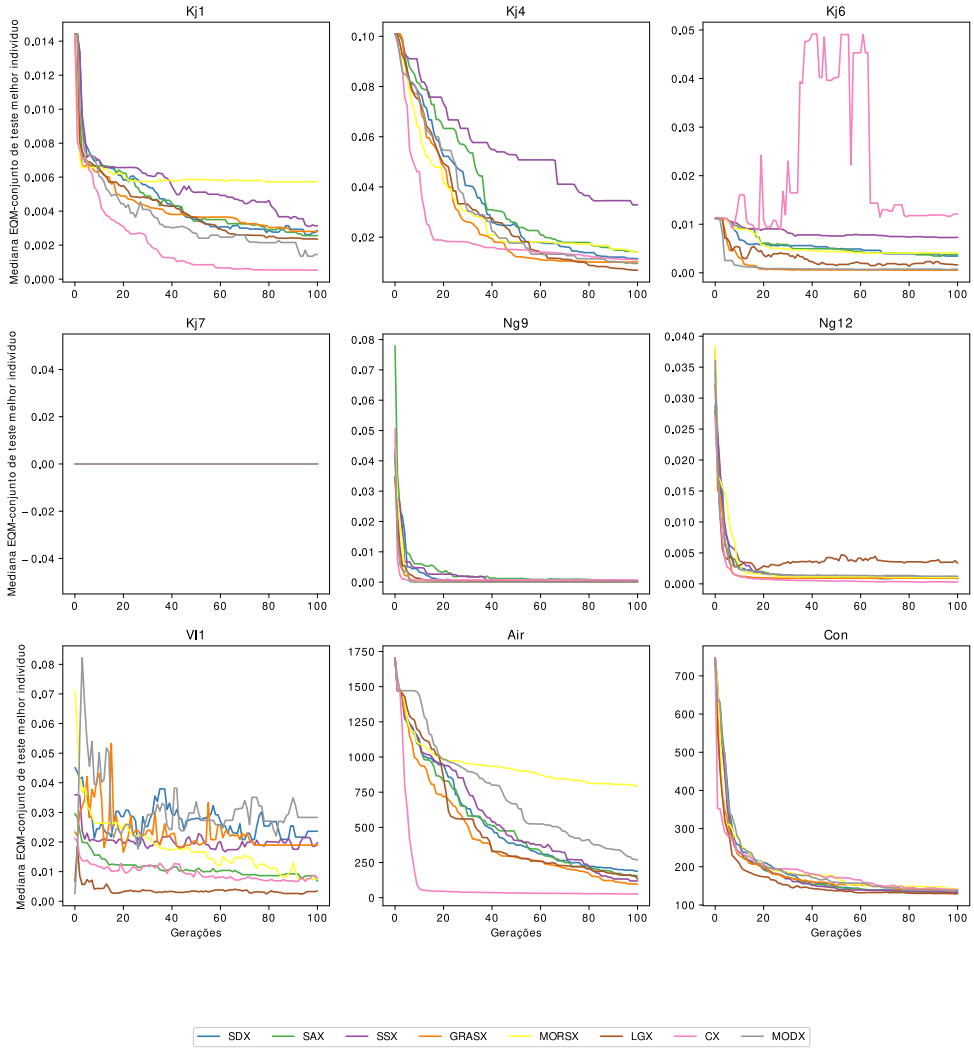


Figura 5.6 – Mediana do desempenho do melhor indivíduo para o conjunto de dados de testes ao longo das gerações no caso de operadores de cruzamento.

5.3.3 Tamanho do programa

As Tabelas 5.15 e 5.16 apresentam a média e o intervalo de confiança de 95% do número de nós do melhor indivíduo ao final da execução e a Figura 5.7 apresenta os mesmos valores para o melhor indivíduo ao longo das gerações.

Entre os métodos indiretos, programas maiores foram produzidos pelo operador GRASX, possuindo o maior indivíduo em 6 dos 9 problemas. Isso se deve à estratégia

Tabela 5.13 – Mediana e 95% do intervalo de confiança para o desempenho no conjunto de dados de testes no caso de operadores de cruzamento indiretos.

