UNIVERSIDADE FEDERAL DE JUIZ DE FORA
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

João Marcos de Freitas

Uma Técnica de Programação Genética Gramatical e Semântica para Regressão Simbólica

Juiz de Fora
2024

João Marcos de Freitas

Uma Técnica de Programação Genética Gramatical e Semântica para Regressão Simbólica

Dissertação apresentada ao Programa de Pós-graduação em Ciência da Computação da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Heder Soares Bernardino

Coorientador: Prof. Dr. Itamar Leite de Oliveira

Juiz de Fora

2024

Ficha catalográfica elaborada através do Modelo Latex do CDC da UFJF
com os dados fornecidos pelo(a) autor(a)

de Freitas, João Marcos.
Uma Técnica de Programação Genética Gramatical e Semântica para Regressão Simbólica / João Marcos de Freitas. – 2024.
79 f. : il.

Orientador: Heder Soares Bernardino
Coorientador: Itamar Leite de Oliveira
Dissertação (Mestrado) – Universidade Federal de Juiz de Fora, Instituto de Ciências Exatas. Programa de Pós-graduação em Ciência da Computação, 2024.

1. Programação Genética. 2. Semântica. 3. Gramáticas Formais Livres de Contexto. 4. Aprendizado de Máquina.

João Marcos de Freitas

Uma Técnica de Programação Genética Gramatical e Semântica para Regressão Simbólica

Dissertação apresentada ao Programa de Pós-graduação em Ciência da Computação
da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do título de Mestre em Ciência da Computação. Área de concentração: Ciência da Computação.

Aprovada em 20 de setembro de 2024.

BANCA EXAMINADORA

Prof. Dr. Heder Soares Bernardino - Orientador
Universidade Federal de Juiz de Fora

Prof. Dr. Itamar Leite de Oliveira - Coorientador
Universidade Federal de Juiz de Fora

Profª. Dra. Luciana Brugiolo Gonçalves
Universidade Federal de Juiz de Fora

Profª. Dra. Jaqueline da Silva Angelo
Fundação Oswaldo Cruz

Juiz de Fora, 22/08/2024.

Documento assinado eletronicamente por **Jaqueline S. Angelo**, **Usuário Externo**, em 20/09/2024, às 16:51, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Heder Soares Bernardino**, **Professor(a)**, em 20/09/2024, às 16:51, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Luciana Brugiolo Goncalves**, **Professor(a)**, em 24/09/2024, às 16:34, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Itamar Leite de Oliveira**, **Professor(a)**, em 24/09/2024, às 17:17, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no Portal do SEI-Ufjf (www2.ufjf.br/SEI) através do ícone Conferência de Documentos, informando o código verificador **1939433** e o código CRC **FE94BAE0**.

Dedico ao meu avô Jorge, que faleceu durante a produção deste trabalho, que mesmo em sua simplicidade sempre me incentivou a estudar e se interessou pelo meu progresso, desde aprender a nadar até à minha formação acadêmica.

AGRADECIMENTOS

Ao Prof. Dr. Heder, meu orientador, pelo apoio incondicional, paciência, disponibilidade, pelo aconselhamento assertivo e pelo estímulo permanente, desde o início da minha graduação até agora. Ao Prof Dr. Itamar, meu co-orientador, pelo aconselhamento e ensinamentos que possibilitaram o desenvolvimento de minhas pesquisas e trabalhos. A todo corpo docente do Departamento de Ciência da Computação, por ter acreditado em mim e transferido os conhecimentos necessários para que eu concluísse o mestrado. À UFJF, por fornecer a estrutura necessária para que tudo isso possa ocorrer. Aos meus pais, pelo amor e apoio incondicional desde o começo, e por sempre acreditarem em mim. Aos meus familiares pelo suporte e aos meus amigos, por animarem meu espírito para passar por esse período. À Luísa, por ter sido minha companheira nos momentos mais difíceis e também nas minhas vitórias, me alegrando nos momentos difíceis e me estimulando quando pensava em desistir.

Agradeço ainda ao apoio financeiro das agências de fomento CNPq, FAPEMIG e CAPES, que viabilizaram o desenvolvimento deste trabalho.

"Sometimes it is the people no one imagines anything of who do the things that no one can imagine."
Alan Turing

RESUMO

A Regressão Simbólica (RS) é uma técnica de modelagem cujo objetivo é encontrar uma expressão completa, e não apenas ajustar coeficientes de um modelo fixo, como ocorre em métodos de regressão convencionais. A RS consiste em descobrir automaticamente a estrutura de um modelo a partir de dados, o que pode resultar em soluções mais precisas e interpretáveis para problemas complexos. A utilização da Programação Genética (PG) para resolver problemas de RS é particularmente vantajosa, pois a PG explora o espaço de soluções de forma eficiente e garante que as expressões matemáticas geradas sejam sintaticamente válidas. Além disso, o uso de gramáticas formais livres de contexto (GLC) gera a Programação Genética Gramatical (PGG), que possibilita a definição de regras e delimitações no espaço de soluções a ser explorado durante a regressão. Isso é crucial em RS, onde simplicidade e interpretabilidade dos modelos são fatores determinantes. Contudo, a geração livre de candidatos em métodos de RS frequentemente leva à criação de soluções redundantes ou irrelevantes. A Programação Genética Semântica (PGS), ao considerar a semântica dos candidatos durante suas operações, mostrou-se capaz de evitar esses problemas entre diversas abordagens, aumentando tanto a eficiência da busca quanto a qualidade das soluções. Isso resulta em modelos de RS que capturam de maneira mais precisa as relações intrínsecas nos dados. A Programação Genética Gramatical e Semântica (PGGS) combina as gramáticas livres de contexto da PGG com o uso de semântica, aprimorando ainda mais a capacidade de orientar a busca para a produção de expressões que sejam tanto válidas quanto semanticamente relevantes para o problema em questão. Neste trabalho, a PGGS é estudada em profundidade, com foco em seu operador de recombinação, considerando diferentes abordagens e seus efeitos sobre a busca e a qualidade das soluções geradas. Propõe-se o *Roulette Semantic Crossover* (RSC) como um novo operador de recombinação, que cria uma roleta entre as soluções candidatas baseada em suas semânticas, aumentando as chances de gerar novos candidatos relevantes. Além disso, é realizada uma análise paramétrica desses operadores e do desempenho da técnica em um conjunto de problemas de um *benchmark* desenvolvido para avaliar métodos de RS, destacando as vantagens e limitações da abordagem proposta. O RSC demonstrou ser robusto, alcançando a maior área sob a curva nos Perfis de Desempenho (PP) e obtendo o maior número de sucessos nos testes. Esses resultados indicam que a PGGS é uma abordagem promissora para RS.

Palavras-chave: Programação Genética, Semântica, Gramáticas Formais Livres de Contexto, Aprendizado de Máquina.

ABSTRACT

Symbolic Regression (SR) is a modeling technique aimed at finding a complete expression, rather than just adjusting coefficients in a fixed model, as is done in conventional regression methods. SR involves automatically discovering the structure of a model from data, which can result in more accurate and interpretable solutions for complex problems. The use of Genetic Programming (GP) to solve SR problems is particularly advantageous because GP explores the solution space efficiently and ensures that the generated mathematical expressions are syntactically valid. Additionally, the use of context-free grammars (CFG) leads to Grammatical Genetic Programming (GGP), which allows for the definition of rules and boundaries within the solution space to be explored during regression. This is crucial in SR, where simplicity and interpretability of models are key factors. However, the unrestricted generation of candidates in SR methods often leads to redundant or irrelevant solutions. Semantic Genetic Programming (SGP), by considering the semantics of candidates during its operations, has shown the ability to avoid these issues across various approaches, enhancing both search efficiency and solution quality. This results in SR models that more accurately capture the intrinsic relationships within the data. Grammatical and Semantic Genetic Programming (GSGP) combines the context-free grammars of GGP with the use of semantics, further enhancing the ability to guide the search towards producing expressions that are both valid and semantically relevant to the problem at hand. In this study, GSGP is analyzed in depth, focusing on its recombination operator and considering different approaches and their effects on search and solution quality. We propose the Roulette Semantic Crossover (RSC) as a new recombination operator that creates a roulette among candidate solutions based on their semantics, increasing the likelihood of generating relevant new candidates. Furthermore, a parametric analysis of these operators and the technique's performance is conducted on a benchmark set of problems developed to evaluate SR methods, highlighting the advantages and limitations of the proposed approach. RSC demonstrated robustness, achieving the highest area under the curve in Performance Profiles (PP) and obtaining the highest number of successes in the tests. These results indicate that GSGP is a promising approach for SR.

Keywords: Genetic Programming, Semantics, Context-Free Grammars, Machine Learning.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ABS	Distância Absoluta
AE	Algoritmos Evolutivos
AG	Algoritmo Genético
AGC	*Approximating Geometric Crossover*
AM	Aprendizado de Máquina
ANOVA	Análise de variância
AUC	*Area under the curve*
CE	Computação Evolucionista
DS	Distância Semântica
ED	Evolução Diferencial
EE	Estratégia Evolutiva
EG	Evolução Gramatical
ES	Equivalência Semântica
GLC	Gramática Formal Livre de Contexto
k-fold	*k-Folds Cross Validation*
NSM	*No Same Mate*
NT	Não Terminal
PG	Programação Genética
PGG	Programação Genética Gramatical
PGGS	Programação Genética Gramatical e Semântica
PGS	Programação Genética Semântica
PP	*Performance Profiles*
ROBDD	*Reduced Ordered Binary Decision Diagrams*
RS	Regressão Simbólica
RSC	*Roullete Semantic Crossover*
SAC	*Semantic Aware Crossover*
SBS	*Soft Brood Selection*
SCPS	*Semantic Crossover for Program Synthesis*
SDC	*Semantically Driven Crossover*
SS	Similaridade Semântica
SSC	*Semantic Sensitive Crossover*

LISTA DE SÍMBOLOS

ϵ	Limite inferior de equivalência semântica
α	Limite inferior de similaridade semântica
β	Limite superior de similaridade semântica

SUMÁRIO

1 INTRODUÇÃO

A modelagem precisa de relações complexas são essenciais para descrever fenômenos em diversos domínios. Modelos são ferramentas para representar fenômenos muitas vezes abstratos e complexos de uma forma simplificada e compreensível. Eles podem ser usados para realizar predições e simular o fenômeno em diferentes condições. Sua aplicação é ampla, desde a pesquisa científica de base até a prática no dia a dia das pessoas, passando pela indústria e comércio. A capacidade de representação dos modelos é enorme, cobrindo fenômenos naturais como o clima, o movimentos de fluidos, o funcionamento de máquinas e algoritmos, a análise de dados em ciências sociais e econômicas, agentes em processos, entre muitos outros (Bernardino, 2012; Little, 2013; Veiga et al., 2015; de Freitas et al., 2015). Tão importante quanto a capacidade de modelar e reproduzir esses fenômenos, conseguir compreender os fatores que os regem é crucial. A regressão simbólica é uma ferramenta eficaz para extrair modelos a partir de dados, mantendo um alto grau de interpretabilidade e explicabilidade (Koza, 1992; Bongard and Lipson, 2007; Schmidt and Lipson, 2009). Ao contrário dos métodos tradicionais de regressão, que assumem uma estrutura de modelo predefinida, a regressão simbólica busca, no espaço de expressões fornecido, a forma de um modelo que melhor se ajusta aos dados. Essa flexibilidade permite descobrir padrões e relações intrínsecas que poderiam permanecer ocultos.

A Programação Genética (PG) (Koza, 1992) estabeleceu-se como uma técnica poderosa na regressão simbólica, principalmente devido à sua capacidade de encontrar modelos utilizando o conceito de seleção natural (Darwin, 1859), através de operações bioinspiradas como a recombinação e mutação genéticas, além da seleção baseada em aptidão. A PG se mostrou capaz de gerar soluções promissoras em diversos tipos de problemas e aplicações, sempre aliando a capacidade de generalização com representação das soluções de forma simbólica. No entanto, uma parte considerável de suas operações de recombinação frequentemente não resultam em mudanças significativas na estrutura dos modelos, consumindo processamento sem contribuir para o processo evolutivo do algoritmo. Essa ineficiência pode ser atribuída a um processo de recombinação cego, sem considerar o impacto que a operação causa, resultando em descendentes que são muito semelhantes ou radicalmente diferentes de seus pais, sem melhoria significativa no desempenho e levando a uma baixa diversidade genética.

Para resolver esses problemas, a semântica pode ser incorporada à PG para melhorar tanto a diversidade da população quanto sua localidade (Nguyen et al., 2009). Abordagens baseadas em semântica garantem que as operações de recombinação produzam descendentes que introduzem variações úteis, melhorando a eficiência do processo evolutivo. Essas abordagens diferem entre si no formato de aplicação da semântica, começando com estratégias mais simples, como evitar que sejam geradas novos indivíduos idênticos aos pais, como no caso do *No Same Mate* (NSM) que evita inserir na população novos indivíduos

idênticos a um dos pais. Outra alternativa é explorar mais uma recombinação entre os pais, gerando uma quantidade maior de filhos, como no caso do *Soft Brood Selection* (SBS) (Tackett and Carmi, 1994), onde somente o filho gerado com melhor aptidão é inserido na população. Esse tipo de abordagem considera, ainda que de uma forma mais simples, o uso de informações das soluções como critério de entrada na população.

O uso da semântica vai além disso, é baseado em considerar essa informação de forma mais enfática. O *Approximating Geometric Crossover* (AGC) (Krawiec and Lichocki, 2009) associa uma semântica às soluções, e de forma similar ao SBS gera um conjunto de filhos durante a recombinação, porém o critério de entrada na população deixa de ser aquele com melhor aptidão, mas passa a ser aquele mais similar a ambos os pais. No entanto, o custo computacional da operação é alto, dado que é necessário gerar muitos filhos para escolher apenas um. Já o *Semantically Driven Crossover* (SDC) (Beadle and Johnson, 2008) se assemelha ao NSM, adicionando o filho na população se for diferente semanticamente dos pais. Esse tipo de abordagem ainda considera a semântica apenas do modelo completo, enquanto técnicas de PG realizam operações no genótipo, que são refletidas no fenótipo. Os trechos alterados dentro do genótipo representam uma parte do modelo, como um submodelo, e também possuem uma semântica. Essa informação semântica em um submodelo pode facilitar seu uso, por não ser necessário calcular a semântica de todo o modelo, reduzindo o custo computacional de seu cálculo. Além disso, é possível evitar a geração de múltiplos filhos a fim de obter pelo menos um com significância.

No contexto da PG já existem aplicações bem sucedidas de semântica, o *Semantic Aware Crossover* (SAC) (Nguyen et al., 2009) apresenta uma versão de recombinação semântica ainda mais simples, inserindo os filhos na população apenas se não forem equivalentes aos pais. Já sua versão aprimorada *Semantic Sensitive Crossover* (SSC) (Uy et al., 2010) realiza tentativas de troca de subárvores, no entanto é verificada a semântica das subárvores em um conjunto reduzido de dados, e somente caso elas sejam similares é realizada a recombinação.

A Programação Genética Gramatical (PGG) (de Freitas et al., 2015) é uma variante notável da PG é estruturada por gramáticas livres de contexto, a fim de gerar apenas soluções viáveis. Essa abordagem impõe regras que regem a geração dos modelos, podendo representar restrições e introduzir viés com base no conhecimento prévio do problema, o que pode melhorar significativamente a qualidade dos modelos durante a evolução.

Uma integração para a PGG com uso de semânticas é o operador *Semantic Crossover for Program Synthesis* (SPCS) (Forstenlechner, 2019), que faz a síntese de programas utilizando uma abordagem de PGG. Já nesse seu caso, a semântica é fortemente dependente da representação, onde cada variável do programa gerado têm sua própria semântica.

Apesar das vantagens inerentes ao uso de gramáticas e semântica de forma in-

dependente, sua aplicação combinada é desafiadora devido às dependências intrincadas das características específicas do domínio do problema, podendo ser difícil alcançar uma integração harmoniosa.

Neste trabalho é proposto um novo operador de recombinação baseado em semântica e adaptando três outros operadores existentes da literatura para funcionar com a versão gramatical da PG para regressão simbólica de funções de valor real. Os operadores da literatura são SAC, SSC e SCPS. O SAC e SSC são operadores criados para a PG, e portanto necessitam de adaptações para serem aplicados à PGG, já o SCPS já é uma abordagem que considera o uso de gramáticas formais livres de contexto (GLC). O algoritmo de recombinação proposto, *Roulette Semantic Crossover* (RSC), inverte a lógica de uso da semântica, onde anteriormente a semântica era utilizada para aceitar o resultado de uma recombinação, no RSC ela passa a ser critério de escola do que será recombinado. Além disso, a roleta cria um mecanismo onde todas as possíveis trocas de subárvores podem acontecer, não somente a mais similar, ou não equivalente. A cada uma das possibilidades de troca é atribuída uma probabilidade de ser escolhida, com uma vantagem para trocas mais similares.

Para avaliar o desempenho do RSC e comparar com os resultados do SAC e SSC adaptados para PGG, e o SCPS, um conjunto de experimentos serão realizados através de um *benchmark* para PG. Os resultados são comparados realizando testes estatísticos, contagem de sucessos, e através da avaliação dos Perfis de Desempenho (PP) (Dolan and Moré, 2002).

1.1 Organização do Texto

O restante deste trabalho está organizado da seguinte forma:

- **Capítulo 2: Regressão Simbólica** - Aborda o tema de regressão, apresenta a regressão simbólica e discute a importância dos *benchmarks* para teste e comparação entre técnicas.
- **Capítulo 3: Programação Genética** - Introduz a Programação Genética, a computação evolucionista em geral, a Programação Genética Gramatical (PGG), gramáticas formais, a representação da PGG e os operadores genéticos.
- **Capítulo 4: Programação Genética Semântica** - Apresenta a semântica, conceitos de distância, similaridade e equivalência, e revisa as aplicações de semântica e métodos de recombinação da literatura de PG.
- **Capítulo 5: Estratégias Propostas** - Introduz as estratégias propostas, apresenta a PGG Semântica, a representação da semântica na PGG e o novo operador *Roullete Semantic Crossover* (RSC).

- **Capítulo 6: Experimentos Computacionais** - Apresenta os experimentos propostos, o protocolo experimental do *benchmark*, as parametrizações dos algoritmos e as ferramentas de avaliação dos resultados.
- **Capítulo 7: Conclusões e Trabalhos Futuros** - Apresenta os resultados, compara os algoritmos da literatura com a PGG e o RSC, e discute as conclusões dos resultados e trabalhos futuros.

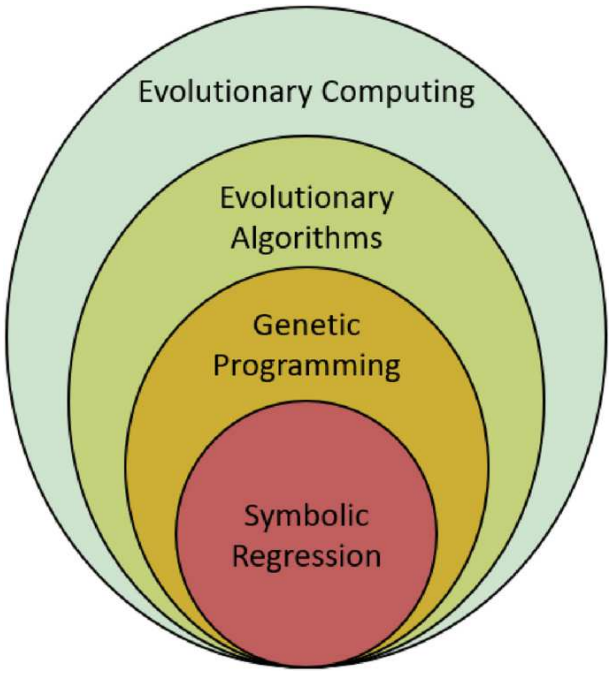
2 REGRESSÃO SIMBÓLICA

A busca por modelos capazes de explicar fenômenos é uma constante em todas as áreas da ciência, especialmente no que diz respeito a modelos matemáticos que representam conjuntos de dados. A capacidade gerar modelos a partir de dados é importante para nos capacitar a simular e descrever o fenômeno, ganhando a habilidade preditiva e compreensão sobre ele. A regressão é um método estatístico desenvolvido para abordar esse problema, focando na investigação da relação entre variáveis de um problema e seu resultado.