	SDX			SAX			SSX			GRASX			MORSX	
kj1	0.002 ≤	**0.003**	≤ 0.004	0.002 ≤	**0.003**	≤ 0.004	0.002 ≤	**0.003**	≤ 0.007	0.002 ≤	**0.003**	≤ 0.004	0.005 ≤	0.006	≤ 0.006
kj4	0.007 ≤	0.011	≤ 0.05	0.008 ≤	0.014	≤ 0.027	0.009 ≤	0.033	≤ 0.077	0.005 ≤	**0.010**	≤ 0.015	0.009 ≤	0.014	≤ 0.022
kj6	0.002 ≤	0.003	≤ 0.007	0.002 ≤	0.004	≤ 0.007	0.003 ≤	0.007	≤ 0.009	0.0 ≤	**0.001**	≤ 0.003	0.003 ≤	0.004	≤ 0.007
kj7	0.0 ≤	**0.000**	≤ 0.0	0.0 ≤	**0.000**	≤ 0.0	0.0 ≤	**0.000**	≤ 0.0	0.0 ≤	**0.000**	≤ 0.0	0.0 ≤	**0.000**	≤ 0.0
ng9	0.0 ≤	**0.000**	≤ 0.001	0.0 ≤	**0.000**	≤ 0.004	0.0 ≤	**0.000**	≤ 0.002	0.0 ≤	**0.000**	≤ 0.0	0.0 ≤	**0.000**	≤ 0.001
ng12	0.0 ≤	**0.001**	≤ 0.001	0.001 ≤	**0.001**	≤ 0.001	0.001 ≤	**0.001**	≤ 0.002	0.001 ≤	**0.001**	≤ 0.001	0.0 ≤	**0.001**	≤ 0.002
vl1	0.012 ≤	0.024	≤ 0.049	0.003 ≤	**0.007**	≤ 0.012	0.014 ≤	0.020	≤ 0.056	0.009 ≤	0.019	≤ 0.051	0.005 ≤	0.008	≤ 0.025
air	106.4 ≤	189.1	≤ 259.4	79.0 ≤	153.7	≤ 506.0	77.9 ≤	119.4	≤ 366.3	56.5 ≤	**94.3**	≤ 177.7	599.7 ≤	795.8	≤ 919.8
con	123.3 ≤	136.4	≤ 145.9	123.9 ≤	**131.4**	≤ 144.3	122.6 ≤	133.2	≤ 142.9	117.9 ≤	139.4	≤ 165.3	123.5 ≤	141.9	≤ 183.9
Rank		3.05			2.61			3.27			2.38			3.66	

Tabela 5.14 – Mediana e 95% do intervalo de confiança para o desempenho no conjunto de dados de teste no caso de operadores de cruzamento diretos.

	LGX			CX			MODX		
kj1	0.002 ≤	0.002	≤ 0.003	0.0 ≤	**0.001**	≤ 0.002	0.001 ≤	**0.001**	≤ 0.003
kj4	0.003 ≤	**0.007**	≤ 0.012	0.008 ≤	0.011	≤ 0.013	0.003 ≤	0.010	≤ 0.015
kj6	0.0 ≤	0.002	≤ 0.004	0.001 ≤	0.012	≤ 0.112	0.001 ≤	**0.001**	≤ 0.004
kj7	0.0 ≤	**0.000**	≤ 0.0	0.0 ≤	**0.000**	≤ 0.0	0.0 ≤	**0.000**	≤ 0.0
ng9	0.0 ≤	**0.000**	≤ 0.001	0.001 ≤	0.001	≤ 0.001	0.0 ≤	**0.000**	≤ 0.0
ng12	0.002 ≤	0.003	≤ 0.005	0.0 ≤	**0.000**	≤ 0.001	0.001 ≤	0.001	≤ 0.002
vl1	0.001 ≤	**0.003**	≤ 0.006	0.005 ≤	0.008	≤ 0.014	0.006 ≤	0.028	≤ 0.147
air	110.48 ≤	133.21	≤ 201.97	24.76 ≤	**26.78**	≤ 31.17	210.84 ≤	268.63	≤ 370.33
con	107.84 ≤	**129.75**	≤ 134.16	109.44 ≤	140.29	≤ 166.13	111.77 ≤	137.71	≤ 156.94
Rank		1.83			2.16			2.00	

de construção do arquivo de subárvores, a qual utiliza a sintaxe de um dos pais. É de se esperar que o tamanho desses itens também tenha tendência em crescer ao longo das gerações.

Observando a extensão multiobjetivo do operador GRASX, nota-se um controle efetivo no crescimento do indivíduo, com o menor tamanho em todos os problemas. Por consequência, a constituição dos itens presentes no arquivo de subárvores foi controlada indiretamente pelos conceitos multiobjetivo utilizados, produzindo subárvores menores.

Assim como apresentado no experimento com os operadores de mutação, o operador CM superou o operador multiobjetivo em 3 dos 9 problemas, devido ao cálculo da constante que mais se aproxima da semântica desejada, conforme observado no problema *air*, onde os todos os melhores indivíduos concebidos das 30 execuções tinham ao menos uma constante.

A dificuldade do operador LGX ficou evidente, concebendo os maiores indivíduos em todos os problemas. Em específico, para o problema ng9 o operador concebeu a equação desejada em 3 vezes das 30 execuções. Em contrapartida, o operador direto multiobjetivo proposto atingiu a equação desejada em 25 vezes das 30 execuções. Abaixo são apresentados os dois indivíduos concebidos pelos operadores LGX e MODX utilizando a notação em *Lisp*.

- LGX: $+$ ($\times$ (log ($+$ (x_2 , x_1)), sin ($+$ (sin (sin ($\times$ (sin (sin ($+$ (x_2 , x_1))), cos (sin (sin ($+$ ($-$ (x_2 , x_1), $+$ (x_1 , x_2))))))), sin ($\div$ (sin (x_2), cos ($+$ ($-$ (sin (sin (exp (cos (x_1)))), x_2), $\times$ (exp (log (x_2)), cos (cos (x_2))))))))), $\times$ (cos ($\times$ (sin (sin (sin (exp (cos (x_1))))), sin (sin (sin ($+$ ($\times$ (exp ($\div$ (log ($+$ (sin (x_2), cos (x_2))), exp ($\times$ ($\times$ ($+$ (x_2 , x_1), cos (x_2)), cos (x_2))))), x_2), $+$ (sin ($\div$ (sin (x_2), cos (x_1))), x_2))))))), cos (cos ($\times$ ($+$ (x_2 , cos (cos (sin (x_1)))), cos ($\times$ (cos

$(\times\ (sin\ (sin\ (x_1\ ))\ ,\ sin\ (sin\ (exp\ (cos\ (cos\ (sin\ (x_1\ ))))))))\ ,\ sin\ (sin\ (+\ (-\ (x_2\ ,\ x_1\ )\ ,\ +\ (x_1\ ,\ x_2\ ))))))))))$.

- MODX: $+\ (-\ (sin\ (\times\ (x_2\ ,\ x_2\ ))\ ,\ \div\ (-\ (x_1\ ,\ x_1\ )\ ,\ log\ (x_1\ )))\ ,\ sin\ (x_1\ ))$.

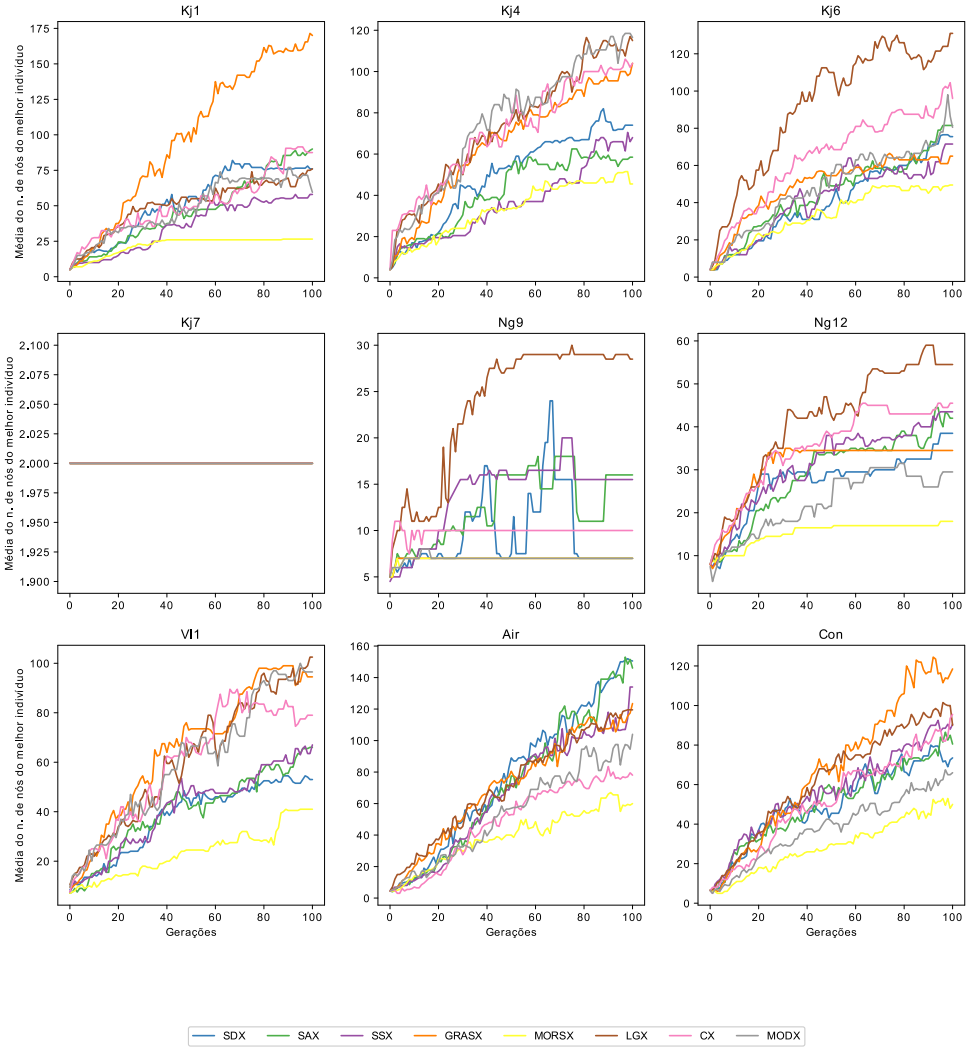


Figura 5.7 – Média do tamanho do melhor indivíduo ao longo das gerações no caso de operadores de cruzamento.