Tradicionalmente, os métodos de regressão incluem a regressão linear, polinomial e não linear (Cheng et al., 2019; Huang and He, 2023). Dessa forma, há uma modelagem correspondente que busca o melhor ajuste dos dados a um tipo de modelo, permitindo desde abordagens mais simples, como a linear, até opções mais complexas, como a não linear. Através dessas abordagens a regressão é capaz de ajustar o modelo para aqueles dados.

No entanto, em alguns casos esse tipo de modelagem pode não ser capaz de fornecer um grau satisfatório de representatividade dos dados. Surge então a necessidade de utilizar métodos computacionais que tenham maior capacidade de representação, como algoritmos de CE capazes de resolver problemas de maior complexidade. Esses métodos devem funcionar como uma regressão, mas devem também possuir a habilidade de encontrar, de forma autônoma, uma expressão para o problema em questão.

Figura 1 – Regressão Simbólica é uma subárea de Computação Evolucionista, mas também pode ser aplicada em outras áreas.



Fonte: Angelis et al. (2023)

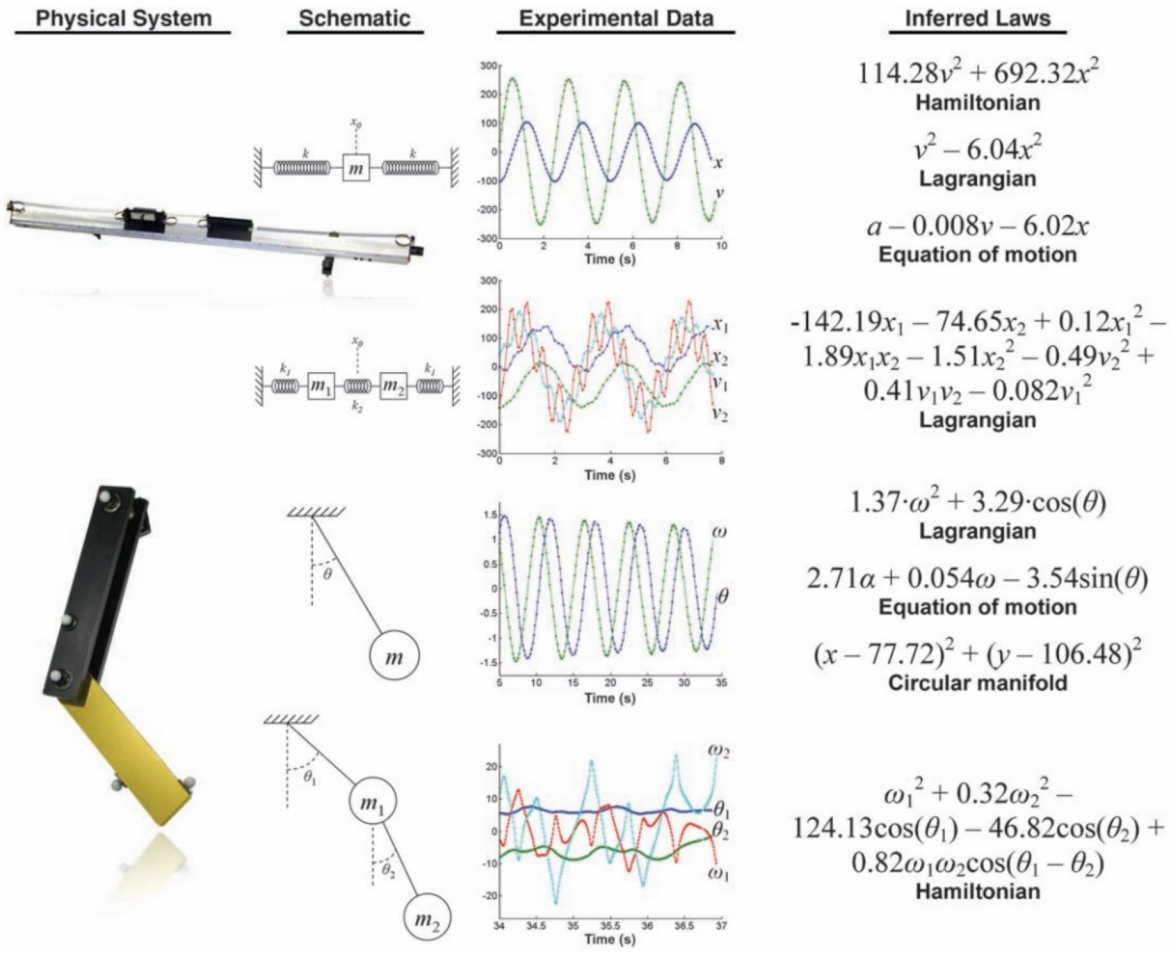
A Regressão Simbólica (RS) surge como uma solução para esse desafio, sendo mais flexível que a regressão clássica e utilizando algoritmos para encontrar uma forma funcional mais adequada ao problema. Essa busca pode ser realizada de forma sistemática ou heurística, variando entre diversos métodos com o mesmo objetivo. Em geral, métodos de Aprendizado de Máquina (AM) são empregados para realizar essa busca, devido a sua capacidade de lidar com a complexidade da busca a um custo computacional menor do que uma abordagem sistemática. A complexidade da busca depende de diversos fatores, como o conjunto de variáveis do problema, as estratégias de geração de constantes numéricas, a quantidade de parâmetros e sensibilidade das operações que podem aparecer no modelo, além do custo computacional da avaliação de uma solução (execução do modelo no conjunto de dados) (Orzechowski et al., 2018). Outro fator de complexidade é o tamanho dos modelos e as possibilidades de relação entre as variáveis. Por exemplo, um modelo polinomial como $f(x) = x^2 + 3$ pode ser maior que o modelo não linear $f(x) = e^x$, mas o segundo é muito mais sensível às variações de x, onde dependendo do domínio, uma pequena alteração no valor no expoente leva a uma imagem muito diferente da função.

Dentre as abordagens de AM utilizadas em RS, a Programação Genética (PG) tem se mostrado como uma das com maior potencial, sendo capaz de realizar a regressão de diversos tipos de problemas baseados em dados, especialmente aqueles que envolvem valores numéricos. Sua capacidade de abstração e de lidar com ruídos nos dados resultou em desempenhos notáveis, permitindo a descoberta de padrões anteriormente desconhecidos. Além disso, a RS via PG é capaz de captar comportamentos intrínsecos aos dados, onde dependendo da modelagem do problema pode ser capaz de detectar até mesmo o fenômeno que explica uma classe de problemas, e não apenas aquela instância do evento representado pelos dados (Schmidt and Lipson, 2009).

Além da capacidade de geração de modelos e da habilidade de definir regras para sua formação, a PG ainda oferece um diferencial extremamente importante: a legibilidade e interpretabilidade dos modelos. Uma vez que os modelos são gerados através de regras da gramática, eles são escritos na linguagem definida por ela, tornando-se facilmente legíveis e compreensíveis, além de serem simplificáveis quando possível. Aliando essas características à precisão dos modelos, a PG se torna uma ferramenta de grande relevância no campo da RS e da AM como um todo.

Um caso especial de RS é sua aplicação com a PG, e em especial a PGG. A característica principal da PGG é o uso de GLC, o que agrega muito valor à busca de modelos. Através de gramáticas, é possível definir o conjunto de variáveis e operadores que serão utilizados para construir os modelos, assim como em outros métodos. Mas além disso, é possível estabelecer facilmente regras para a formação das expressões que irão compor o modelo. Dessa maneira, pode-se elaborar uma gramática que gere modelos lineares, polinomiais e não lineares. Ao mesmo tempo, é possível controlar aspectos dos modelos, permitindo que certos elementos sejam presentes ou completamente eliminados, como

Figura 2 – Exemplo de problemas resolvidos pela Programação Genética. A imagem ilustra diferentes sistemas físicos, onde são extraídos dados como a posição das massas ao longo do tempo, e então são encontrados modelos que descrevem as leis que regem o movimento das massas nos fenômenos.



Fonte: Schmidt and Lipson (2009)

restringindo certas relações entre variáveis ou expressões com variáveis como expoentes. Além disso, a definição de constantes numéricas é possível, através de operações entre constantes presentes na gramática. Também é viável explorar conhecimento prévio sobre o problema e, por meio das regras, incorporá-lo ao processo de busca, como nas expressões gênicas onde existem cinéticas conhecidas como Michaelis-Menten (Srinivasan, 2020) que tem um formato específico.

Para verificar as capacidades de algoritmos, especialmente em AM, geralmente são utilizados *benchmarks*. Um *benchmark* é uma ferramenta desenvolvida para avaliar e comparar o desempenho de diferentes metodologias de forma justa e equivalente (Orzechowski et al., 2018). Essa capacidade comparativa torna os *benchmarks* muito presentes na validação de algoritmos em AM, especialmente para RS e PG. Alguns elementos podem ser destacados, como a padronização de comparação entre métodos, com conjuntos de dados estabelecidos ou regras para geração das bases de dados, além de critérios de avalia-

ção. Também se destacam a validação de modelos, a avaliação de robustez, medições de desempenho e verificação de resultados. Isso facilita que novos trabalhos sejam comparados com a literatura existente, ou que os algoritmos comparados não precisem ser executados novamente.

3 PROGRAMAÇÃO GENÉTICA

Este capítulo apresenta a área da Computação Evolucionista e introduz os Algoritmos Evolucionistas na linha de desenvolvimento da Programação Genética. Em seguida são apresentados seus fundamentos, mecanismo de busca, operadores genéticos e sua adaptação para a Programação Genética Gramatical.

3.1 Introdução

A automatização de processos vem sendo aprimorada através dos séculos, mas foi impulsionada de forma ímpar nos últimos anos com o advento e rápido avanço da computação moderna, e é expressada especialmente pelo Aprendizado de Máquina (AM). O AM desenvolve modelos que aprendem a realizar tarefas a partir de dados, melhorando o desempenho automaticamente sem intervenção explícita (El Naqa and Murphy, 2015).

Nesse contexto, um conjunto amplo de técnicas apresenta características baseadas em conceitos biológicos, e em especial no conceito de seleção natural, a Computação Evolucionista (CE). Nesse grupo, técnicas como a Programação Genética (PG) foram desenvolvidas com o intuito de auxiliar na descoberta de conhecimento, especialmente em áreas científicas. A PG, formalizada por Koza (1992), é uma meta-heurística bioinspirada nos princípios da seleção natural propostos por Darwin (1859). Embora compartilhe semelhanças com outros métodos de CE como os Algoritmos Genéticos (AG), a PG se destaca como um dos algoritmos evolutivos mais relevantes, sendo objeto de extensa pesquisa em suas diversas variações e aplicações.

O principal objetivo da PG é a criação automática de programas. A PG busca gerar um programa que resolva um determinado problema da maneira mais precisa possível, utilizando uma abordagem simbólica que se baseia em um conjunto de símbolos definidos pelo usuário. Diferentemente da regressão clássica, onde a forma do modelo é pré-estabelecida, na regressão simbólica e, em particular, na PG, a estrutura do modelo é livre e não imposta *a priori*.

A capacidade de descobrir programas automaticamente é extremamente valiosa, especialmente em situações onde não existe uma solução conhecida. Além disso, a PG pode ser utilizada para encontrar representações alternativas de soluções já conhecidas, mas que apresentem menor complexidade ou maior acurácia. Outro aspecto relevante é a avaliação da PG em termos de sua eficácia em identificar soluções específicas. A automatização desse processo não apenas torna a descoberta de conhecimento viável, mas também é crucial para a resolução de problemas que exigem uma leitura e interpretação cuidadosa das soluções obtidas.

As aplicações da PG são vastas e abrangem diversas áreas, como engenharia, biologia e ciência da computação. Na engenharia, por exemplo, a PG tem sido utilizada

para otimização de projetos e sistemas complexos, permitindo a criação de soluções inovadoras que atendem a requisitos específicos, como a identificação de componentes para o cálculo da máxima deformação longitudinal em dutos com amassamento, a fim de identificar avarias críticas em dutos utilizados na extração de petróleo (de Freitas et al., 2015). Em biologia, a PG tem sido utilizada para a descoberta de padrões em dados genômicos, contribuindo para avanços na pesquisa biomédica, como a inferência de desenvolvimento de alergia em jovens no nordeste do Brasil (Veiga et al., 2015, 2018). A versatilidade da PG em lidar com problemas complexos e sua habilidade de gerar soluções criativas a tornam uma ferramenta poderosa em contextos onde métodos tradicionais podem falhar.

3.2 Aprendizado de Máquina

A automatização de processos é uma prática que remonta a séculos, mas ganhou um impulso significativo com o advento da computação e, em particular, métodos de Aprendizado de Máquina (AM) ganharam muito destaque. O AM é um campo da computação focado no desenvolvimento de modelos que buscam aprender, de forma autônoma, a realizar tarefas. Esse processo de aprendizado busca melhorar automaticamente o seu desempenho de resolução da tarefa sem intervenção explícita (El Naqa and Murphy, 2015), e pode ser baseado em dados ou não. Quando o aprendizado utiliza dados para treinar é chamado aprendizado supervisionado, já quanto não são utilizados dados de treinamento é chamado de não supervisionado. Alguns elementos estão comumente presentes em métodos de aprendizado, como dados, parâmetros e formas de avaliação de soluções. Os dados são elementos fundamentais para o aprendizado supervisionado, pois seu objetivo é aprender a resolver problemas a partir de deles. Geralmente, esses dados são apresentados em diferentes momentos aos métodos de aprendizado: o primeiro é o treinamento, e depois há um processo de validação e/ou teste. O treinamento é o processo de apresentar uma parte dos dados ao algoritmo utilizado, e então avaliar aquela solução. Esse conjunto de dados é chamado de *dados de treinamento*. Dado o resultado, é possível medir o desempenho daquela solução, e essa informação é considerada pelo processo de aprendizagem. Ao longo de iterações, as soluções vão se adaptando a fim de aumentar seu desempenho, geralmente minimizando o erro ao resolver o problema utilizando a solução. Ao final do processo de aprendizado é obtido uma solução, e, para validar se essa solução é generalista, são apresentados novos dados, que não participaram do treinamento, sendo chamado de conjunto de validação. Ainda, ao comparar diferentes métodos de aprendizado supervisionado resolvendo os mesmos problemas, pode-se definir ainda um conjunto de teste, separado dos demais.

3.3 Computação Evolucionista

Dentre os diversos algoritmos de otimização no escopo da Inteligência Artificial, destaca-se a subárea da Computação Evolucionista (CE) (De Jong, 2010). As metaheurísticas desenvolvidas em CE são conhecidas como Algoritmos Evolutivos (AEs), que constituem um conjunto de métodos de busca e otimização inspirados biologicamente e fundamentados na teoria darwiniana da seleção natural (Darwin, 1859). Os AEs operam, em geral, como algoritmos que promovem a melhoria de soluções por meio de operações sobre uma população de candidatos ao longo de diversas iterações. Essas operações geram novas soluções baseadas nas anteriores, com a tendência de adaptação ao problema, através das pressões de seleção, onde as soluções de maior aptidão perpetuam seus genes, refletindo o processo observado na seleção natural.

Os AEs geralmente iniciam o processo evolutivo criando uma população inicial utilizando alguma estratégia, que pode gerar indivíduos de forma aleatória ou heurística. Essa população contém diversos candidatos a solução do problema, cada um com suas características contidas nos seus genes, e também uma medida de qualidade da solução que define a aptidão. Cada iteração do algoritmo é chamada de geração, durante a qual ocorrem os processos de seleção, reprodução, modificação via mutação e substituição da população.

A seleção é o processo de escolher dentre as soluções candidatas aquelas que irão reproduzir e gerar filhos, perpetuando seu material genético. Em geral a seleção envolve a competição entre soluções para determinar quais terão acesso à reprodução, sendo geralmente aquelas mais adaptadas ao problema as selecionadas, propagando seu material genético e suas características. A produção de novas soluções é conhecida como reprodução, que geralmente se baseia na reprodução sexuada, onde um par de pais selecionados gera filhos com a mistura de seus materiais genéticos. Contudo, em alguns casos, essa reprodução pode envolver apenas um ou mais de dois pais. A modificação das soluções é realizada por meio da mutação, um processo aleatório que ocorre na natureza e que introduz variabilidade na população. A pressão de seleção favorece a reprodução de indivíduos com maior qualidade de solução, enquanto a mutação adiciona novas informações ao espaço de busca, levando a uma gradual adaptação das soluções ao longo das gerações.

A representação das soluções nos indivíduos pode variar de um AE para outro, dependendo do tipo de problema e das capacidades do algoritmo em questão. As representações podem variar desde um único número inteiro ou em ponto flutuante até listas de valores e estruturas mais complexas, como árvores e grafos, que oferecem maior capacidade de representação. Problemas menos complexos como o ajuste de parâmetros de modelos geralmente são representados usando valores que são posteriormente mapeados no modelo desejado. Problemas de permutação irão apresentar uma representação com uma lista de valores únicos. Já problemas mais complexos irão demandar uma capacidade

de representação mais robusta.

As diversas classes de problemas a serem resolvidos, na prática levam ao surgimento de AEs com características variadas. Desde os mais básicos, como os Algoritmos Genéticos (AG), até os mais complexos, como a Programação Genética (PG). Existem AEs que são aptos a lidar com problemas numéricos, como a Evolução Diferencial (ED), e outros que geram circuitos lógicos através de grafos, como a Programação Genética Cartesiana (PGC) (Möller et al., 2023).

Embora muitos métodos utilizem as características básicas de um AE, alguns apresentam particularidades únicas. A Estratégia Evolutiva (EE) (Arnold and Beyer, 2003; Sharifipour et al., 2018; Motta et al., 2018), em sua variação (1 + 1), por exemplo, possui uma população de tamanho um, mantida a cada geração. Além disso, a geração de novas soluções depende de apenas um pai nesse tipo de variação. Em outros cenários, existem métodos que utilizam exclusivamente a mutação, dispensando a recombinação.

3.3.1 Algoritmo Genético

Um dos mais simples algoritmos evolutivos, e amplamente utilizado, é o Algoritmo Genético (AG). Desenvolvido na década de 1960 e formalizado por Holland (1992), um AG é um Algoritmo Evolutivo que incorpora os principais operadores evolutivos: seleção, cruzamento, mutação e substituição. Tipicamente, as soluções são codificadas em um indivíduo por meio de uma estrutura específica do problema, como um vetor, uma matriz ou outra estrutura de dados. Em geral, os AGs lidam com cromossomos de valores, frequentemente binários ou numéricos. Esse vetor de valores é chamado de genótipo, que carrega a informação genética do indivíduo, enquanto a manifestação dessa informação é denominada fenótipo.

É comum utilizar AGs para otimizar funções, minimizando o erro obtido ao avaliá-las sobre um conjunto de dados. Geralmente essa otimização envolve o ajuste dos coeficientes numéricos de funções, como por exemplo os coeficientes a, b e c na função $F(x) = ax^2 + bx + c$. Uma representação comum de uma solução num AG é o vetor de genes $[a, b, c]$, compondo o genótipo dessa solução. O objetivo da busca é maximizar a aptidão dos indivíduos, o que geralmente significa minimizar o erro calculado através da função objetivo $f(x)$, como a definida na Equação 3.1, onde $F(x)$ representa os resultados originais da função F, F_i representa a a execução da função representada pelo i-ésimo indivíduo da população sob o conjunto $x = [x_1, ..., x_n]$. A população de indivíduos de um AG é exemplificada na Equação 3.2, onde cada indivíduo i da população tem como genótipo $[a_i, b_i, c_i]$. O fenótipo de uma solução é obtido ao avaliar o resultado de $F(x)$ com as variáveis do indivíduo, resolvendo $F_i(x) = a_i x^2 + b_i x + c_i$ sob um conjunto de dados, e em seguida calculando a aptidão através da função objetivo.

$$f_i(x) = \sum_{j=1}^{n} \|F(x_j) - F_i(x_j)\| \tag{3.1}$$

$$\text{população} = \begin{cases} F_0(x) = a_0x^2 + b_0x + c_0 & \rightarrow & [a_0, b_0, c_0] \\ \cdots & & \\ F_i(x) = a_ix^2 + b_ix + c_i & \rightarrow & [a_i, b_i, c_i] \\ \cdots & & \\ F_n(x) = a_nx^2 + b_nx + c_n & \rightarrow & [a_n, b_n, c_n] \end{cases} \tag{3.2}$$

O Algoritmo 1 ilustra o funcionamento padrão de um AG. A população de soluções candidatas é geralmente inicializada com indivíduos aleatórios ou gerados através de uma heurística, com valores dentro de um intervalo para cada variável. Após a avaliação dessa população, começa o processo iterativo geracional, onde na geração corrente são criados novos indivíduos baseados na geração anterior.