5.3.4 Tempo de execução

A Figura 5.8 e as Tabelas 5.17 e 5.18 apresentam média e intervalo de confiança de 95% para o tempo total de CPU consumido pelos operadores considerados ao longo das gerações, respectivamente.

Tabela 5.15 – Média e 95% do intervalo de confiança para o tamanho do melhor indivíduo no caso de operadores de cruzamento indiretos.

	SDX		SAX		SSX		GRASX		MORSX	
kj1	82.06	± 43.69	88.83	± 34.14	60.30	± 36.82	175.63	± 94.70	**27.90**	± 13.26
kj4	80.63	± 40.84	67.70	± 32.13	64.76	± 41.63	109.53	± 52.44	**52.30**	± 26.37
kj6	77.56	± 56.66	78.43	± 39.14	89.00	± 68.64	90.16	± 60.44	**57.26**	± 28.24
kj7	**2.00**	± 0.0	**2.00**	± 0.0	**2.00**	± 0.0	**2.00**	± 0.0	**2.00**	± 0.0
ng9	19.66	± 18.22	29.40	± 26.09	24.86	± 20.67	21.13	± 37.07	**10.33**	± 6.3
ng12	47.16	± 27.58	47.10	± 23.28	44.30	± 14.60	48.93	± 36.23	**20.23**	± 8.20
vl1	54.96	± 27.57	72.13	± 32.60	65.70	± 26.63	104.30	± 83.17	**43.50**	± 24.91
air	156.03	± 73.17	154.73	± 73.83	163.90	± 110.98	138.53	± 65.93	**60.30**	± 22.96
con	84.83	± 36.14	86.63	± 41.01	97.20	± 47.96	122.63	± 44.03	**49.66**	± 19.34
Rank	2.88		3.44		3.22		4.22		1.22	

Tabela 5.16 – Média e 95% do intervalo de confiança para o tamanho do melhor indivíduo no caso de operadores de cruzamento diretos.

	LGX		CX		MODX	
kj1	80.83	± 41.45	83.36	± 41.71	**70.33**	± 44.16
kj4	123.96	± 41.88	**107.26**	± 48.14	111.36	± 48.31
kj6	145.63	± 62.95	105.63	± 44.85	**91.63**	± 40.82
kj7	**2.00**	± 0.0	**2.00**	± 0.0	**2.00**	± 0.0
ng9	34.76	± 21.54	18.96	± 26.33	**7.66**	± 1.64
ng12	60.36	± 27.34	54.56	± 28.59	**35.00**	± 23.11
vl1	103.63	± 46.87	**74.20**	± 32.68	92.63	± 47.75
air	117.10	± 47.12	**84.70**	± 29.40	102.16	± 37.62
con	97.60	± 25.88	87.76	± 32.64	**67.50**	± 24.22
Rank	2.77		1.77		1.44	

Observa-se um custo computacional notoriamente elevado dos operadores GRASX e MORSX, possuindo os maiores custos computacionais, respectivamente, dentre os métodos indiretos. Os processos de construção do arquivo de itens e cálculo da distância semântica influenciam negativamente em seus custos computacionais. Nota-se que, quando comparado os operadores GRASX e MORSX, a inserção do critério de não-dominância pode ser benéfica para o tempo de processamento, devido à redução do tamanho dos indivíduos presentes na população e, consequentemente, à diminuição do tempo de avaliação dos mesmos.

Similarmente ao operador MORSX, a vertente multiobjetivo dos operadores diretos apresentou menor tempo computacional quando comparada ao operador CX, tendo melhor desempenho em 3 dos 9 problemas. Nota-se que os métodos indiretos LGX e CX consumiram maior tempo computacional que os métodos indiretos presentes na literatura. Entretanto, quando comparado o mesmo tempo computacional, o desempenho dos operadores semânticos diretos supera aquela produzida pelos operadores indiretos.