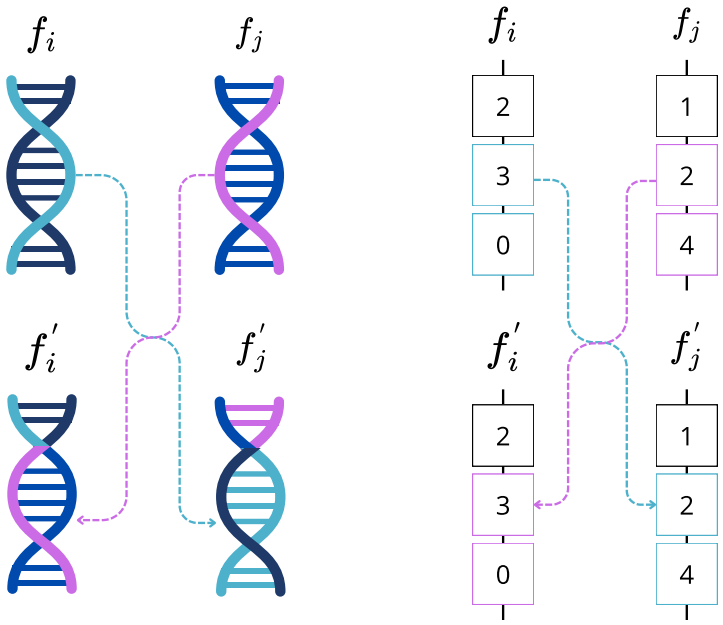
Algoritmo 1: Algoritmo Genético

```
1  Inicializar a população;
2  Avaliar a aptidão de cada indivíduo;
3  while Critério de Parada não atendido do
4  |  while Há espaço na nova população do
5  |  |  Selecionar pais para reprodução;
6  |  |  Aplicar operador de recombinação;
7  |  |  Aplicar operador de mutação nos filhos;
8  |  end while
9  |  Avaliar a aptidão da nova população;
10 |  Selecionar a próxima geração;
11 end while
12 return Melhor solução encontrada na população final;
```

Para gerar novos indivíduos, primeiro ocorre um processo de seleção, onde são escolhidos na população aqueles com melhor aptidão, que, por seleção natural, tem maior propensão a gerar bons descendentes. Essa seleção envolve a competição entre dois ou mais indivíduos, onde aquele que atende aos critérios de seleção participará da recombinação. Os critérios geralmente priorizam o indivíduo com melhor aptidão, mas podem incluir outros aspectos, como para desempate ou em otimizações multiobjetivo.

Os indivíduos selecionados participam da recombinação, um operador que cria novos indivíduos baseados no genótipo de outros chamados pais, gerando novas soluções a partir dos genes dos mais aptos. É importante ressaltar que, de forma geral, todos os indivíduos têm a oportunidade de acesso à seleção, garantindo assim a chance de reprodução, mesmo que com baixa probabilidade. A recombinação é a troca de informação genética, exemplificada na Figura 3. Utilizando o exemplo anterior, a reprodução dos indivíduos F_i e F_j implica a troca de material genético entre eles, ou seja, a troca de valores

Figura 3 – Recombinação em um AG.



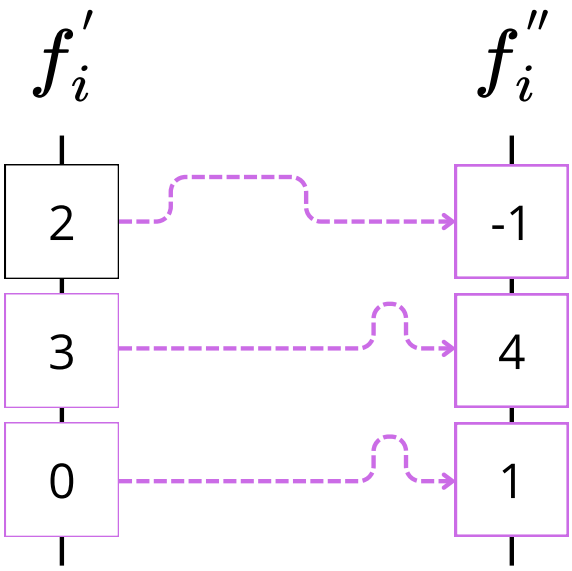
Fonte: Elaborado pelo próprio autor (2024).

dos vetores de representação. Existem diversas estratégias de recombinação, como o *Single-Point Crossover* (exemplificado na figura), onde é definido um ponto de corte do genótipo e feita a troca dos trechos após o ponto entre os pais. De forma similar, o *Two-Point Crossover* realiza a troca no trecho entre dois pontos de corte, enquanto o *Multi-Point Crossover* utiliza múltiplos pontos. O *Uniform Crossover* seleciona aleatoriamente de qual pai o filho herdará informação a cada ponto de seu gene. Esses operadores se aplicam a problemas de domínios booleano, inteiro, real ou simbólico. O *Arithmetic Crossover* realiza cálculos para definir o valor do gene em cada ponto, como a média aritmética entre os valores dos pais, sendo aplicável a problemas de domínio real. Além disso, existem métodos específicos para problemas de permutação, como o PMX (*Partially Mapped Crossover*), que garante uma solução factível mesmo após a operação. Em geral, a partir de um par de pais é gerado um par de filhos, onde seus códigos genéticos têm contribuição de cada um dos pais. Em casos específicos pode-se ocorrer a troca total de genes, como no *Single-Point Crossover* selecionando o início do vetor de representação como ponto de troca. No entanto, esses casos podem não ser ideais, pois apenas geram cópias dos pais, o que aumenta a representatividade de seus genes porém não contribuem para a diversidade genética da população.

O próximo passo é a aplicação do operador de mutação. Esse operador gera alterações no genótipo de um indivíduo com o objetivo de provocar uma perturbação e reposicioná-lo no espaço de busca. A mutação é responsável por introduzir variabilidade e até mesmo novas informações ao *pool genético* da população, posicionando um indivíduo

em um novo ponto do espaço. O *pool genético*, ou reserva genética, se refere a toda a diversidade genética disponível dentro da população, incluindo todos os possíveis genes que podem ser transmitidas para gerações futuras. Por exemplo, considerando o problema anterior, se os valores únicos de a_i existentes na população atual gerados na população inicial forem positivos, e o valor ideal for $a_i = -1$, não será possível obtê-lo através de recombinações, pois -1 não existe no espaço de busca. Contudo, através da mutação, é possível realizar uma perturbação nos valores de a existentes que gere o valor -1, disponibilizando-o no espaço de busca.

Figura 4 – Mutação em um AG.

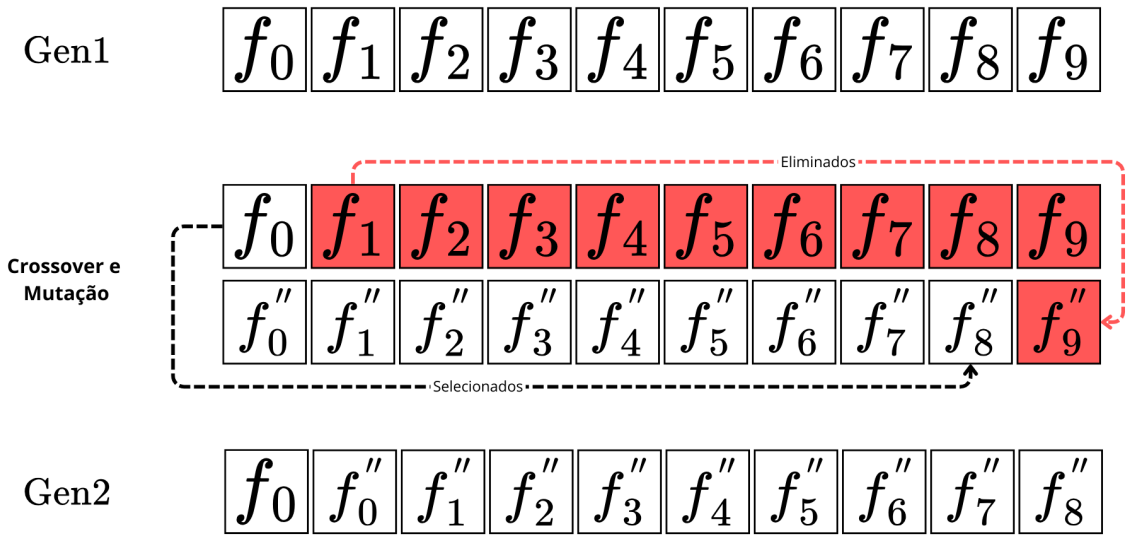


Fonte: Elaborado pelo próprio autor (2024).

Ao final de cada geração, ocorre a substituição da população da geração atual pelos novos indivíduos. Diversas estratégias podem ser adotadas nesse processo. Uma estratégia básica é substituir toda a população antiga pelos novos indivíduos. No entanto, essa abordagem pode resultar em piora entre gerações que pode levar a um processo de busca mais demorado em direção a um ótimo. Uma estratégia que visa evitar essa possibilidade é o elitismo, onde uma elite da população corrente, representada pelos indivíduos de melhor aptidão, é mantida no lugar dos piores da nova população.

A Figura 5 exemplifica esse processo em uma população de tamanho 10 e uma elite de 10%. A população da geração 1 é representada por $Gen1$, e está ordenada de forma crescente em relação à aptidão dos indivíduos. Em seguida são aplicados os operadores gerando uma nova população também de tamanho 10, que também está ordenada. Então, 10% dos melhores indivíduos de $Gen1$ entram no lugar dos 10% piores da nova população. Os demais, representados em vermelho na imagem, são descartados. Os restantes irão

Figura 5 – Seleção em um AG.



Fonte: Elaborado pelo próprio autor (2024).

compor a nova população $Gen2$.

Essa iteração de gerações se repete até que um critério estabelecido seja atendido. Esse critério pode variar de acordo com o objetivo desejado. Em casos onde se busca encontrar a melhor resposta possível, a satisfazibilidade da solução pode ser um critério. Quando o tempo de execução é limitado, o tempo de execução pode ser utilizado como critério de parada. Para realizar comparações, é comum utilizar uma quantidade máxima de cálculos da função objetivo, permitindo comparações mesmo com parâmetros diferentes, como diferentes tamanhos de população.

Quando o critério de parada é atingido, obtém-se uma população que contém os melhores indivíduos gerados na busca guiada pelos dados de treinamento. Geralmente, essa população é avaliada em outros conjuntos de dados, como os conjuntos de teste e validação, que calculam o aptidão em dados não apresentados durante o treinamento. A resposta da busca é frequentemente o melhor indivíduo em relação ao treinamento ou validação, sendo então avaliado no teste.

Uma variação dos AGs para problemas mais complexos é a Programação Genética (PG). A PG é um AG que tem como tarefa evoluir programas para resolver tarefas. Um programa é uma estrutura mais complexa, com maior capacidade de representação. Essa complexidade é incorporada desde a representação das soluções até os operadores genéticos. Além disso, variações de PG, como a PG Gramatical, foram desenvolvidas com o objetivo de auxiliar a descoberta de conhecimento.

3.4 Programação Genética

Com a intenção de utilizar AGs para encontrar estruturas mais complexas, criando programas de computador mais complexos e principalmente sem a necessidade de estabelecer um formato fixo, no final da década de 1980 seguindo o trabalho de Holland, seu estudante John Koza desenvolve a Programação Genética (PG) e formaliza em seu livro Koza (1992). A PG é a especialização de um AG, com seu funcionamento base muito similar: inicializa uma população, gera uma nova população atrávés da recombinação e mutação, substitui a anterior aplicando elitismo, itera as gerações até o critério de parada e retorna o melhor indivíduo.

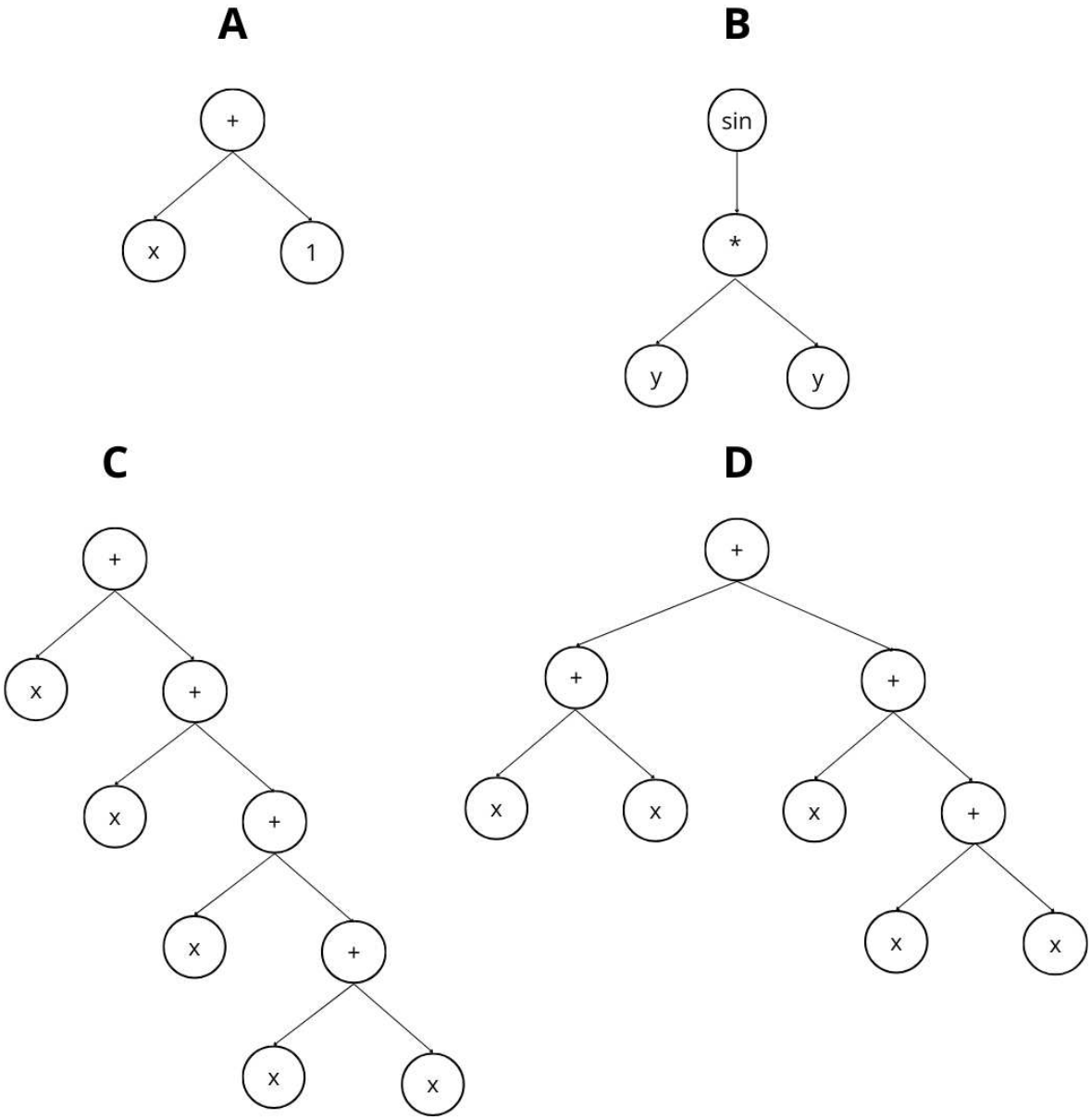
A diferenciação da PG é a capacidade de gerar soluções mais complexas na forma de programas, e também a não necessidade de explicitamente programar as soluções como em um AG. Diversas soluções de problemas podem ser representadas como programas, como funções aritméticas, até programas escritos numa linguagem de programação como Python, Java e C++ (de Freitas et al., 2018). Para isso, a PG conta com uma representação mais robusta dos problemas codificados nos indivíduos em seu genótipo. Além disso os operadores de recombinação e mutação são adaptados para funcionar com essa nova representação, que impõe novas regras.

Em Forsyth (1981) é apresentado o primeiro trabalho que tenta evoluir programas simples baseado em árvores para classificação de cenas de crime. Com a mesma inspiração, a representação clássica de programas em PG utiliza árvores. Primeiro são definidos dois conjuntos: *operadores* como o conjunto de operações permitidas, como o conjunto $[+, -, \times, \div, ln, tanh]$, e *terminais* como o conjunto de variáveis e valores numéricos que servem como parâmetros dos operadores, como $[0, ..., 9, x_1, ..., x_N]$, para um problema com N variáveis. Nesse exemplo é possível criar programas que representam operações aritméticas utilizando esses operadores e terminais. As árvores, como exemplificado na Figura 6 contém nós internos e folhas, sendo respectivamente valores dos conjuntos de operadores e terminais. Assim, é possível executar um programa realizando uma avaliação recursiva da árvore.

A recombinação em PG tem o mesma função como no AG: gerar novos indivíduos baseado no material genético de pais selecionados. Porém com uma representação mais robusta um operador mais complexo precisa ser empregado. No caso de uma PG clássica, a recombinação envolve a troca de subárvores dos pais para gerar os filhos. A Figura 7 exemplifica a recombinação entre as árvores das expressões $A = x + 1$, $B = \sin(y^2)$, onde um nó das árvores é selecionado em ambos os pais, e em seguida são trocados, gerando dois novos indivíduos A^1 e B^1 com genes de ambos os pais.

A mutação também é afetada, mas de forma muito similar à recombinação é orientada a subárvores. Um nó da árvore é selecionado e sua subárvore é substituída por uma outra gerada aleatoriamente. Esses operadores têm características intrínsecas como a

Figura 6 – Representação de programas em árvores na PG. As árvores representam as expressões numéricas: $A = x + 1$, $B = \sin(y^2)$, $C = 5x$ e $D = 5x$.

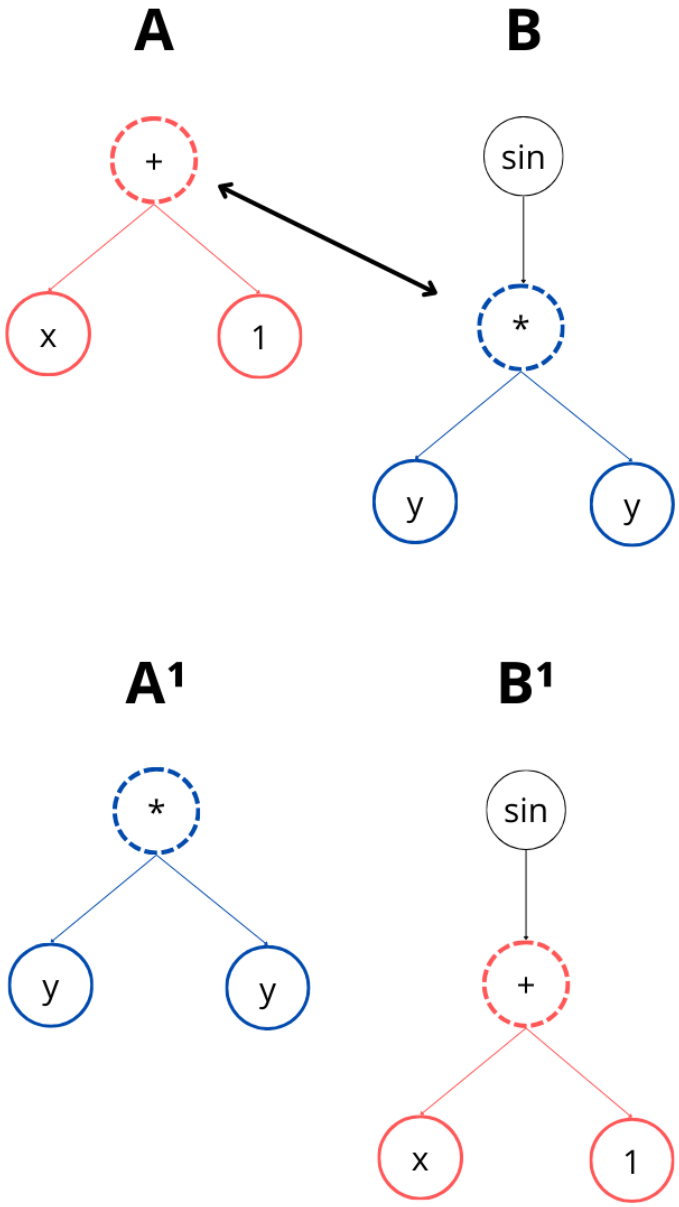


Fonte: Elaborado pelo próprio autor (2024).

capacidade de realizar modificações finas, por exemplo alterando nós próximos às folhas, ou mais fortes alterando próximo à raiz da árvore. Assim como na recombinação, é gerada uma nova subárvore para substituir aquela com raiz no nó selecionado.

A representação em um AG compreende toda solução, sendo geralmente linear e de tamanho fixo, enquanto representações mais robustas como a árvore na PG já apresentam tamanho e forma variáveis. Dessa forma é possível que os indivíduos tenham tamanhos diferentes, sendo necessário tomar cuidados adicionais para evitar problemas como o crescimento indiscriminado das árvores (*bloating*). Uma forma comum de evitar o *bloating* tanto nas trocas de subárvores na recombinação quanto na mutação, é limitar a altura

Figura 7 – Operadores de Recombinação na PG. A recombinação de A e B gerou novas árvores com as expressões $A^1 = y^2$ e $B^1 = sin(x + 1)$.

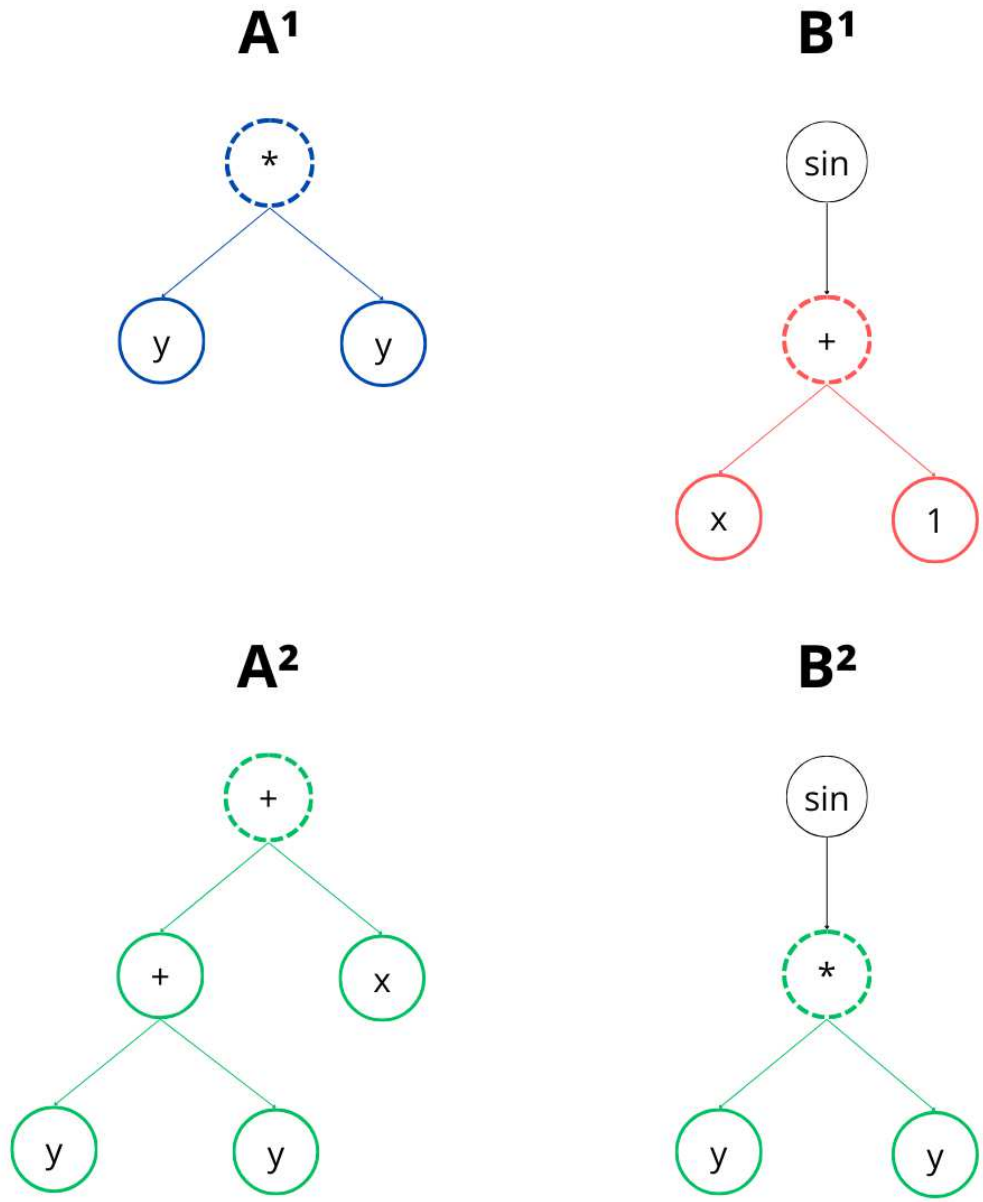


Fonte: Elaborado pelo próprio autor (2024).

máxima da árvore durante a operação. Os outros processos continuam os mesmos de um AG, a seleção de pais para a recombinação, a substituição da população a cada geração, critério de parada, sento todos processos não dependentes da representação.

AEs geralmente são adaptados para resolver problemas específicos, como utilizar operadores genéticos apropriados para o problema. Um exemplo claro é, ao resolver problemas onde a solução é uma permutação, é mais interessante utilizar recombinações que levam em consideração a natureza do problema como o *Partially Mapped Crossover* (PMX) (Scholz, 2019), desenvolvido para resolver problemas de otimização que envolvem permutações. Porém em alguns casos, ocorrem modificações mais expressivas, como

Figura 8 – Operadores de Mutação na PG.



Fonte: Elaborado pelo próprio autor (2024).

no caso da *Evolução Diferencial* (ED) (Wong and Dong, 2007) que é desenhada para resolver o ajuste de valores numéricos, e tem operadores de recombinação e mutação desenvolvidos especificamente para trabalhar sob este domínio de forma mais eficiente, além de estratégias de seleção e substituição da população que trazem um maior balanço entre o custo computacional e a exploração do espaço de busca.

Com a PG não é diferente, existem variações com diferentes características dependendo do propósito. Uma das vertentes da PG, cujo objetivo é tomar mais controle sobre o espaço de busca é a *Evolução Gramatical* (EG) (Ryan et al., 1998), onde são aliados os conceitos de PG com uma estrutura linear como a de um AG, gerando um programa através de uma gramática formal. Sua ideia base é manipular uma palavra composta de

valores inteiros assim como um AG que guiam o processo de derivação de um programa a partir de uma linguagem definida em uma gramática formal. Dessa forma, é possível a criação de programas complexos, que é o objetivo da PG, com a implementação do método de busca mais simples do AG associado à derivação dos programas guiada pela gramática. Variações como PG Linear (Markus F. Brameier, 2007), PG Cartesiana (Miller et al., 1997) entre outras, surgiram para atender a problemas com características específicas como a criação de programas em linguagem imperativa e geração de circuitos lógicos. Seguindo a mesma linha, chega-se na *Programação Genética Gramatical* (PGG), que assim como a EG é uma técnica de PG, mas que se especializa mais para lidar com o conceito de gramáticas de forma intrínseca.

3.5 Programação Genética Gramatical

A criação automática de programas utilizando PG trouxe grande avanço para a área de CE, mas ainda haviam espaços a serem explorados dentro da PG, sendo um desses a geração de programas inválidos. Um exemplo de solução inválida é a aplicação de um parâmetro incompatível com o operador, por exemplo seno de um valor booleano, a soma de textos que são problemas de sintaxe, ou simplesmente uma raiz par de um valor negativo sendo um problema de semântica. Para solucionar esse tipo de problema foi incorporado à PG as *Gramáticas Formais*, criando a EG. No entanto, dentro da EG também havia espaço para melhorias, como melhorar a sensibilidade de operações de recombinação e mutação que geram alterações muito bruscas nos programas, que são gerados via derivação através da gramática.

Contornando esses problemas surge a estratégia da PGG (McKay et al., 2010), que se difere de uma PG clássica pelo uso de gramáticas formais e especialmente na representação do programa no indivíduo. Na PGG, a representação passa a ser uma árvore de derivação. Os operadores genéticos são adaptados para funcionar com essa representação e obedecer a suas regras, e o mecanismo de busca continua sendo o mesmo de um AG.

3.5.1 Gramáticas

Dentro da Ciência da Computação, as *Gramáticas Formais Livres de Contexto* (GLC) são ferramentas chave para a criação e representação de estruturas. Elas são capazes de definir uma linguagem a partir de um conjunto de regras, formando um certo domínio, definido por um subconjunto composto somente por expressões sintaticamente válidas. As GLCs são amplamente utilizadas para impor restrições e definir estrutura de sentenças (McKay et al., 2010), se tornando popular em PG com a EG.

A GLC atua como uma geradora de sentenças que obedece a um conjunto de regras

estipuladas, e é definida pela quádrupla $G = (N, \Sigma, S, P)$, onde:

N : alfabeto de não terminais (elementos auxiliares da gramática)

Σ : alfabeto de terminais (elementos que compõem os programas)

S : regra inicial $S \in N$

P : conjunto de produções (possibilidades de derivação de cada regra)

Cada elemento de N é um não terminal da gramática, e possui um conjunto de possibilidades de derivação chamadas de produções. Uma ilustração das produções de uma GLC pode ser como:

$$< NT > ::= prod1 \mid prod2 \mid prod3$$

onde $< NT >$ é um não terminal, e $prod1$, $prod2$ e $prod3$ são suas produções. Isso significa que, em uma sentença sendo derivada, o não terminal $< NT >$ pode ser substituído por qualquer uma de suas produções. Cada produção pode ser composta por combinações de elementos de N e Σ, e são realizadas substituições de regras por produções até que a sentença seja formada somente por símbolos de Σ.

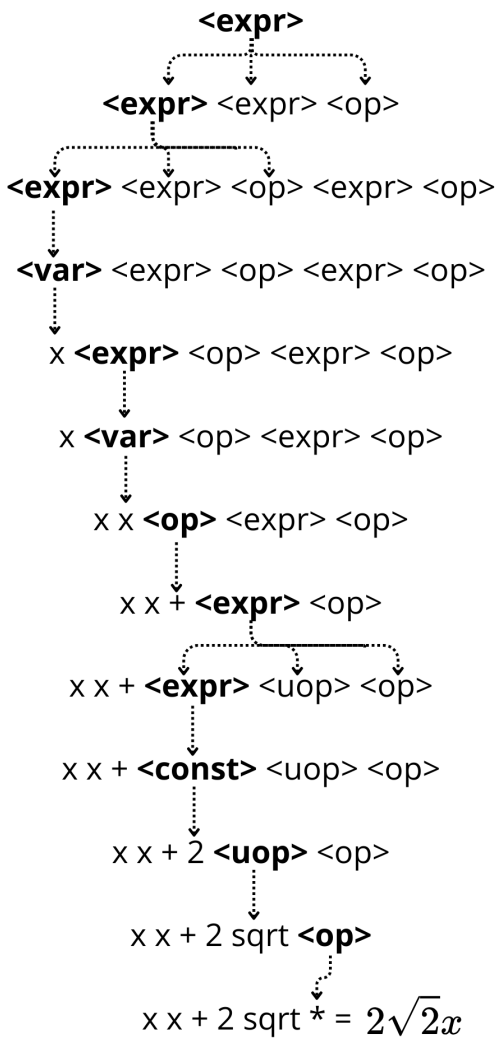
A gramática a seguir exemplifica um conjunto de regras que definem uma linguagem de programas para a geração de expressões aritméticas posfixadas:

$$\begin{aligned}
<expr> &::= <expr><expr>< op > \mid <expr>< uop > \mid < var > \mid < const > \\
< op > &::= + \mid - \mid * \mid / \mid pow \\
< uop > &::= log \mid exp \mid sqrt \mid sen \mid cos \mid tg \\
< var > &::= x \mid y \mid z \\
< const > &::= 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \mid \pi
\end{aligned}$$

3.5.1.1 Representação

Ao adotar uma representação em estrutura de árvore combinada com GLC, os programas passam a ser árvores de derivação, onde os nós internos são não terminais de N e as folhas terminais de Σ. Devido a suas características, esta representação foi escolhida para a PGG, trazendo ganho de estrutura para as operações genéticas. A derivação de um modelo a partir de uma GLC é ilustrada na Figura 9. A partir do não terminal inicial (<*expr*>), uma de suas produções é escolhida aleatoriamente, e cada um de seus elementos tornam-se os nós filhos da raiz da árvore de derivação. Em seguida, para cada não terminal gerado é feito uma nova derivação seguindo o mesmo procedimento. Esses passos se repetem até que seja gerado um nó terminal (folha), encerrando a recursão da gramática. Neste exemplo são necessários 12 passos para obter um modelo.

Figura 9 – Exemplo da derivação utilizando a GLC.



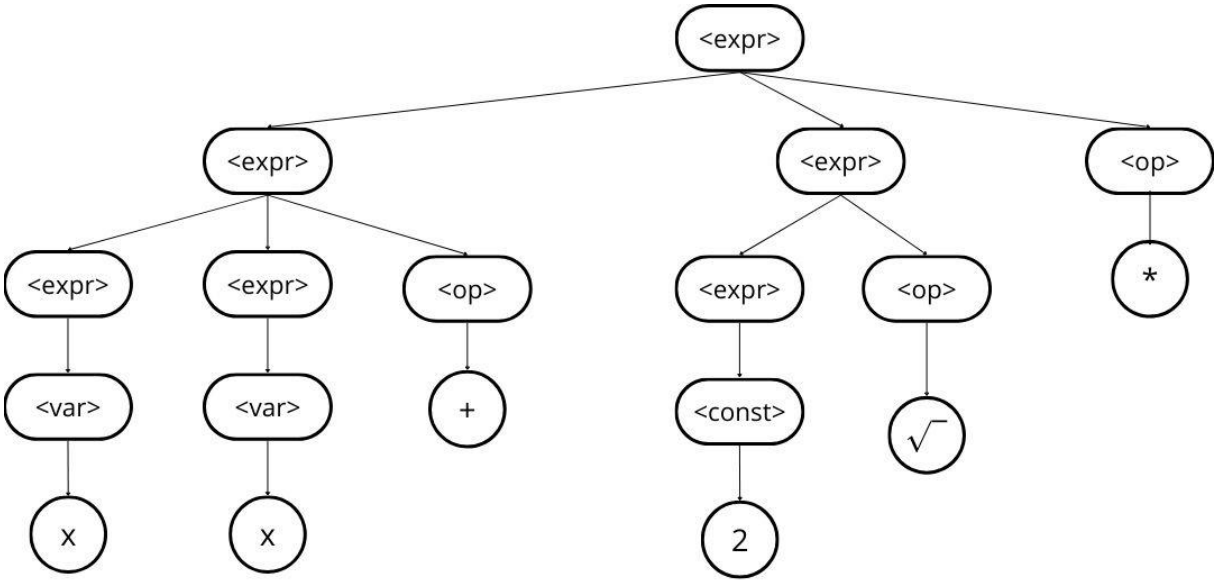
Fonte: Elaborado pelo próprio autor (2024).

3.5.1.2 Operadores Genéticos

Os operadores genéticos na PGG são derivados de operadores da PG, seguindo a mesma lógica mas com particularidades dependentes da representação do indivíduo e das regras da GLC utilizada. De forma geral, as operações podem ser aplicadas somente nos nós internos das árvores, ou seja, em não terminais da gramática. Os operadores são orientados pela regra do nó escolhido, seja para realizar uma recombinação ou uma mutação.

O primeiro operador aplicado é a recombinação. Originalmente, em PG, a recombinação seria a troca entre duas subárvores aleatórias, mas qualquer troca de subárvores é válida. Já na PGG a recombinação deve obedecer às regras da GLC, assim como outras restrições como a altura máxima da árvore. Na recombinação entre as árvores A e B, é primeiro selecionado um nó não terminal NT $nt_i \in A$ e toda sua subárvore. Em seguida

Figura 10 – Exemplo da geração de um modelo ao acaso usando uma GLC.



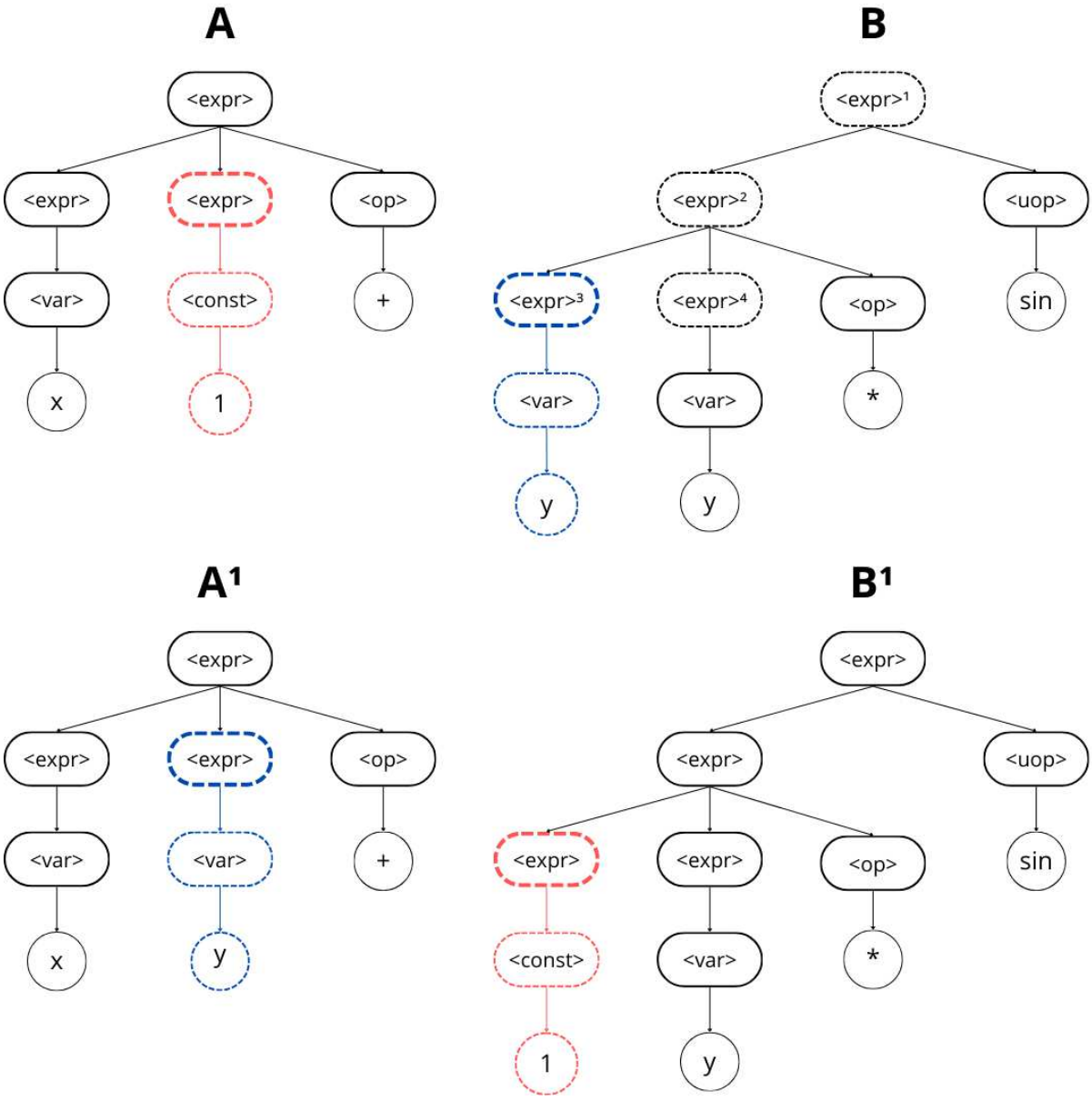
Fonte: Elaborado pelo próprio autor (2024).

todas as subárvores de B com raiz $NT = nt_i$ são identificados e um deles escolhido para realizar a recombinação. Isso garante que, independente da troca, obedecendo a regra definida na gramática sempre é obtido um par de filhos A' e B' válidos. Caso algum tipo de lógica seja embutida na gramática, seu efeito será mantido. Por exemplo, pode-se garantir que o modelo seja sempre linear, impedindo que variáveis estejam no expoente de operações de potência. A Figura 11 exemplifica a recombinação entre dois indivíduos.