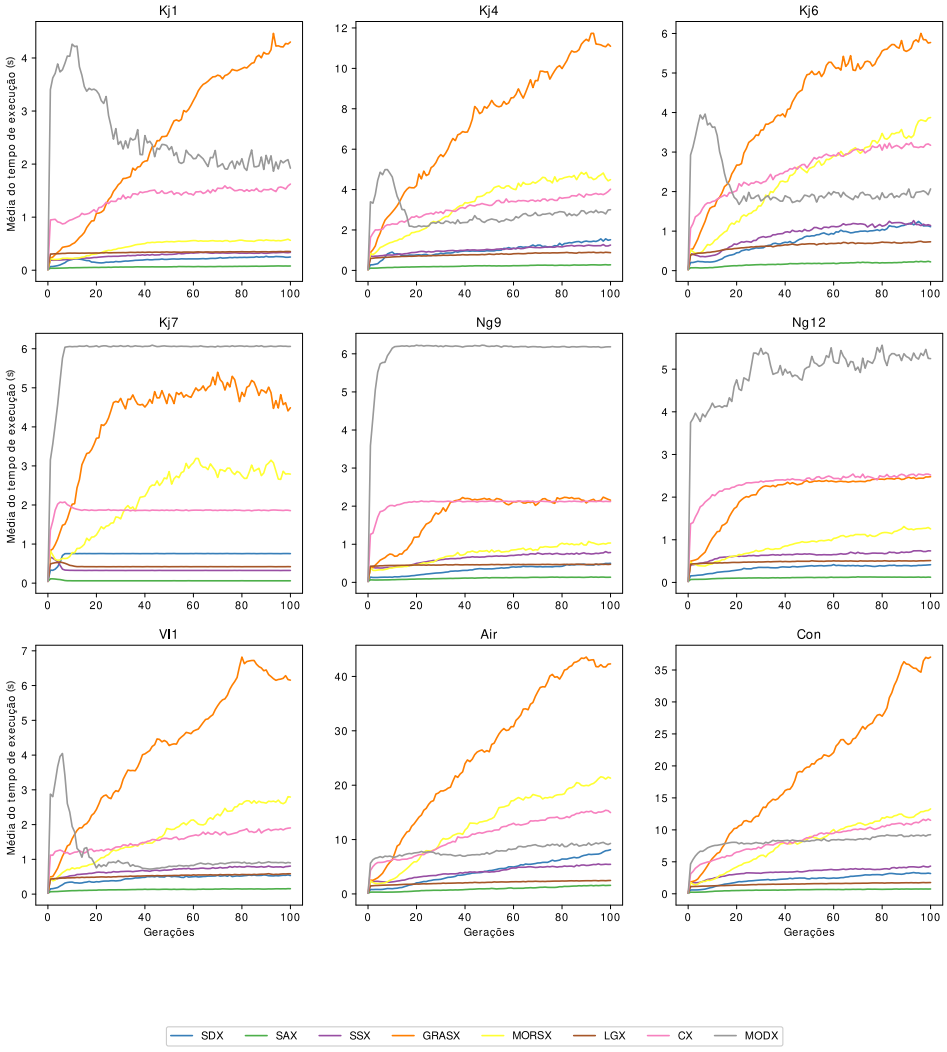


Figura 5.8 – Média do tempo computacional consumido ao longo das gerações no caso de operadores de cruzamento.

Tabela 5.17 – Média e 95% do intervalo de confiança para o tempo computacional consumido no caso de operadores de cruzamento indiretos.

	SDX		SAX		SSX		GRASX		MORSX	
kj1	21.74	± 5.91	**6.45**	± 1.1	28.25	± 5.16	243.45	± 90.74	47.23	± 12.44
kj4	125.89	± 82.93	**22.01**	± 5.32	101.59	± 25.05	772.13	± 276.60	348.50	± 108.88
kj6	82.34	± 32.52	**17.05**	± 3.34	94.63	± 38.56	491.62	± 257.66	270.08	± 103.50
kj7	74.28	± 0.38	**6.40**	± 0.11	34.09	± 0.43	442.83	± 213.8	240.57	± 114.26
ng9	36.32	± 16.91	**12.18**	± 4.07	70.27	± 21.09	232.01	± 161.67	80.083	± 30.89
ng12	39.60	± 17.35	**11.44**	± 1.48	65.96	± 8.28	249.63	± 137.08	91.34	± 34.97
vl1	47.53	± 14.20	**13.37**	± 1.87	69.78	± 8.55	449.03	± 198.01	177.13	± 51.65
air	444.3	± 146.4	**96.3**	± 17.3	419.1	± 73.5	2963.4	± 1018.2	1334.7	± 357.1
con	244.3	± 47.3	**63.6**	± 11.5	354.3	± 37.0	2011.2	± 524.0	853.9	± 171.1
Rank	2.33		1.00		2.66		5.00		4.00	

Tabela 5.18 – Média e 95% do intervalo de confiança para o tempo computacional consumido no caso de operadores de cruzamento diretos.