De forma similar, a mutação também é guiada pela gramática e deve obedecer às mesmas regras e restrições que a recombinação. Para aplicar a mutação em uma árvore é selecionado um não terminal $nt_i \in A'$ e toda sua subárvore, que são removidos da árvore. Utilizando a GLC é gerada uma nova subárvore a partir da regra nt_i. Por ser gerada a partir da mesma subárvore, a garantia de validade sintática é mantida, e outras restrições como altura máxima também deve ser mantida. A Figura 12 exemplifica a mutação de um indivíduo.

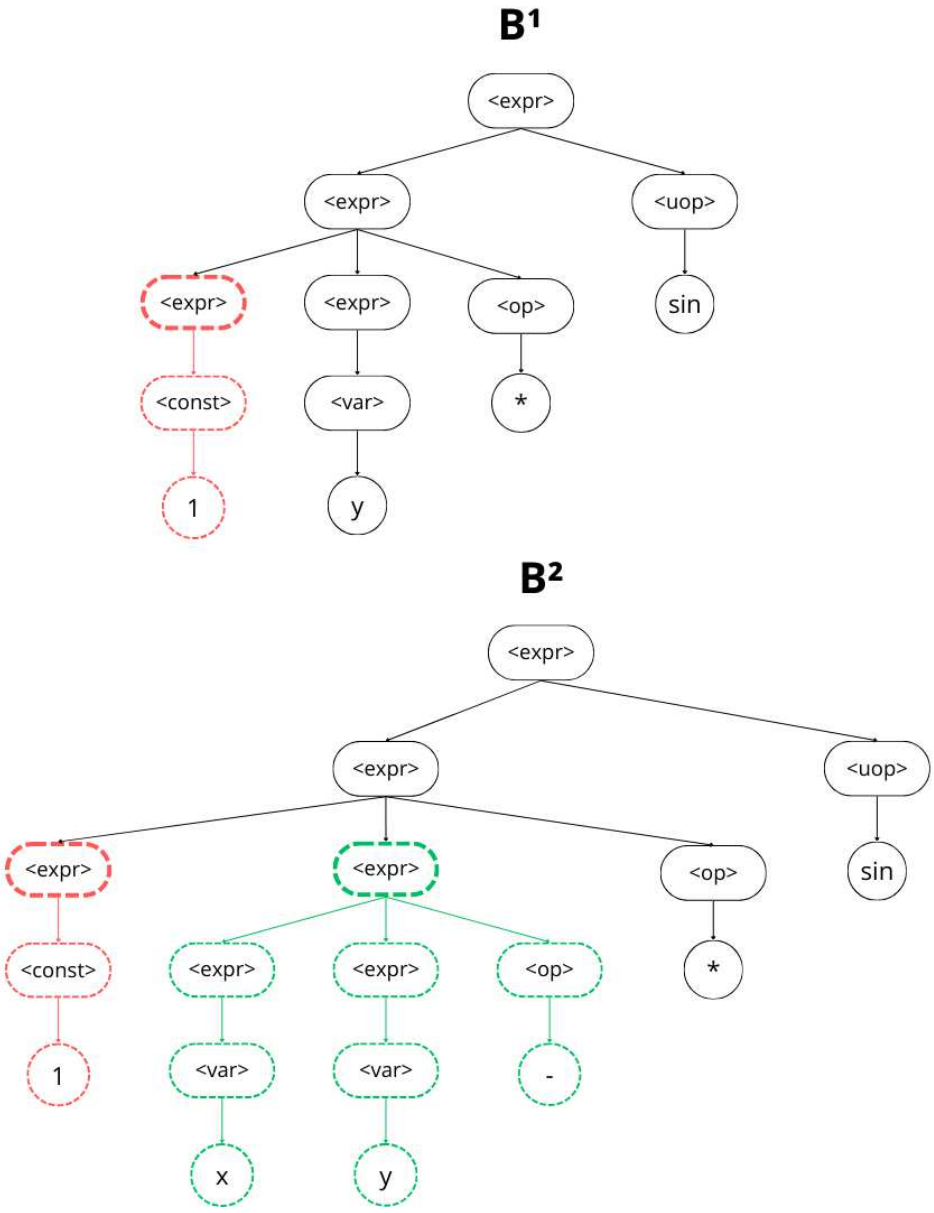
Alguns detalhes devem ser observados ao realizar as operações de recombinação, mutação e até mesmo a geração da população inicial. As árvores têm uma altura limitada para evitar que as árvores cresçam de forma desenfreada gerando programas com um ajuste excessivo aos dados ($overfit$), perdendo a capacidade de generalização. Dessa forma, pode-se favorecer uma maior generalização do modelo e garantir que há memória suficiente para as computações. Ao mesmo tempo que uma altura muito pequena pode limitar muito o modelo que não conseguirá generalizar o problema de modo satisfatório.

Figura 11 – Exemplo de Cruzamento na PG.



Fonte: Elaborado pelo próprio autor (2024).

Figura 12 – Exemplo de Mutação na PG.



Fonte: Elaborado pelo próprio autor (2024).

4 PROGRAMAÇÃO GENÉTICA SEMÂNTICA

Este capítulo define os conceitos de semântica, apresenta a Programação Genética apoiada por Semântica, e explora a literatura de operadores de recombinação adaptados para lidar com a semântica.

4.1 Semântica

O conceito de semântica faz parte de diferentes áreas de conhecimento como Psicologia, Linguagem Natural e Ciência da Computação (Uy et al., 2010). Associado à semântica, geralmente vem acompanhado o conceito de sintaxe onde, de forma simplificada, a primeira define o significado de uma sentença enquanto a outra sua forma. Em Ciência da Computação pode-se interpretar a sintaxe como a sequência de instruções que geram um programa correto, e a semântica como seu significado, ou, seu resultado.

A relação entre semântica e sintaxe é particular, onde entre dois programas com mesma sintaxe obtemos sempre a mesma semântica. Porém é uma relação que pode não ser recíproca, pois nem sempre a mesma semântica é obtida por sintaxes idênticas. Por exemplo as Equações 4.1 e 4.2, apesar de apresentarem sintaxes diferentes, possuem a mesma semântica, pois independente do conjunto de dados em que sejam avaliados produzem o mesmo resultado.

$$y = x \times x \tag{4.1}$$

$$y = x^2 \tag{4.2}$$

Três grandes grupos de semântica aplicada a PG podem ser identificados, em relação à forma como ela é aplicada, como a semântica baseada em gramáticas, em métodos formais ou em representação, abordagens utilizadas para inserir ou extrair semântica dos modelos (Uy et al., 2010).

As gramáticas de atributos são extensões de GLCs (Knuth, 1968), onde um conjunto finito de atributos utilizados para incorporar informações semânticas diretamente na representação dos indivíduos. Essa incorporação de semântica permite eliminar indivíduos de menor aptidão da população ou prevenir a geração de indivíduos semanticamente inválidos. No contexto de PG, os indivíduos das árvores geradas pela gramática de atributos podem utilizar essas informações semânticas para guiar o processo evolutivo. Porém a utilização de atributos para incorporar a semântica geralmente é dependente do problema em questão, e não é sempre fácil como determinar atributos adequados para um problema (Wong and Leung, 1995; Cleary and O'Neill, 2005).

A incorporação da semântica através de Métodos Formais é feita através de técnicas matemáticas, geralmente utilizadas para a especificação, desenvolvimento e verificação de sistemas de software e hardware (Johnson, 2002, 2004, 2007). Esses métodos suportam a extração e aproximação de declarações matemáticas úteis para o *design* e verificação de sistemas. A informação semântica extraída por métodos formais é utilizada para quantificar a aptidão dos indivíduos em problemas onde as medidas de aptidão tradicionais baseadas em amostras são indisponíveis ou enganosas. Em (Keijzer, 2003), é aplicado como a análise de intervalos, verificando se um indivíduo está definido em todo intervalo dos valores de entrada, e caso não esteja definido em algum lugar é atribuído uma aptidão mínima. Assim, a semântica é utilizada para fornecer orientação adicional ao processo evolutivo.

A semântica aplicada à representação é também muito comum em PG, onde a semântica de um programa é obtida a partir de sua representação, como por exemplo em árvores. Assim como com gramáticas de atributos, a semântica é dependente do problema, mas aqui pode ser mais generalizada para algumas classes de problemas, como booleanos, inteiros, reais, discretos, entre outros. Inicialmente testado com problemas no domínio booleano, a semântica foi aplicada em operações de recombinação onde os indivíduos eram transformados em *Reduced Ordered Binary Decision Diagrams* (ROBDDs) (Bryant, 1986). A árvore é avaliada e extraída dela sua forma ROBDD, possibilitando comparar o indivíduo de forma objetiva. Caso duas árvores tenham o mesmo ROBDD, mesmo que com estrutura (sintaxe) diferente, são semanticamente equivalentes. Nesse caso ao recombinar pais, seus filhos são comparados e verifica-se se são equivalentes aos pais, e caso sejam, são descartados e a recombinação ocorre novamente. Foi destacado que esse processo aumentou a diversidade da população, levando a uma melhor desempenho da PG. Assim como no caso booleano, outras categorias de problemas podem ter sua sintaxe incorporada na representação, como veremos nos operadores de recombinação para problemas de valor real. Assim, a incorporação da semântica na PG não apenas assegura a correção sintática para a construção de programas eficazes.

Além das estratégias de atribuição da semântica aos indivíduos, existem diferentes abordagens do uso da semântica no mecanismo de busca, cada uma com um efeito e uma finalidade. Quando é aplicado na seleção dos indivíduos que irão participar da próxima população, pode-se obter um efeito de uma população que não é composta apenas pelos melhores modelos, mas modelos que são diferentes entre si em relação a seu resultado, o que leva a uma maior variedade de soluções real. A variedade é real pois uma variedade genética pelo ponto de vista sintático pode existir, com diferentes genótipos presentes na população, no entanto é possível que muitos deles apresentem uma semântica similar ou até mesmo igual, como no caso das equações 4.1 e 4.2. Ao ser aplicado na operação de mutação, pode-se obter um efeito de gerar mutações de maior significado (Beadle and Johnson, 2009). Através dessa abordagem é possível gerar mutações que gerem uma perturbação grande

o suficiente para ser expressiva, e gerar uma alteração real no fenótipo de um indivíduo, evitando que o esforço computacional gere resultados irrelevantes na operação. Também pode ser possível controlar mutações mais agressivas que geram maiores perturbações, explorando de forma mais bruta o espaço de busca, ou uma mutação mais tênue explorando o espaço de busca de forma mais refinada. Ao ser aplicado na recombinação, podem existir diferentes abordagens em qual momento aplicar a semântica. Ao ser utilizado no passo de selecionar os indivíduos para operar, é possível encorajar que a operação seja realizada entre indivíduos semanticamente diferentes, onde seus filhos tendem a ter uma diversidade genética maior, melhorando a exploração do espaço de busca. Ao aplicar na escolha dos pontos de recombinação entre os genes dos pais, é possível evitar realizar trocas similares, o que leva a geração de filhos também com maior diversidade genética. Em geral, o objetivo do uso de semântica em PG é aumentar a diversidade da população e explorar o espaço de busca de forma mais inteligente.

4.2 Conceitos

Existem alguns conceitos de semântica, que são utilizados nas operações, gerando um processo de busca guiado pela semântica. O primeiro conceito, e também o mais básico, é o de distância semântica (DS). A ideia da DS é atribuir uma distância entre indivíduos no espaço semântico, calculando a diferença semântica entre os programas. A forma de cálculo da distância pode variar de problema a problema, dependendo do domínio dos dados utilizados. Problemas binários podem utilizar ROBDD, ou quando o domínio das variáveis está em um outro espaço discreto pode levar à uma distância de contagem simples, como a quantidade de valores iguais, ou em casos onde há uma ordem dos valores, a distância de gray ou distância de edição. A DS utilizada aqui é a distância entre dois programas num espaço semântico contínuo de problemas de valor real, definida na Equação 4.3 (Uy et al., 2010) como sendo a média da diferença absoluta das semânticas da dupla.

Sejam $P = [p_1, p_2, ..., p_N]$ e $Q = [q_1, q_2, ..., q_N]$ as semânticas de dois programas, a DS entre elas é definida como

$$DS(P, Q) = \frac{\sum_{i=1}^{N} |p_i - q_i|}{N} \tag{4.3}$$

Outro conceito de semântica é a Sensitividade Semântica (ϵ). A sensitividade ϵ define um valor mínimo de DS para que dois indivíduos possam ser considerados equivalentes ou diferentes em relação à suas semânticas. A partir desse valor, pode-se estabelecer o conceito de Equivalência.

A Equivalência Semântica (ES) define se duas semânticas são equivalentes, ou seja, expressam a mesma informação. Em problemas de valores reais, como na regressão

simbólica, é muito comum o acúmulo de erros numéricos, provocando pequenas diferenças entre operações. Para isso têm-se que

$$ES(P,Q) = \begin{cases} \text{se } DS(P,Q) \leq \epsilon, & \text{equivalentes} \\ \text{senão ,} & \text{não equivalentes} \end{cases} \tag{4.4}$$

A partir da ES é possível evitar operações entre genes que não provocarão alterações significativas, pois suas semânticas são suficientemente iguais. Assim, é possível focar o esforço computacional em realizar operações de maior significado.

A última relação é a Similaridade Semântica (SS), que indica se duas semânticas podem ser consideradas similares. A ideia é utilizar essa informação para promover operações de troca entre programas que não são idênticos mas também não são muito diferentes. Dessa forma, as movimentações passam a ser mais significativas, evitando tanto trocas equivalentes e não muito drásticas.

$$SS(P,Q) = \begin{cases} \text{se } \alpha < DS(P,Q) < \beta, & \text{verdadeiro} \\ \text{senão,} & \text{falso} \end{cases} \tag{4.5}$$

4.3 Operadores

A busca realizada por algoritmos evolutivos é constituída de alguns operadores inspirados em processos da seleção natural. Desses operadores, três são comumente encontrados na maioria dos trabalhos em PG: seleção, recombinação e mutação. Como já explorado anteriormente, a seleção escolhe quais indivíduos irão gerar descendentes; a recombinação utiliza informação genética desses pais para gerar novos filhos; e a mutação gera uma perturbação em uma solução a fim de realocá-la no espaço de soluções, além de introduzir novos genes ao espaço explorado pela recombinação. Esse processo gera novos indivíduos e explora os possíveis genes do espaço de busca, além de manter os indivíduos com pior aptidão na população, participando de mais recombinações e assim perpetuando seus genes através das gerações.

No entanto, essa busca geralmente é focada apenas na aptidão dos indivíduos. Para escolher quem participa da recombinação é comum que torneios entre indivíduos sejam feitos, onde aqueles com maior aptidão são escolhidos. Os filhos gerados nessa operação podem sofrer mutação, independente de sua aptidão, mas ainda assim, sob efeito de uma escolha feita sob a luz apenas da aptidão de seus pais, e ainda gerar um gene muito similar ao que está sendo alterado. A seleção geralmente verifica apenas a aptidão dos indivíduos, mantendo na população aqueles mais aptos entre a elite da geração anterior e os novos indivíduos criados nessa geração. Até mesmo em abordagens multiobjetivo, onde a seleção

pode envolver mais fatores, é comum que os fatores envolvidos sejam valores como aptidão e complexidade.

Para considerar a semântica dos indivíduos, operadores específicos podem ser desenvolvidos com foco em trabalhar considerando a semântica das soluções. Assim como na maioria dos trabalhos que abordam o uso de semântica em PG, aqui consideramos que a semântica de um indivíduo está associada ao seu fenótipo, ou seja, seu resultado ao ser aplicado em um conjunto de dados do problema que está sendo resolvido. Dessa forma, é possível operar indivíduos considerando sua aptidão e sua semântica.

Como explicado anteriormente, existem na recombinação dois momentos onde a semântica pode ser aplicada: na seleção de indivíduos ou na escolha dos pontos de cruzamento. Cada um desses casos pode ser obtido dependendo de onde a semântica é calculada.

O primeiro caso é uma alteração do processo de seleção dos indivíduos que participam da recombinação, onde geralmente é feito um torneio selecionando dois pais. Seja o indivíduo A, com semântica S_A, o primeiro selecionado para participar da recombinação, o segundo indivíduo B de semântica S_B passa a ser escolhido com base no primeiro. Geralmente, um valor é aplicado baseado na distância semântica $DS(A, B)$, onde pode-se focar em selecionar uma $DS(A, B) < \epsilon$ e obter um par (A, B) que são semanticamente similares, mesmo que não necessariamente sintaticamente similares. Também é possível focar em uma $DS > \epsilon$ e selecionar um par (A, B) semanticamente e sintaticamente diferentes. Há também o caso onde a DS está compreendida num intervalo $\alpha < DS(A, B) < \beta$. Dessa forma, as movimentações no espaço de busca durante a recombinação passam a considerar também a semântica, realizando as movimentações com maior controle, sendo possível até mesmo escolher quando se quer dar passos menores (DS menor) ou passos maiores (DS maior) escolhendo indivíduos mais parecidos ou mais distintos.

O segundo caso é similar ao primeiro, mas mais refinado. Geralmente, ele ocorre após o passo de selecionar os pais A e B. Nesse processo, uma parte do gene, um subprograma no indivíduo A, é selecionado para a recombinação. Em seguida um subprograma em B deve ser selecionado mas, ao invés de selecionar aleatoriamente e sem estratégia um ponto de recombinação, é escolhida uma subárvore cuja semântica atende a um determinado critério. Assim como no primeiro caso, ao se escolher subárvores similares, são dados passos menores no espaço de busca levando a uma busca mais localizada mas que pode ficar contida em uma região. Já passos maiores são dados trocando subárvores dissimilares, o que pode levar a uma busca que abrange uma maior região, mas que pode ter dificuldades em fazer ajustes mais finos. Em geral essa abordagem toma um maior controle sobre a busca baseado na semântica através da recombinação.

Um estudo de efetividade do operador de recombinação mostrou que geralmente 75% das movimentações não afetam de forma expressiva a semântica dos programas

resultantes em relação a seus pais (McPhee et al., 2008). Isso se deve ao fato de, à medida que as gerações avançam a diversidade genética diminui, e cada vez mais os indivíduos se parecem com a melhor solução encontrada até então devido à dispersão de seus genes. Além disso é apontado que quão maior o tamanho da árvore que representa o programa, maior a probabilidade de que uma recombinação não seja efetiva. Esses dados evidenciam a necessidade de desenvolver operações de recombinação que apresentem melhor desempenho dado o custo computacional envolvido na operação.

A literatura apresenta diversos trabalhos propondo operadores de recombinação semânticos para PG. *Semantically Driven Crossover* (SDC) (Beadle and Johnson, 2008) traz uma proposta de recombinação semântica simples e eficiente, onde um indivíduo gerado entra para a população corrente somente se for semanticamente diferente de seus pais, mesmo que tenha uma maior aptidão. A premissa é que uma diversidade genética maior dos programas na população leva a uma melhor exploração do espaço de busca. Os resultados demonstram que a técnica obtém uma melhor desempenho em relação à PG original, e ainda obteve soluções menores, que tendem a ser mais simples e menos complexas.

Em (Nguyen et al., 2009) é apresentado o *Semantic Aware Crossover* (SAC) como uma adaptação do operador de recombinação ao conceito de semântica. É apresentado o conceito de Equivalência Semântica, explicado anteriormente na Equação 4.5. Sua estratégia é baseada em primeiro selecionar as subárvores p na árvore do indivíduo P, e q no indivíduo Q. Então é feita a troca entre elas gerando os filhos P' e Q', e verificado se eles são equivalentes aos pais. Caso não sejam equivalentes, são inseridos na população, caso contrário são descartados. Sua estratégia busca evitar a clonagem, problema que soma nos 75% de operações ineficientes. Vale ressaltar que a distância semântica utilizada por Nguyen et al. (2009) é apenas a diferença absoluta das semânticas, e não sua média.

Em (Uy et al., 2010) é realizada uma adaptação do SAC, criando o *Semantic Sensitive Crossover* (SSC) que simplifica as verificações e aumenta a chance de sucesso em uma recombinação. Seu funcionamento começa selecionando um par de pais P e Q. É definido então um parâmetro Max_Trial que define o número máximo de tentativas de recombinação. Até que seja atingido esse valor, são selecionadas subárvores p e q de P e Q respectivamente. São gerados N pontos aleatoriamente para calcular suas semânticas e então a distância entre elas. Esses pontos são chamados de *fitness cases*. Se as semânticas forem similares, $SS(p, q)$ for verdadeiro, então a recombinação é concluída. Caso contrário outra tentativa é realizada seguindo o mesmo protocolo. Caso não tenha sucesso em nenhuma dessas tentativas, uma recombinação tradicional é realizada. Nesse trabalho a distância semântica utilizada é a definida anteriormente na Equação 4.3.

Em (Forstenlechner, 2019) é apresentado o *Semantic Crossover for Program Synthesis* (SPCS). Sua estratégia é similar ao SSC, baseado em um número máximo de tentativas,

mas caso essa fase de tentativas falhe, ainda há uma alternativa antes de realizar uma recombinação tradicional. O SCPS é aplicado no contexto de PGG para a Síntese de Programas, e é sujeito às restrições de uma recombinação em árvores de derivação geradas através de uma GLC. Primeiro é selecionada uma subárvore p de um pai P. Em seguida são selecionadas até Max_Tries subárvores com possibilidade de troca no outro pai Q, e caso não tenha nenhum, não é realizada a recombinação. A *Diferença Semântica* de cada subárvore para p é calculada e caso hajam subárvores com diferença, o processo segue, caso contrário, é realizada uma recombinação comum. A Diferença Semântica é uma verificação simples, onde há diferença semântica se os valores da semântica de P forem diferentes de Q em pelo menos ponto, e iguais também em pelo menos um ponto. Ou seja, previne uma troca entre os idênticos ou entre os totalmente diferentes. Por se tratar da síntese de programas, é selecionado aleatoriamente um tipo de variável. Isso acontece pois a semântica no SPCS representa um programa, e cada variável do programa possui uma lista de valores representado sua semântica na subárvore. É realizada a recombinação entre p e a subárvore q mais semanticamente similar para o tipo de variável selecionado.

Os operadores presentes na literatura apresentaram bons resultados, no entanto existem características em cada um que podem ser melhor exploradas. O SSC por exemplo, realiza Max_Tries operações, mas pode selecionar o mesmo par de subárvores em cada tentativa. O SCPS considera a semântica de uma variável, porém pode realizar operações em subárvores onde ela não é expressa. Os operadores também se mostram dependentes de parâmetros, podendo torná-los ineficientes ao mudar de problema sem realizar um ajuste desses parâmetros.

5 ESTRATÉGIAS PROPOSTAS

Diversos trabalhos exploram o uso de PG apoiada por semântica. Existem variantes que consideram em qual operador genético ela será aplicada, a forma de calcular, as considerações se algo é similar ou não, o momento da aplicação antes, durante ou depois de realizar a troca de genes na recombinação, entre outras formas de lidar com a semântica. No entanto, somente alguns trabalhos apresentam o uso de GLC em combinação a semântica em regressão simbólica (Forstenlechner, 2019). Este trabalho visa explorar estratégias encontradas na literatura, adaptá-las para o presente caso, e propor um novo operador de recombinação semântico para a PGG. O resultado esperado é encontrar um operador que tenha um melhor desempenho em regressão simbólica. Além disso, uma métrica de distância alternativa baseada na distância de cosseno também é proposta, a fim de verificar se outras interpretações de distância podem afetar de forma positiva a PGG apoiada por semântica.

5.1 Programação Genética Gramatical e Semântica

O uso de semântica em PG foi estudado em diversos trabalhos que mostraram um aumento de desempenho em relação à PG padrão (Nguyen et al., 2009; Uy et al., 2010; Forstenlechner, 2019). Em geral, a semântica de um programa é a representação de seu resultado para cada registro de um conjunto de dados de avaliação. Comumente essa semântica é armazenada em uma lista com cada um desses resultados. Em PG podemos dizer que o fenótipo de um indivíduo é associado a sua semântica. Pensando em uma árvore clássica de PG, a semântica pode ser atribuída à árvore mas também a suas sub-árvores, visto que elas representam subprogramas completos que também podem ser avaliados. Dessa forma, os operadores podem utilizar essa informação semântica para fazer manipulações nas operações genéticas tanto considerando o indivíduo quanto partes do programa.

No caso da PGG não é diferente, é possível atribuir uma semântica para a árvore e suas subárvores. Mas, diferente de árvores da PG tradicional, as árvores da PGG são árvores de derivação, onde seus nós internos são não terminais da gramática, enquanto as folhas são os terminais. De forma análoga à PG, uma sub-árvore da PGG representa um pedaço do modelo completo da árvore, um subprograma, e que é pertencente ainda à linguagem definida pela GLC. Sendo assim, alguns dos operadores de recombinação da literatura podem ser adaptados para a PGG, salvo as restrições impostas às operações da PGG pelas regras da GLC.

5.1.1 Representação

A fim de utilizar a informação semântica de forma mais precisa e computacional-

mente otimizado, evitando recalcular a semântica de um mesmo trecho muitas vezes, é comum que a informação semântica seja inserida na representação. Diferente de uma árvore da PG, que pode ter operações feitas em qualquer nó, a árvore na PGG só aceita operações em nós internos, representando os não terminais da gramática. Adotou-se aqui uma representação onde cada nó interno passa a ter em sua estrutura além de sua regra e seus filhos, sua semântica. A PGG não faz movimentações nos nós folha, logo, eles não são alterados.

A representação da semântica pode variar de acordo com o tipo de problema. Em (Forstenlechner, 2019) sua aplicação na Síntese de Programas traz uma semântica atrelada ao problema, que cria códigos com atribuições de variáveis. Ao mesmo tempo, a síntese é um problema muito generalista, onde ela tem informações de cada variável gerada num subprograma representado por uma subárvore. No caso de uma regressão, a síntese de programas tenta encontrar apenas uma variável. No entanto, para a regressão simbólica de funções de valor real, a semântica pode ser mais simples, geralmente representada por uma lista contendo as respostas da execução do programa ou subprograma num conjunto de dados.

Junto da representação da semântica de um programa vêm sua relação com outros programas de acordo com a semântica. Noções de similaridade e diferença passam a ser pertinentes, pois são artifícios usados nas operações de recombinação. Aqui, serão importantes os conceitos de semântica explorados anteriormente: distância, sensibilidade, equivalência e similaridade.

5.1.2 *Roulette Semantic Crossover*

O uso da semântica na PG geralmente envolve alterações nos operadores genéticos. Esses operadores precisam ser adaptados para realizar movimentações sob as restrições do método sob o qual são aplicados. O objetivo em geral é promover uma maior qualidade das operações realizadas sobre os indivíduos. De uma forma geral, a aplicação de semântica em recombinação significa incluir novas restrições. Essas restrições têm, em geral, dois objetivos: impedir a troca de genes que expressam uma mesma semântica ou que expressam semânticas muito diferentes. Interpretando de outra forma, os objetivos são evitar trocas de subárvores equivalentes ou que representam informações muito distintas. Ao evitar trocas equivalentes, a diversidade dos indivíduos aumenta, o que promove uma maior exploração do espaço de busca. De forma análoga, ao evitar trocas entre componentes muito diferentes, são tomados saltos menores no espaço de busca, levando também a uma exploração mais refinada. No entanto, deve-se encontrar um balanço entre os dois aspectos de exploração, visto que tanto uma busca muito focada numa região do espaço, quanto uma busca muito abrangente, podem atrasar a convergência.

As abordagens de recombinação apoiada por semântica mencionadas anteriormente

apresentam resultados mostrando que seu uso aumenta o desempenho da PG. No entanto, todas estão apoiadas em parâmetros, como sensibilidade, limites superior e inferior, número de tentativas, o que os torna muito dependentes, visto que os valores ideais para esses parâmetros podem variar entre problemas. Este trabalho propõe uma alternativa para o operador de recombinação com semântica, o *Roulette Semantic Crossover* (RSC). O RSC é uma estratégia simples, que privilegia a diversidade semântica e tem um componente aleatório que considera a semântica como critério de escolha entre as opções. Ele pode usar parâmetros como limites superior e inferior de similaridade, mas também pode ser livre e aceitar qualquer subárvore desde que a DS seja maior que zero. O RSC pode tomar tanto passos curtos, o que privilegia a localidade, quanto passos longos, que pode tirar as soluções de mínimos locais.

O algoritmo proposto adota o conceito de roleta para escolha da subárvore q em uma recombinação. Geralmente, durante a recombinação na PG, duas subárvores p e q são simplesmente trocadas aleatoriamente. Já na PGG uma subárvore p é selecionada no primeiro pai, e uma q é escolhida aleatoriamente entre as subárvores de mesmo não terminal que p, salvo restrições como limitação da altura máxima da árvore. O RSC é simples como a recombinação tradicional da PGG entre os indivíduos P e Q. Porém, uma vez escolhida a subárvore p, a probabilidade de escolher cada uma das subárvores candidatas a recombinar em Q não é igual. Uma roleta é construída para a escolha de q, onde a probabilidade de escolher uma subárvore mais similar a p é inversamente proporcional à distância semântica entre p e q

$$p_i = \frac{1}{DS(p, q_i)} \tag{5.1}$$

onde p_i é a probabilidade de escolher a i-ésima subárvore $q_i \in Q$ para a recombinação. Um valor aleatório é sorteado na roleta, onde a probabilidade de escolher uma subárvore mais similar é maior do que uma mais diferente. Vale evidenciar novamente que, a roleta não pode incluir subárvores idênticas ($DS(p, q) = 0$), pois haveria uma indeterminação do tipo divisão por zero. O Algoritmo 2 ilustra o funcionamento do RSC.

Algoritmo 2: Roulette Semantic Crossover.

1 Seleciona subárvore p no pai P
2 Seleciona as subárvores no pai Q com mesmo NT
3 Calcula a distância de cada par $DS(p, q_i)$
4 Cria a roleta com probabilidades $p_i = 1/DS(p, q_i)$
5 Normaliza fazendo $p_i = p_i/sum(DS)$
6 Sorteia um valor $s \in [0, 1]$
7 Seleciona q_s equivalente
8 $swap(p, q_s)$

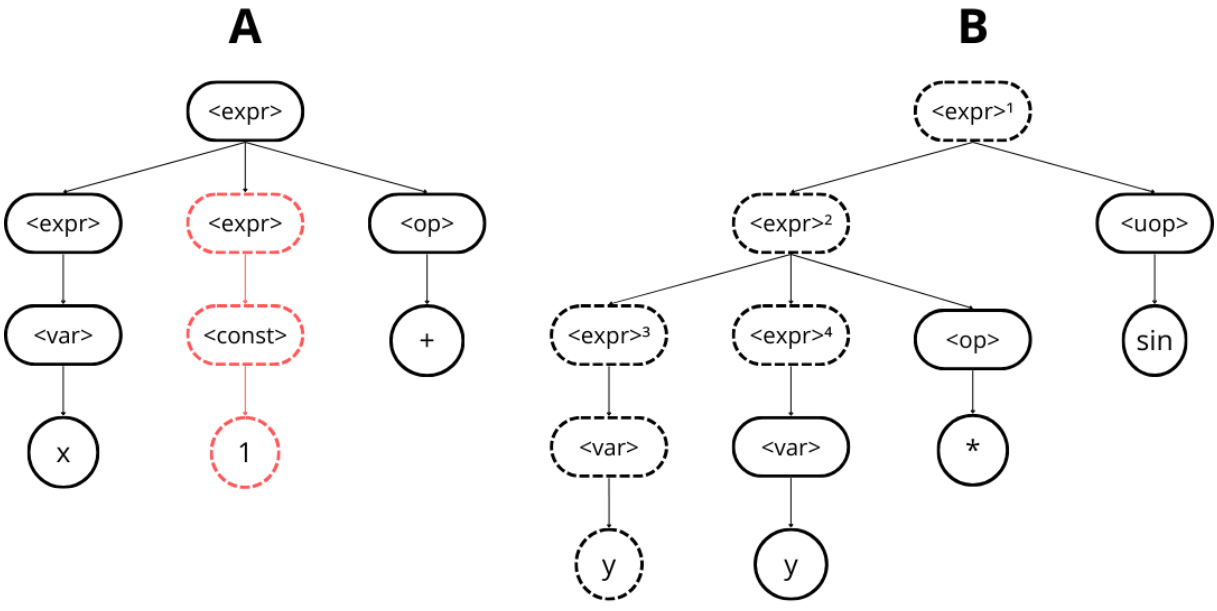
Ao não utilizar parâmetros de limites de similaridade, a roleta funciona atribuindo

valores menores às subárvores menos similares, porém pode ficar muito tendenciosa por soluções muito similares dominarem a roleta. Já ao utilizar os limites, em especial o inferior, é possível evitar que apenas uma solução domine a roleta de forma demasiada, porém o RSC se torna dependente de problemas, o que tende a ser ruim para um algoritmo.

A Figura 13 apresenta um exemplo do RSC sendo aplicado numa recombinação. Seja p o não terminal selecionado no pai A para a recombinação, destacado em vermelho com bordas pontilhadas na imagem. E sejam os candidatos à recombinação os não terminais destacados pelas bordas pontilhadas no pai B, enumerados de 1 a 4. A Figura 14a representa a relação de distância entre os não terminais candidatos, assim como exemplifica a existência de limites inferior e superior de similaridade. No centro está $<expr>$, e ao seu redor vê-se os não terminais candidatos. Cada um com sua distância semântica calculada, em relação ao não terminal que está posicionado no centro. Existem dois raios a partir do centro, um que delimita o limite inferior e outro que limita o limite superior. Eles representam respectivamente α (círculo interior) e β (círculo exterior) apresentados na Equação 4.5. Métodos que utilizam essa estratégia definem uma região delimitada pelo raio equivalente a esses valores, onde as trocas são válidas. Ao utilizar $\alpha = 0$ pode-se permitir que qualquer troca seja realizada, salvo casos como no RSC onde indivíduos com DS nula são eliminados. Ao remover o limite superior é permitido que trocas entre sub-árvores totalmente distintas seja feita.

A Tabela 1 apresenta a semântica de cada uma das subárvores, assim como das variáveis x e y. A partir desses valores, é possível calcular as DS entre cada candidato, que podem ser vistos na Tabela 2.

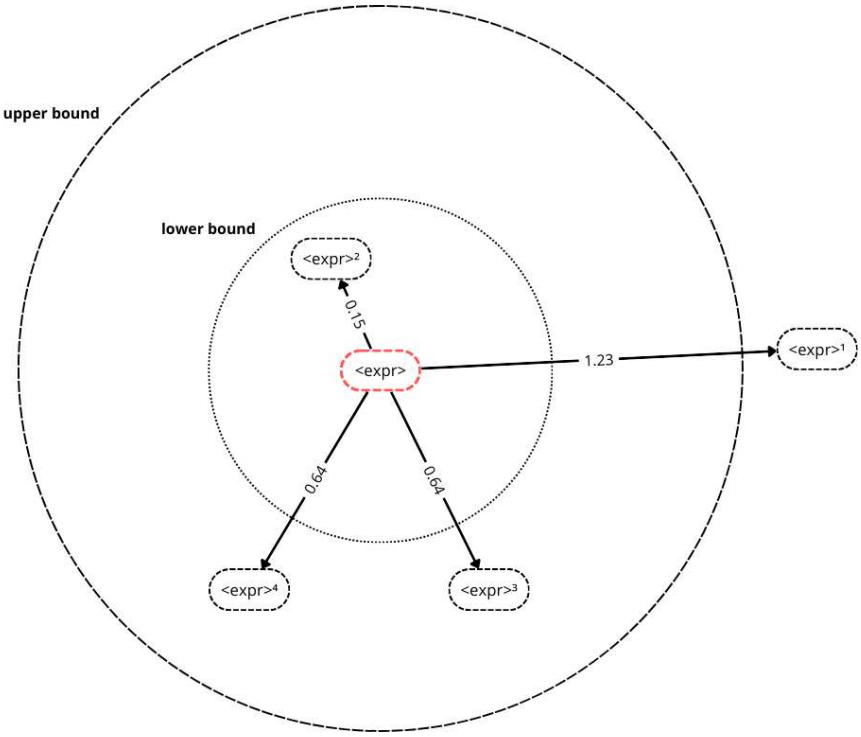
Figura 13 – Exemplo de Recombinação com o algoritmo RSC.



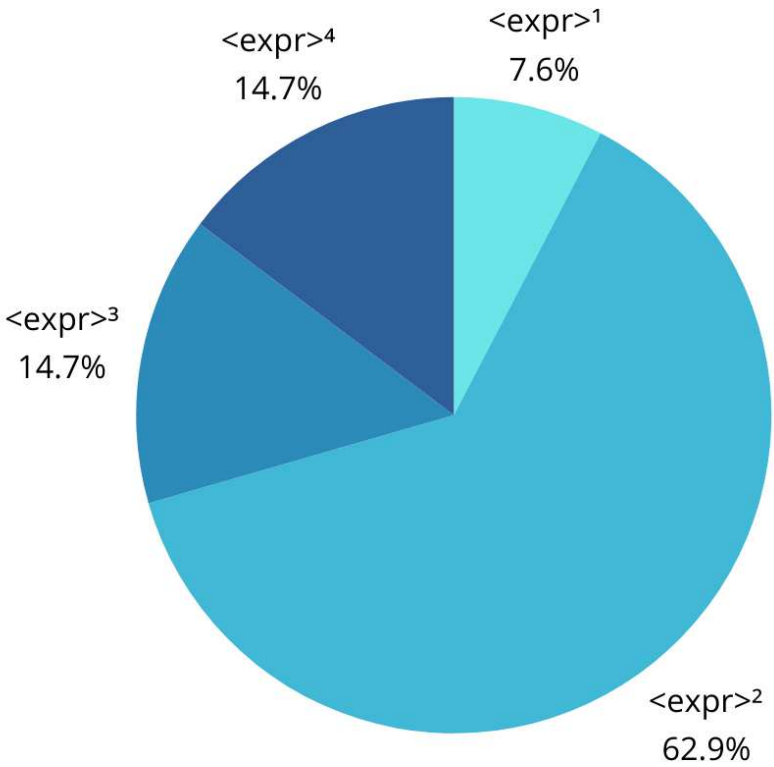
Fonte: Elaborado pelo próprio autor (2024).

Figura 14 – Representação das distâncias semânticas e distribuição de probabilidades na roleta.

(a) Relação entre a distância semântica e limites.



(b) Roleta com as probabilidades de escolha.



Fonte: Elaborado pelo próprio autor (2024).

Tabela 1 – Semântica das subárvores do exemplo da Figura 13

x	= [	0.0,	1.0,	2.0,	3.0,	4.0,	5.0,	6.0,	7.0,	8.0,	9.0]
y	= [	0.0,	0.5,	1.0,	1.5,	2.0,	2.5,	3.0,	3.5,	4.0,	4.5]
$<expr>$	= [	1.0,	1.0,	1.0,	1.0,	1.0,	1.0,	1.0,	1.,0	1.0,	1.0]
$<expr>^1$	= [	0.0,	0.2,	0.8,	0.8,	-0.8,	-0.0,	0.4,	-0.3,	-0.3,	1.0]
$<expr>^2$	= [	0.0,	0.2,	1.0,	2.2,	4.0,	6.2,	9.0,	12.2,	16.0,	20.2]
$<expr>^3$	= [	0.0,	0.5,	1.0,	1.5,	2.0,	2.5,	3.0,	3.5,	4.0,	4.5]
$<expr>^4$	= [	0.0,	0.5,	1.0,	1.5,	2.0,	2.5,	3.0,	3.5,	4.0,	4.5]

Fonte: Elaborado pelo próprio autor (2024).

Tabela 2 – Distância Semântica das subárvores do exemplo da Figura 13

$$DS(<expr>, <expr>^1) = 1.23$$
$$DS(<expr>, <expr>^2) = 0.15$$
$$DS(<expr>, <expr>^3) = 0.64$$
$$DS(<expr>, <expr>^4) = 0.64$$

Fonte: Elaborado pelo próprio autor (2024).

A Figura 14b é a representação da roleta do exemplo, com as probabilidades de seleção de todas as sub-árvores candidatas. Assim, é possível identificar que $<expr>^3$ e $<expr>^4$ possuem mesma sintaxe, logo, têm a mesma semântica. Ainda, podemos ver que $<expr>^1$ é o mais distante, seguido por $<expr>^2$, $<expr>^3$ e $<expr>^4$ são iguais e os mais próximos de $<expr>$. Dessa forma a roleta toma essa configuração, onde todos os candidatos têm chance de serem selecionados, porém o peso para a seleção dos mais próximos é maior do que dos mais distantes.