	LGX		CX		MODX	
kj1	**33.83**	± 1.10	135.60	± 23.22	313.11	± 148.36
kj4	**79.07**	± 6.38	332.45	± 70.09	313.19	± 125.06
kj6	**64.88**	± 7.45	250.81	± 56.43	217.75	± 41.52
kj7	**43.33**	± 0.34	189.67	± 10.23	597.81	± 3.70
ng9	**46.39**	± 1.40	208.35	± 16.45	601.53	± 61.59
ng12	**49.15**	± 1.8	237.73	± 27.37	432.43	± 146.28
vl1	**53.03**	± 3.04	163.16	± 46.65	178.58	± 149.93
air	**210.63**	± 8.70	1103.47	± 164.70	790.43	± 122.78
con	**153.33**	± 6.47	831.93	± 163.49	808.93	± 76.21
Rank	1.00		2.44		2.55	

6 Considerações Finais e Perspectivas Futuras

Esta pesquisa apresentou os principais conceitos de programação genética, geralmente associados a técnicas de aprendizado de máquina e aplicados em problemas de classificação e regressão. Um ramo recente da programação genética, que utiliza informações semânticas ao longo do processo evolutivo, conhecido como programação genética semântica, foi descrito juntamente com os principais operadores genéticos que fazem uso desse conceito. Este ramo explorado permite a concepção de operadores genéticos voltados para a natureza local da busca, possivelmente levando a uma evolução mais eficiente. Entretanto, o fenômeno de *bloating*, caracterizado por um aumento descontrolado no tamanho do programa ao longo das gerações, se mostra mais frequente.

Foi explorado o potencial de um formalismo multiobjetivo para conceber operadores efetivos capazes de controlar o fenômeno de *bloating*, sem comprometer seu desempenho. Em particular, desenvolvemos dois conjuntos de operadores genéticos, um direto e outro indireto. A terceira família de operadores genéticos utiliza o conceito da heurística construtiva do GRASP, podendo controlar seu grau de localidade. Recorreu-se a um arquivo de programas para fornecer subárvores aos operadores propostos de modo que eles possam criar modificações na população durante o processo de evolução. Ademais, uma nova estratégia de construção do arquivo de programas foi elaborada, visando manter parte da sintaxe do indivíduo após a troca de material genético em uma operação de cruzamento.

O desempenho dos algoritmos propostos foi avaliado em 9 problemas de regressão e foi incluída uma análise comparativa com concorrentes tidos como estado-da-arte na literatura. Entre os operadores de mutação, destaca-se o operador MODM, sendo que sua capacidade de generalização foi superior, para os problemas de *benchmarks*, com uma diminuição significativa no tamanho da solução, quando comparado aos demais métodos. Deste modo, ficam evidentes os benefícios da abordagem multiobjetivo para o controle de crescimento da solução. Além disso, o controle multiobjetivo pode ser benéfico para o tempo de execução, pois o tempo computacional está relacionado com o tamanho dos indivíduos da população. Para os operadores de cruzamento, as vertentes multiobjetivo possuíram uma convergência suave e decrescente, com um controle efetivo do crescimento do indivíduo. Entretanto, a constituição dos itens presentes no arquivo de subárvores foi controlada indiretamente pelos conceitos multiobjetivo utilizados, produzindo subárvores menores e, assim, influenciando negativamente na diversidade semântica do arquivo.

Para trabalhos futuros, uma possível frente é o estudo para combinação dos

operadores de mutação e cruzamento e uma extensão dos objetivos para lidar com outros critérios de desempenho, por exemplo, envolvendo noções mais avançadas de complexidade do programa. Além disso, novas formas de construção do arquivo podem ser examinadas, incluindo formas dinâmicas de construção que promovam diversidade nos objetivos utilizados e contendo item com valores constantes. Finalmente, poderão ser explorados operadores genéticos multiobjetivo em outros domínios, por exemplo, voltados ao tratamento de problemas booleanos e de predição de séries temporais.

Referências

ANGELINE, P. J. An investigation into the sensitivity of genetic programming to the frequency of leaf selection during subtree crossover. In: *Proceedings of the 1st Annual Conference on Genetic Programming*. Cambridge, MA, USA: MIT Press, 1996. p. 21–29. Citado na página 25.

BACK, T.; FOGEL, D. B.; MICHALEWICZ, Z. (Ed.). *Evolutionary computation 1: Basic algorithms and operators*. Bristol, UK: IOP Publishing Ltd., 2000. Citado na página 18.

BEADLE, L. C. J. *Semantic and Structural Analysis of Genetic Programming*. Tese (Ph.D. Thesis) — University of Kent at Canterbury, 2009. Citado 3 vezes nas páginas 16, 29 e 32.

BLEULER, S.; BADER, J.; ZITZLER, E. Reducing Bloat in GP with Multiple Objectives. In: *Multiobjective Problem Solving from Nature: From Concepts to Applications*. Berlin, Heidelberg: Springer, 2008. p. 177–200. Citado 3 vezes nas páginas 16, 28 e 43.