5.1.3 Adaptações dos algoritmos para PGG

A literatura de PG apresenta alguns algoritmos de recombinação que utilizam semântica para guiar seus movimentos. No entanto, soluções aplicadas à PGG ainda são escassas. Apesar da aplicação direta desses algoritmos ser possível, pode ser vantajoso explorar características da PGG. No geral, a recombinação troca genes aleatoriamente selecionados. Já ao se aplicar a semântica na recombinação, passa a ser avaliado o resultado da operação antes de adicionar os novos indivíduos à população. A escolha aleatória apresenta uma desvantagem, pois pode realizar trocas entre árvores similares ou até até mesmo idênticas, o que tende a contribuir menos para uma boa localidade e diversidade. Já a aplicação clássica de semântica em alguns operadores somente irá evitar inserir os novos filhos semelhantes aos pais na população, e o custo computacional da recombinação deve ser repetido para gerar novos indivíduos. No entanto, ainda assim, a mesma escolha pode ser feita, levando a ciclos e à necessidade de implementar dispositivos de segurança, como marcar trocas já realizadas.

A fim de evitar esses problemas, e aproveitando as capacidades da PGG, os

operadores SAC e SSC são adaptados, tendo a avaliação semântica sendo feita sob subárvores dos pais e não sob os filhos gerados através de uma recombinação aleatória. Devido à representação do indivíduo na PGG, uma árvore de derivação, a recombinação é feita através da troca de subárvores, que representam regras da gramática. Para a troca dessas subárvores ser possível, ambas devem pertencer ao mesmo não terminal, além de atender a outras restrições como limite de altura máxima da árvore. No caso do SAC e SSC, ao invés de realizar a avaliação dos indivíduos resultantes da recombinação, é feita a avaliação nas subárvores que serão trocadas.

A adaptação desses operadores para o contexto da PGG se inicia muito similar entre si, e são próximos da recombinação padrão. Inicialmente no primeiro pai é selecionada uma subárvore p aleatória. Em seguida, é identificado o não terminal da raiz dessa subárvore e listadas as opções de subárvores do outro pai cuja raiz pertence ao mesmo não terminal. A partir de agora, o processo incorpora o conceito de semântica. Uma vez identificadas as subárvores candidatas, o processo do SAC e do SSC passam a se diferenciar.

No caso do SAC, é verificada a Equivalência Semântica (ϵ). Um par é selecionado aleatoriamente e então sua distância semântica é calculada e verificada a equivalência ($DS(p, q) < \epsilon$). Caso passe no teste, a recombinação é realizada gerando os novos filhos com material genético dos pais e que não são semanticamente equivalentes, num processo que garante a escolha válida dadas restrições da GLC e da semântica.

Já para o SSC, é verificada a Similaridade Semântica (SS), identificando se a distância de um par está dentro de um intervalo ($\alpha < DS(p, q) < \beta$) e caso não esteja no intervalo de similaridade, ele é desconsiderado. São realizadas Max_Tries tentativas de realizar as trocas, aumentando as chances de sucesso e caso a SS seja verificada, a recombinação é completada gerando novos indivíduos semanticamente e sintaticamente válidos.

5.1.4 Distância de Cosseno

A literatura explora de forma exaustiva a distância absoluta (ABS), definida na Equação 5.2, como medida de distância entre a semântica de um par de subprogramas p e q. No entanto, existem outras formas de cálculo de distância que podem representar a similaridade de uma forma diferente.

$$ABS(p, q) = |p - q| \tag{5.2}$$

A distância de cosseno (COS) é uma métrica amplamente utilizada para medir a similaridade entre dois vetores em um espaço vetorial multidimensional. Ela é particularmente útil em contextos onde a orientação dos vetores é mais importante do que suas magnitudes, como na análise de texto e na recuperação de informações. Um estudo recente de Juvekar and Purwar (2024) explora como COS pode ser usada para melhorar a recupe-

ração de informações em grandes bases de dados, demonstrando que essa abordagem pode ser particularmente eficaz na recuperação de informações esparsas. Outro trabalho aborda a reconciliação entre a distância Euclidiana e a similaridade de cosseno em problemas de tomada de decisão individual (Mukherjee and Sonal, 2023), destacando a importância de escolher a medida correta de similaridade dependendo do contexto do problema. A COS pode ser definida como

$$DS(p,q) = COS(p,q) = 1 - cos(\theta) = 1 - \frac{p \ . \ q}{||p||\ ||q||} \tag{5.3}$$

sendo $p \ . \ q$ o produto interno entre os vetores e $||p||\ ||q||$ o produto das normas. A COS varia entre $[0, 2]$, onde 0 indica que os vetores são idênticos em termos de direção (ou altamente similares), enquanto 2 indica que os vetores são diametralmente opostos. É usual que a distância seja normalizada para variar em $[0, 1]$, onde 0 indica que não há diferença, e 1 indica a maior dissimilaridade possível.

COS mede o ângulo entre vetores, fornecendo uma medida de quão próximos estão em termos de direção. Dois vetores podem ter uma pequena distância de cosseno mesmo se estiverem longe um do outro em termos absolutos, desde que apontem na mesma direção. Apesar de que, em contextos numéricos, ABS é mais utilizada por sua fácil interpretação geométrica, aqui propomos também o uso de COS. A interpretação geométrica para o uso de COS é inspirada recuperação de informações e mineração de textos. A a semântica de um texto, neste caso um modelo, representa a informação contida nele, e geometricamente é sua direção no espaço semântico.

6 EXPERIMENTOS COMPUTACIONAIS

Os experimentos foram projetados para avaliar a eficiência de operadores de recombinação apoiados por semântica: o algoritmo proposto RSC, o operador de recombinação padrão da PGG (ilustrado na Figura 7), e outros métodos semânticos da literatura, sendo eles o SAC (Nguyen et al., 2009), o SSC (Uy et al., 2010) e o SCPS (Forstenlechner, 2019). O objetivo é determinar o impacto da incorporação de informações semânticas na busca e mensurar a melhora do desempenho da PGG em problemas de regressão simbólica. Ainda, os experimentos envolvem a comparação do uso da distância absoluta ABS e a distância de cosseno COS, como medidas de distância semântica, com o objetivo de avaliar qual o desempenho de cada uma.

Para avaliar o desempenho dos diferentes métodos de recombinação, foi utilizado um conjunto de 10 funções de benchmark extraídas de (Uy et al., 2010). Essas funções variam em complexidade e número de dimensões, proporcionando uma ampla gama de desafios para os algoritmos testados. As funções incluem polinômios e funções com operadores trigonométricos e logaritmo. A Tabela 3 lista as funções, assim como o número de variáveis, os operadores presentes nelas, a quantidade de pontos e o domínio das variáveis.

Tabela 3 – Funções do benchmark utilizado nos experimentos.

Função	# Variáveis	Operadores	# Pontos	Domínio
$F1 = x + x^2 + x^3$	1	+, pow	20	[-1, 1]
$F2 = x + x^2 + x^3 + x^4$	1	+, pow	20	[-1, 1]
$F3 = x + x^2 + x^3 + x^4 + x^5$	1	+, pow	20	[-1, 1]
$F4 = x + x^2 + x^3 + x^4 + x^5 + x^6$	1	+, pow	20	[-1, 1]
$F5 = sin(x^2)cos(x) - 1$	1	pow, sin, cos	20	[-1, 1]
$F6 = sin(x) + sin(x + x^2)$	1	+, pow, sin	20	[-1, 1]
$F7 = log(x + 1) + log(x^2 + 1)$	1	+, pow, log	20	[0, 2]
$F8 = \sqrt{x}$	1	$\sqrt{}$	20	[0, 4]
$F9 = sin(x_1) + sin(x_2^2)$	2	+, pow, sin	100	[-1, 1]
$F10 = 2sin(x_1)cos(x_2)$	2	*, sin, cos	100	[-1, 1]

Fonte: Uy et al. (2010)

6.1 Configuração dos Experimentos

Esta seção detalha o planejamento e execução dos experimentos, descrevendo de forma detalhada os parâmetros utilizados, a seleção e preparação dos conjuntos de dados, a definição da gramática empregada nos algoritmos e as abordagens adotadas para comparação de desempenho entre as diferentes técnicas testadas.

6.1.1 Seleção de Parâmetros

A Tabela 4 apresenta os parâmetros utilizados nos experimentos. O cálculo da semântica tem um custo computacional associado, que deve ser considerado ao comparar métodos que aferem a semântica de diferentes formas. Um exemplo é o SAC, que realiza um cálculo de semântica para cada filho ao realizar uma recombinação, já o SSC e SCPS realizam até Max_Tries cálculos por recombinação. O RSC faz um cálculo para cada subárvore candidata a recombinar, que pode variar com o tamanho das árvores e tipo de não terminal selecionado. O cálculo de semântica é o conceito de $fitness\ cases$, onde cada cálculo de $DS(p, q)$ é um $fitness\ case$, e sua contagem é utilizada para equalizar o custo computacional entre os operadores de recombinação. Os valores de *lower_bounds* e *upper_bounds* são aplicados em cada um dos algoritmos onde podem ser aplicados, fazendo todas as permutações possíveis. O termo *distance* se refere a qual medida de DS foi utilizada, a distância absoluta entre p e q ou a distância de cosseno. Os parâmetros *crossover_rate* e *mutation_rate* representam a chance de executar uma recombinação e mutação respectivamente a cada chamada dos operadores, sendo possível que a recombinação gere réplicas dos pais e a mutação não gere modificações nos filhos. O parâmetro *elite* representa o número dos melhores indivíduos da população atual que serão mantidos para a próxima geração. O parâmetro *max tree depth* limita a altura máxima da árvore de derivação a fim de evitar derivações degeneradas e problemas como recursos computacionais limitados.

Tabela 4 – Parâmetros Utilizados nos Experimentos. Os valores de *lower_bounds* e *upper_bounds* foram definidos neste trabalho, *crossover_rate* e *mutation_rate* definidos em de Freitas et al. (2015), os demais são definidos em Uy et al. (2010).

Parâmetro	Valor
lower_bounds	$[0, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1]$
upper_bounds	$[0, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1]$
distance methods	$[ABS, COS]$
population size	500
evaluations	15×10^6 $fitness\ cases$
crossover_rate	90%
mutation_rate	10%
elite	5
max tree depth	10

Fonte: Elaborado pelo próprio autor (2024).

6.1.1.1 Conjuntos de Dados

O benchmark apresentado em (Uy et al., 2010) utiliza apenas um conjunto de treinamento de tamanho restrito, variando entre 20 e 100 registros como apresentado na Tabela 3. No entanto, a fim de aferir a aptidão das soluções encontradas no processo de

treinamento, é comum utilizar um conjunto de dados de teste. Este conjunto serve para medir o desempenho do modelo em dados que não são apresentados durante o treinamento.

Além disso, para se avaliar a robustez de métodos estocásticos, geralmente são realizados diversas execuções independentes, onde pode-se aplicar ferramentas mais apropriadas para medir o desempenho do algoritmo sobre todos os seus resultados, como a ferramenta *k-Fold Cross Validation* (Langford, 2005), ou Validação Cruzada. O *k-fold cross-validation*, ou validação cruzada, é uma técnica amplamente utilizada para avaliar o desempenho de modelos preditivos, com o objetivo de medir sua capacidade de generalização. Ao invés de apenas treinar e testar o modelo em um único conjunto de dados, o conjunto é dividido em k partes, chamadas de *folds*. O modelo é treinado em *k-1* dessas partes e testado na parte restante, o processo é repetido k vezes, de forma que cada parte seja usada uma vez para teste. Essa abordagem permite uma avaliação mais robusta do desempenho do modelo em dados não apresentados durante o treinamento, minimizando problemas como o *overfitting* (ajuste excessivo aos dados de treinamento) e o *underfitting* (quando o modelo não captura adequadamente os padrões dos dados).

O processo de *k-fold* é baseado em dividir o conjunto de dados em k partes, onde uma é utilizada para testar a solução final encontrada através do treinamento nas outras $k - 1$ partes. Em cada execução o *fold* de teste é alterado, variando de forma que todos os *folds* sejam utilizados como teste ao menos uma vez. Os resultados das iterações podem ser agregados para fornecer uma estimativa média da desempenho do modelo e sua variabilidade. Um outro conjunto de dados pode ser aplicado, para validar todos os resultados e compará-los sobre um mesmo ambiente. Neste trabalho utiliza-se o dobro dos pontos apresentados na Tabela 3, sendo 40 e 200 pontos gerados para as funções de uma e duas variáveis. Através da estratégia definida em Uy et al. (2010), os pontos são gerados aleatoriamente dentro de um domínio fechado. Utiliza-se a validação cruzada com *5-folds*, mantendo os conjuntos de treinamento e teste com tamanho 20. Ao final todos os melhores resultados são avaliados num conjunto de 1000 pontos para comparação entre os algoritmos.

6.1.2 Gramática

A GLC utilizada nos experimentos é simples e direta. Ela apresenta uma formação posfixada, onde pode criar derivações para operadores com um ou dois parâmetros, além dos terminais que variam de acordo com o problema.

$$
\begin{aligned}
<expr> &::= <expr><expr><op> \mid <expr><uop> \mid <t> \\
<op> &::= + \mid - \mid * \mid / \\
<uop> &::= sin \mid cos \mid exp \mid log \\
<t>^1 &::= x_1 \mid 1 \\
<t>^2 &::= x_1 \mid x_2
\end{aligned}
$$

Apesar de existir um conjunto de operadores definidos para cada uma das funções na Tabela 3, os experimentos realizados utilizaram a mesma gramática, diferindo somente no não terminal $<t>$, onde $<t>^1$ define a regra do não terminal t para os problemas de uma variável e $<t>^2$ para os problemas com duas variáveis. Vale ainda ressaltar que todas as funções apresentadas na Tabela 3 estão contidas na linguagem definida pela GLC.

6.1.3 Comparação de Resultados

Uma ferramenta para comparar diferentes algoritmos em um mesmo conjunto de problemas são os Perfis de Desempenho (PP) (Dolan and Moré, 2002). Os PPs avaliam o desempenho relativo de vários algoritmos em um conjunto de problemas. Dado um conjunto S de algoritmos s_i para $i \in \{1, \ldots, n_s\}$ e um conjunto P de problemas p_j para $j \in \{1, \ldots, n_p\}$, $t_{p,s}$ representa a medida de desempenho do algoritmo s no problema p. A razão de desempenho $r_{p,s} = \frac{t_{p,s}}{\min\{t_{p,s}:s\in S\}}$ indica o desempenho relativo do método s. A probabilidade de que a razão de desempenho $r_{p,s}$ para o método s esteja dentro de um fator $\tau > 0$ do melhor valor observado em S é dada por $\rho_s(\tau) = \frac{1}{n_p}|\{p \in P : r_{p,s} \leq \tau\}|$, onde $\rho_s(\tau)$ define as curvas de PP. De acordo com os PPs (Barbosa et al., 2010), é possível identificar: (i) a abordagem que obtém os melhores resultados para a maioria dos problemas (maior $\rho(1)$), (ii) a abordagem mais confiável (menor τ tal que $\rho(\tau) = 1$), e (iii) a abordagem com o melhor desempenho geral, maior área sob as curvas de PP (AUC). Os PPs serão utilizados para mensurar o desempenho dos melhores parâmetros para cada algoritmo, assim como o desempenho entre os melhores algoritmos de cada tipo.

Outra forma de comparar resultados é utilizando testes estatísticos. O teste de Kruskal-Wallis é um método estatístico não paramétrico utilizado para determinar se existem diferenças significativas entre grupos independentes. Este teste é uma extensão do teste de Mann-Whitney, que compara apenas dois grupos. O teste de Kruskal-Wallis é particularmente útil quando os pressupostos de normalidade e homogeneidade de variâncias não são atendidos, tornando-o uma alternativa robusta ao ANOVA (Análise de Variância) tradicional. O teste se baseia na classificação dos dados em vez de seus valores absolutos. Os dados de todos os grupos são combinados e classificados em ordem crescente. Em seguida, as classificações são somadas para cada grupo, e o teste avalia se as somas das

classificações diferem significativamente entre os grupos. Se a estatística for grande o suficiente, rejeitamos a hipótese nula de que todas as populações têm a mesma distribuição. No contexto da comparação dos algoritmos, o teste de Kruskal-Wallis pode ser utilizado para avaliar se diferentes algoritmos apresentam desempenhos significativamente distintos em um conjunto de problemas. Este método é particularmente relevante quando os dados de desempenho dos algoritmos não seguem uma distribuição normal ou quando há variabilidade heterogênea entre os grupos. O teste foi realizado utilizando a biblioteca *scipy* em ambiente Python.

O ambiente de experimentação é Linux na distribuição Ubuntu 22.04.4 LTS, em um computador com processador Intel i7-10700, com 16Gb de memória RAM. O código da PGG utilizado de Freitas et al. (2022) é implementado em C++11, onde foi adaptado para integrar a semântica e implementar o RSC e os outros algoritmos SAC, SSC e SCPS. Para as análises foram utilizados notebooks Jupyter do Python, utilizando bibliotecas como pandas e numpy para manipulação de dados, scypy, sklearn e random para análise dos resultados e matplotlib para visualização.

Os dados experimentais, assim como o código fonte está disponibilizado no repositório *open source* semantic_ggp[1].

6.2 Resultados

Esta seção apresenta os resultados obtidos nos experimentos. Inicialmente é analisado o desempenho individual de cada algoritmo, comparando sua variação paramétrica a fim de encontrar a melhor combinação para cada algoritmo. Essa análise é feita através dos PPs, analisando a área e crescimento das curvas. Em seguida a mesma análise é conduzida entre os melhores resultados de cada algoritmo. Depois são avaliados os sucessos, ou seja, o número de vezes em que o algoritmo atingiu um resultado considerado sucesso, definido na Tabela 4. Por último é realizada uma análise estatística.

6.2.1 Desempenho dos Algoritmos

Para cada algoritmo proposto foi feita uma análise para entender o comportamento de seus parâmetros, assim como identificar a melhor configuração para resolver as funções descritas anteriormente. Os resultados de cada algoritmo são comparados entre si, elegendo a melhor configuração de cada um. Ao final, esses são comparados entre si para identificar quais os melhores algoritmos.

[1] https://github.com/joao-ufjf/semantic_ggp.git

6.2.1.1 Roullete Semantic Crossover (RSC)

O RSC, algoritmo de recombinação proposto neste trabalho, foi avaliado em diferentes configurações, onde cada uma foi executada 100 vezes, assim como todos os outros algoritmos. Seus parâmetros são a medida de distância, *lower_bound* e *upper_bound*. Ambas medidas de distância foram testadas, ABS e COS, e o par de limites (*lower_bound*, *upper_bound*) também foi testado em todas as permutações possíveis. Os PP são exibidos na Figura 15a, onde os valores de τ estão em escala logarítmica para facilitar a leitura.

Avaliando as AUC do PP, é possível identificar que a maioria das curvas atinge um valor de $\rho(\tau)$ próximo de 1 com $\tau < 10^1$, exceto por uma curva, indicando que quase todos os problemas são resolvidos dentro dessa razão de desempenho. As curvas não obedecem um mesmo padrão, o que indica que diferentes configurações dos parâmetros têm diferentes eficiências. Não há uma linha que se destaque claramente sobre todas as outras em todo o intervalo de τ, mas a maior AUC é a do RSC com distância ABS, limite inferior 10^{-5} e superior 10^{-2}, que aqui será nomeado (RSC, ABS, 10^{-5}, 10^{-2}). Já a variante que obtém sucesso na resolução dos problemas mais vezes, ou seja, a que obtém $\rho(\tau) = 1$ com o menor valor de τ, é (RSC, ABS, 10^{-4}, 10^{-2}).