BLEULER, S. et al. Multiobjective genetic programming: Reducing bloat using SPEA2. *Proceedings of the IEEE Conference on Evolutionary Computation, ICEC*, 05 2001. Citado na página 28.

BURKE, E. K.; KENDALL, G. *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. 2nd. ed. New York, NY, USA: Springer, 2013. Citado na página 41.

CRAWFORD-MARKS, R.; SPECTOR, L. Size control via size fair genetic operators in the pushgp genetic programming system. In: *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002. (GECCO'02), p. 733–739. Citado na página 25.

DEB, K. *Multi-Objective Optimization Using Evolutionary Algorithms*. New York, NY, USA: John Wiley & Sons, Inc., 2001. Citado na página 26.

EBNER, M. et al. *Genetic programming: an introduction: on the automatic evolution of computer programs and its applications*. San Francisco, CA, US: Morgan Kaufmann Publishers, 1998. Citado na página 23.

EKLUND, S. E. A massively parallel architecture for linear machine code genetic programming. In: *Evolvable Systems: From Biology to Hardware*. Berlin, Heidelberg: Springer, 2001. p. 216–224. Citado na página 28.

FEO, T. A.; RESENDE, M. G. Greedy randomized adaptive search procedures. *Journal of global optimization*, Springer, v. 6, n. 2, p. 109–133, 1995. Citado na página 41.

FERARIU, L.; PATELLI, A. Multiobjective genetic programming for nonlinear system identification. In: *Adaptive and Natural Computing Algorithms*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. p. 233–242. Citado na página 43.

FOGEL, L. J.; OWENS, A. J.; WALSH, M. J. *Artificial intelligence through simulated evolution*. New York, NY, USA: John Wiley & Sons, 1966. Citado na página 19.

GALVÁN-LÓPEZ, E. et al. On the Use of Semantics in Multi-objective Genetic Programming. In: *Parallel Problem Solving from Nature – PPSN XIV*. Cham: Springer International Publishing, 2016. p. 353–363. Citado na página 47.

GALVÁN-LÓPEZ, E.; VÁZQUEZ-MENDOZA, L.; TRUJILLO, L. Stochastic semantic-based multi-objective genetic programming optimisation for classification of imbalanced data. In: . Cancún, Mexico: Springer International Publishing, 2017. p. 261–272. Citado na página 47.

HOLLAND, J. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. [S.l.]: University of Michigan Press, 1975. Citado na página 18.

KOZA, J. R. *Genetic programming: On the programming of computers by means of natural selection*. Cambridge, MA, USA: MIT Press, 1992. Citado 7 vezes nas páginas 16, 19, 20, 21, 23, 25 e 31.

KOZA, J. R. *Genetic Programming II: Automatic Discovery of Reusable Programs*. Cambridge, MA, USA: MIT Press, 1994. Citado na página 23.

KRAWIEC, K.; PAWLAK, T. Quantitative Analysis of Locally Geometric Semantic Crossover. In: *Parallel Problem Solving from Nature - PPSN XII*. Berlin, Heidelberg: Springer, 2012. v. 3, p. 397–406. Citado na página 35.

KRAWIEC, K.; PAWLAK, T. Approximating Geometric Crossover by Semantic Backpropagation. *Gecco'13: Proceedings of the 2013 Genetic and Evolutionary Computation Conference*, p. 941–948, 2013. Citado 2 vezes nas páginas 40 e 42.

KRAWIEC, K.; PAWLAK, T. Locally geometric semantic crossover: a study on the roles of semantics and homology in recombination operators. *Genetic Programming and Evolvable Machines*, v. 14, n. 1, p. 31–63, 2013. Citado 3 vezes nas páginas 35, 36 e 42.

LANGDON, W. B.; POLI, R. Fitness causes bloat: Mutation. In: *Proceedings of the First European Workshop on Genetic Programming*. London, UK: Springer-Verlag, 1998. p. 37–48. Citado na página 51.

LANGDON, W. B.; POLI, R. *Foundations of Genetic Programming*. Berlin, Heidelberg: Springer, 2002. Citado 5 vezes nas páginas 20, 23, 25, 31 e 45.

LUKE, S.; PANAIT, L. A Comparison of Bloat Control Methods for Genetic Programming. *Evolutionary Computation*, v. 14, n. 3, p. 309–344, sep 2006. Citado na página 25.

MCDERMOTT, J. et al. Genetic programming needs better benchmarks. In: *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*. New York, NY, USA: ACM, 2012. (GECCO '12), p. 791–798. Citado na página 49.

MILLER, J. F.; THOMSON, P. Cartesian genetic programming. In: *Genetic Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000. p. 121–132. Citado na página 20.

MORAGLIO, A.; KRAWIEC, K.; JOHNSON, C. G. Geometric semantic genetic programming. In: *Parallel Problem Solving from Nature - PPSN XII*. Berlin, Heidelberg: Springer, 2012. p. 21–31. Citado 2 vezes nas páginas 16 e 34.

NGUYEN, Q. U. *Examining Semantic Diversity and Semantic Locality*. Tese (Ph.D. Thesis) — University College Dublin, 2011. Citado 4 vezes nas páginas 16, 29, 33 e 34.

NGUYEN, Q. U.; HOAI, N.; NEILL, M. O. Semantics based mutation in genetic programming: The case for real-valued symbolic regression. *Mendel*, 01 2009. Citado na página 34.

NGUYEN, Q. U.; NGUYEN, X. H.; O'NEILL, M. Semantic aware crossover for genetic programming: The case for real-valued function regression. In: *Genetic Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. p. 292–302. Citado na página 34.

NGUYEN, Q. U. et al. Subtree semantic geometric crossover for genetic programming. *Genetic Programming and Evolvable Machines*, v. 17, n. 1, p. 25–53, 2016. Citado 2 vezes nas páginas 31 e 49.

NIEDERMEIER, R.; SANDERS, P. On the manhattan-distance between points on space-filling mesh-indexings. 06 1996. Citado na página 31.

OLIVEIRA, L. O. V. B. *Improving Search in Geometric Semantic Genetic Programming*. Tese (Doutorado) — Universidade Federal de Minal Gerais, 2016. Citado 3 vezes nas páginas 16, 29 e 49.

O'NEILL, M. et al. Open issues in Genetic Programming. *Genetic Programming and Evolvable Machines*, v. 11, n. 3-4, p. 339–363, 2010. Citado na página 45.

O'NEIL, M.; RYAN, C. Grammatical evolution. In: *Grammatical Evolution*. Boston, MA, USA: Springer, 2003. p. 33–47. Citado na página 20.

PAWLAK, T.; KRAWIEC, K. Competent Geometric Semantic Genetic Programming for Symbolic Regression and Boolean Function Synthesis. *Evolutionary Computation*, p. 1–36, feb 2017. Citado 2 vezes nas páginas 40 e 42.

PAWLAK, T.; WIELOCH, B.; KRAWIEC, K. Review and comparative analysis of geometric semantic crossovers. *Genetic Programming and Evolvable Machines*, v. 16, n. 3, p. 351–386, sep 2015. Citado 2 vezes nas páginas 39 e 49.

PAWLAK, T.; WIELOCH, B.; KRAWIEC, K. Semantic backpropagation for designing search operators in genetic programming. *IEEE Transactions on Evolutionary Computation*, v. 19, n. 3, p. 326–340, 2015. Citado 2 vezes nas páginas 29 e 40.

PAWLAK, T. P. *Competent Algorithms for Geometric Semantic Genetic Programming*. Tese (Ph.D. Thesis) — Poznan University of Technology, 2015. Citado 12 vezes nas páginas 9, 16, 29, 32, 34, 35, 36, 37, 39, 40, 42 e 56.

POLI, R. A simple but theoretically-motivated method to control bloat in genetic programming. *Genetic Programming*, v. 2610, p. 204–217, 2003. Citado na página 25.

POLI, R.; LANGDON, W.; MCPHEE, N. *A field guide to genetic programming*. UK: Lulu Enterprises, 2008. Citado 7 vezes nas páginas 20, 22, 23, 25, 28, 31 e 48.

RECHENBERG, I. Cybernetic solution path of an experimental problem (vol. library translation 1122). *Farnborough, UK: Royal Aircraft Establishment*, 1965. Citado na página 18.

SCHUETZE, O. et al. *NEO 2015: Results of the Numerical and Evolutionary Optimization Workshop*. Tijuana, Mexico: Springer Publishing Company, Incorporated, 2016. Citado 2 vezes nas páginas 9 e 30.

SCHWEFEL, H.-P. *Kybernetische evolution als strategie der experimentellen forschung in der strömungstechnik. Dipl.-Ing.* Tese (Ph.D. Thesis), 1965. Citado na página 18.

SUGANUMA, M.; SHIRAKAWA, S.; NAGAO, T. A genetic programming approach to designing convolutional neural network architectures. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. New York, NY, USA: ACM, 2017. p. 497–504. Citado na página 16.

VANNESCHI, L. *Theory and practice for efficient genetic programming*. Tese (Ph.D. Thesis) — Universite de Lausanne, 2004. Citado na página 19.

VANNESCHI, L.; CASTELLI, M.; SILVA, S. A survey of semantic methods in genetic programming. *Genetic Programming and Evolvable Machines*, v. 15, p. 195–214, 2014. Citado 3 vezes nas páginas 16, 29 e 31.