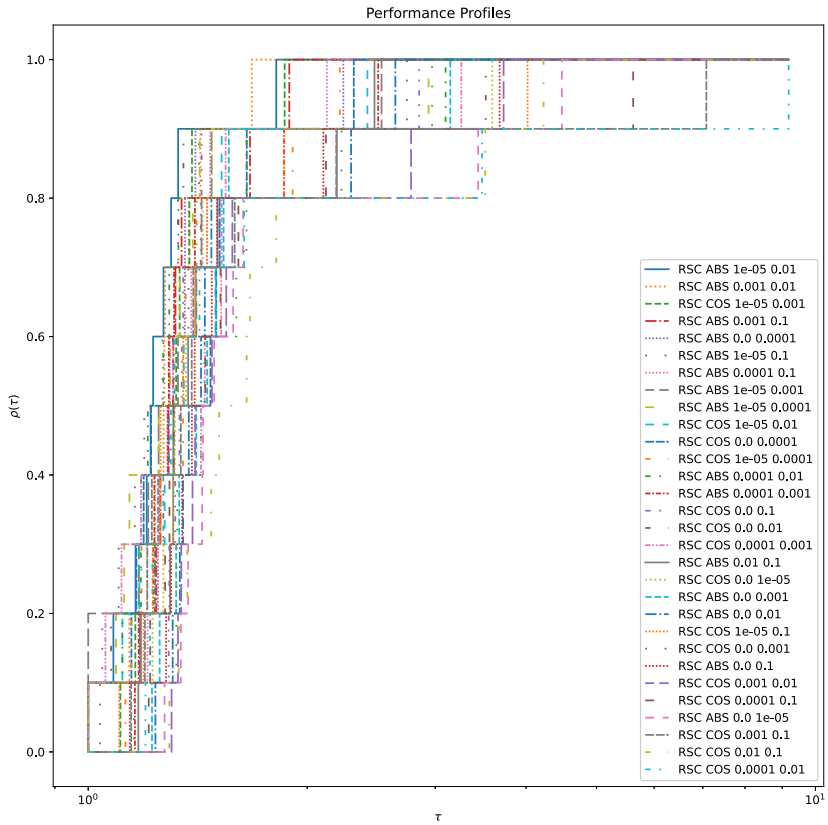
A Figura 15b apresenta as áreas em relação às configurações do algoritmo na forma de um mapa de calor. Apesar de ABS estar presente na curva de maior área, as configurações com ABS e COS estão bem distribuídas, mostrando que ambos os métodos de distância podem ser eficazes, mas com um pequeno favorecimento para a ABS. É possível evidenciar também que valores menores de *lower_bound* tendem a gerar melhores resultados, assim como valores de *upper_bound*, o que indica que o RSC sem limite inferior tende a não ter os melhores resultados, assim como um com limite superior muito alto também não. Por outro ponto de vista, a roleta tende a ser dominada por subárvores similares quando o limite inferior é inexistente.

6.2.1.2 Semantic Crossover for Program Sinthesys (SCPS)

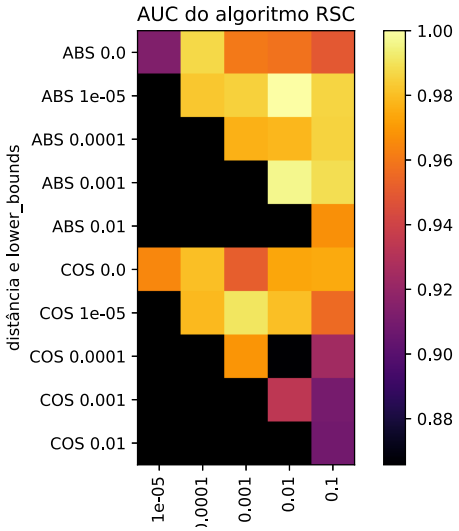
O algoritmo SCPS tem somente duas configurações, a com distância ABS e a com distância COS. As Figuras 16 apresentam os resultados dos perfis de desempenho e áreas sob as curvas dos perfis respectivamente. O SCPS possui duas etapas para selecionar uma subárvore para a recombinação: na primeira etapa ele identifica se existem subárvores com diferença semântica, e caso existam troca pela mais similar; na segunda etapa, caso a primeira falhe, ele seleciona aquela que tem pelo menos uma semelhança e uma diferença. O método de distância impacta a primeira fase, no passo de escolher a subárvore mais semelhante dentre as que tem diferença. Os resultados mostram que o método de distância ABS é superior ao COS.

Figura 15 – Resultados dos Perfis de Desempenho para o algoritmo RSC.

(a) Perfis de Desempenho para o algoritmo RSC. Cada curva é uma variação paramétrica do RSC. Curvas com maior área e $\rho(\tau) = 1$ para menores valores de τ apresentam melhores desempenhos.



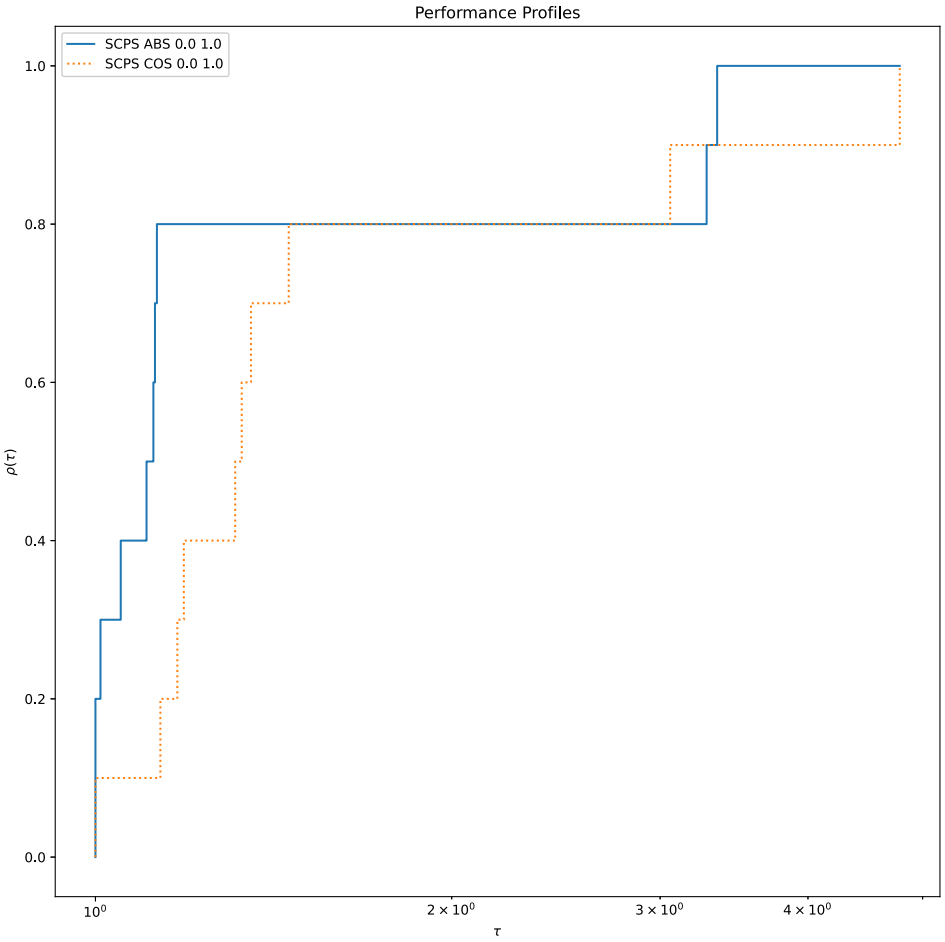
(b) Área sob as curvas dos Perfis de Desempenho do algoritmo RSC. Cada quadrado representa a área do PP de uma combinação de medida de distância, limite superior e limite inferior. Valores menores apresentam cores escuras e representam menores áreas e pior desempenho. Já as maiores valores cores claras, maiores áreas e melhor desempenho.



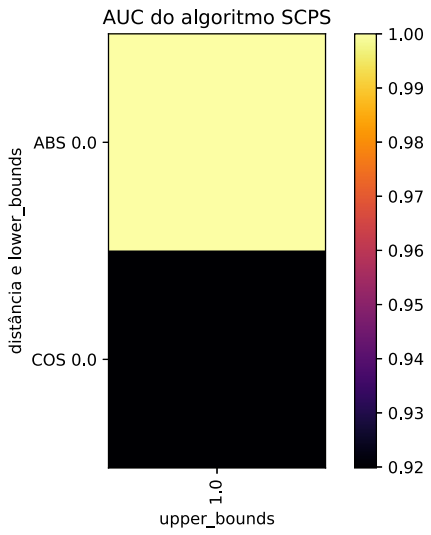
Fonte: Elaborado pelo próprio autor (2024).

Figura 16 – Resultados dos Perfis de Desempenho para o algoritmo SCPS.

(a) Perfis de Desempenho para o algoritmo SCPS.



(b) Área sob as curvas do algoritmo SCPS.



Fonte: Elaborado pelo próprio autor (2024).

6.2.1.3 Semantic-sensitive Crossover (SSC)

O algoritmo SSC, assim como o RSC, possui limites inferior e superior. Ele aceita a troca de subárvores desde que a candidata esteja dentro de uma certa região de similaridade. Pelo perfil de desempenho na Figura 17 é possível notar o domínio da configuração (SSC, COS, 10^{-5}, 10^{-1}), e depois pela (SSC, COS, 10^{-4}, 10^{-1}). Na Figura 17b é possível notar que os melhores resultados claramente estão associados a valores de limite superior. Também pode-se interpretar que os melhores resultados estão agrupados ao redor do limite inferior 10^{-5} e superior 10^{-1}, com uma vantagem para as configurações associadas à distância COS.

6.2.1.4 Semantic Aware Crossover (SAC)

O SAC possui apenas um limite, considerado aqui como o limite superior. Na Figura 18a pode-se ver o domínio evidente da configuração (SAC, COS, 0, 10^{-1}). Já na Figura 18b pode-se acompanhar como valores maiores de limite superior produzem melhores resultados para ambas as métricas de distância. Em geral, a distância COS produziu melhores resultados, enquanto ABS teve apenas uma configuração com resultados expressivos.

6.2.2 Desempenho Geral

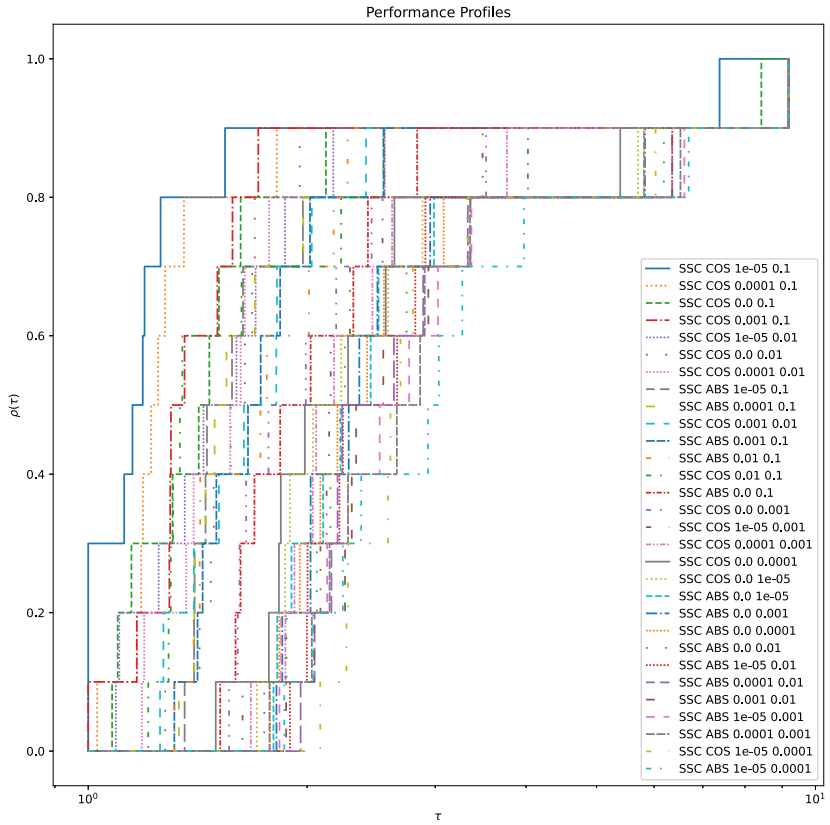
A Figura 19 apresenta o PP calculado entre as melhores variações paramétricas de cada um dos algoritmos apresentados. O algoritmo padrão da PGG é nomeado aqui como DEF. Os resultados dos experimentos mostraram que o RSC apresenta um desempenho robusto e eficiente, superando os métodos de recombinação padrão e outros métodos semânticos em termos de consistência e eficiência geral. Em seguida o DEF, algoritmo de recombinação padrão da PGG, é o segundo em eficácia, seguido respectivamente pelo SCPS, SSC e SAC. A distância ABS está presente no melhor resultado com o RSC e SCPS, enquanto SSC e SAC tiveram melhores resultados com o COS.

6.2.3 Comparação entre Algoritmos

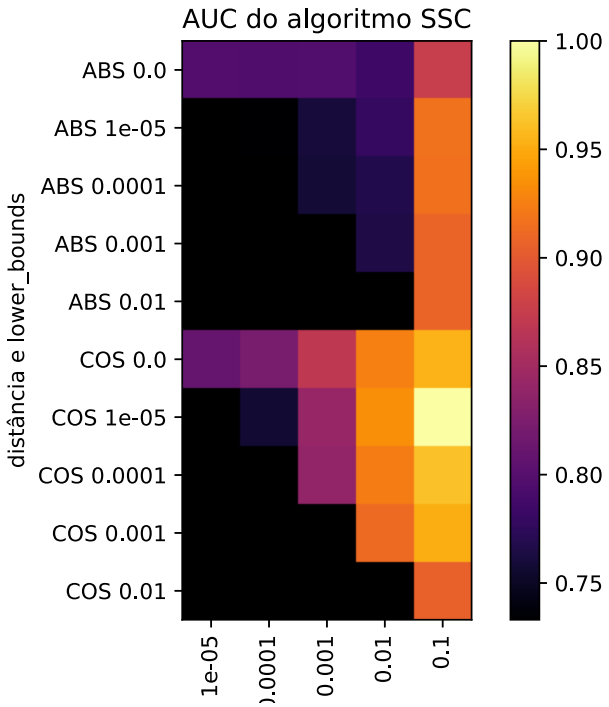
A comparação entre os diferentes algoritmos foi realizada utilizando novamente os Perfis de Desempenho, que avaliam o desempenho relativa dos algoritmos em um conjunto de problemas. Nesta forma o PP conta com o algoritmo com a maior área para cada uma das recombinações testadas. Através das curvas do PP é possível notar que o RSC destacou-se com uma maior área e por obter $\rho(\tau)$ para o menor valor de τ, o que representa seu bom desempenho. O crescimento da curva de um algoritmo denota sua robustez. Ao aproximar de $\rho(\tau) = 1$ primeiro, um algoritmo pode ser descrito como mais robusto que outro. Embora em valores baixos de τ o RSC tenha um crescimento abaixo de outros

Figura 17 – Resultados dos Perfis de Desempenho para o algoritmo SSC.

(a) Perfis de Desempenho para o algoritmo SSC.



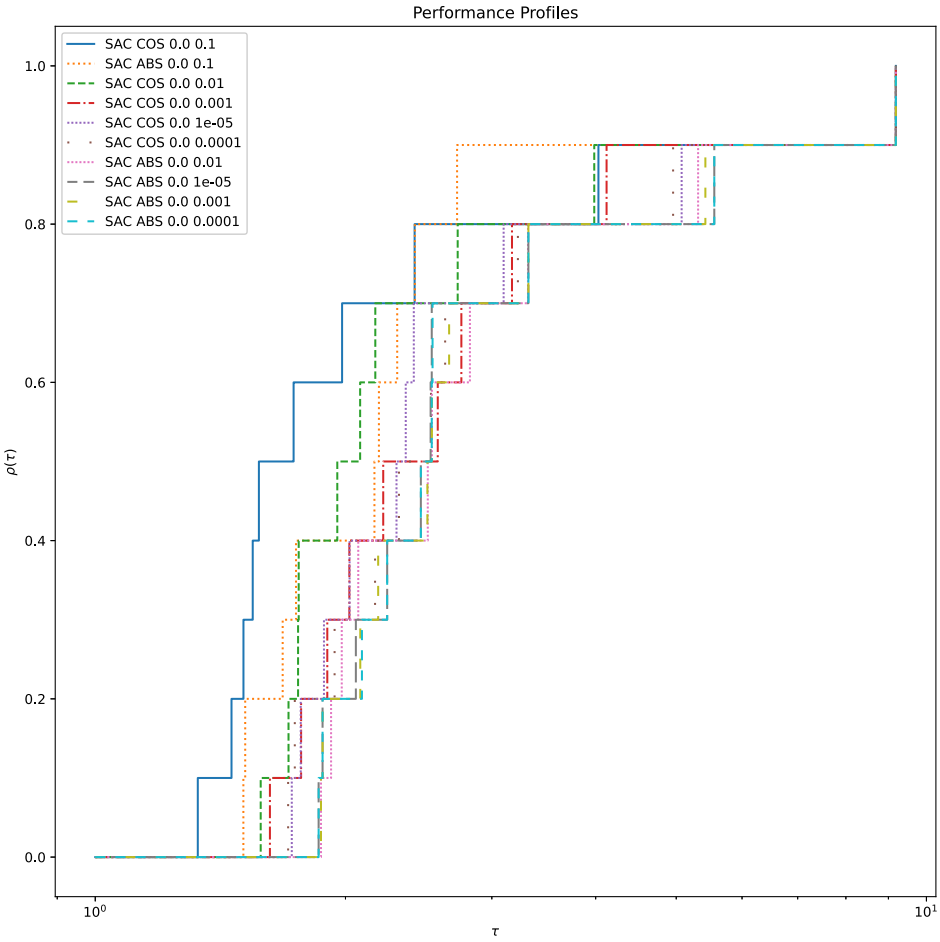
(b) Área sob as curvas do algoritmo SSC.



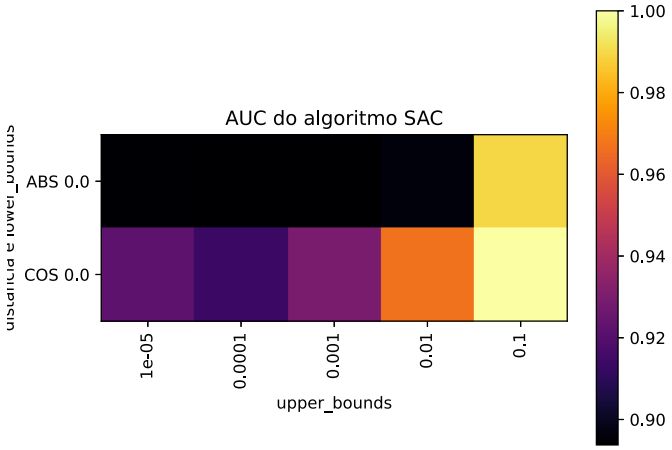
Fonte: Elaborado pelo próprio autor (2024).

Figura 18 – Resultados dos Perfis de Desempenho para o algoritmo SAC.

(a) Perfis de Desempenho para o algoritmo SAC.



(b) Área sob as curvas do algoritmo SAC.



Fonte: Elaborado pelo próprio autor (2024).

métodos como SCPS, SSC e até mesmo DEF, em torno de $\tau \sim 1.3$ ele passa a se destacar, se mostrando um algoritmo robusto e também acumulando uma maior área.

O DEF mostrou um aumento gradual superior ao RSC no início até em torno de $\tau \sim 1.2$, mas em seguida é superado pelo RSC. Assim, DEF é o segundo algoritmo a alcançar $\rho(\tau) = 1$. O desempenho do algoritmo padrão da PGG neste caso pode parecer contraintuitivo, uma vez que os experimentos realizados na literatura mostram uma vantagem do SAC e SSC especificamente nesse conjunto de funções. No entanto, o experimento realizado por Uy et al. (2010) utiliza apenas pequenos conjuntos de treinamento, o que pode não ser suficiente para avaliar a robustez dos algoritmos e até mesmo compará-los de forma satisfatória. Na realidade, é esperado que o DEF tenha uma boa robustez, e seja um método estável, mas aqui vê-se que, para essas funções, usando os parâmetros definidos nos experimentos, ele supera todos os outros métodos exceto a proposta.

O algoritmo SCPS tem a terceira maior área, onde em valores menores de τ tem a maior ascensão, o que garante um acúmulo de área superior aos outros. No entanto, a partir de um certo ponto, é passado pelo RSC, SSC e DEF. Ainda assim, ele foi capaz de atingir $\rho(\tau) = 1$ antes de SSC e SAC, superando-os também em área.

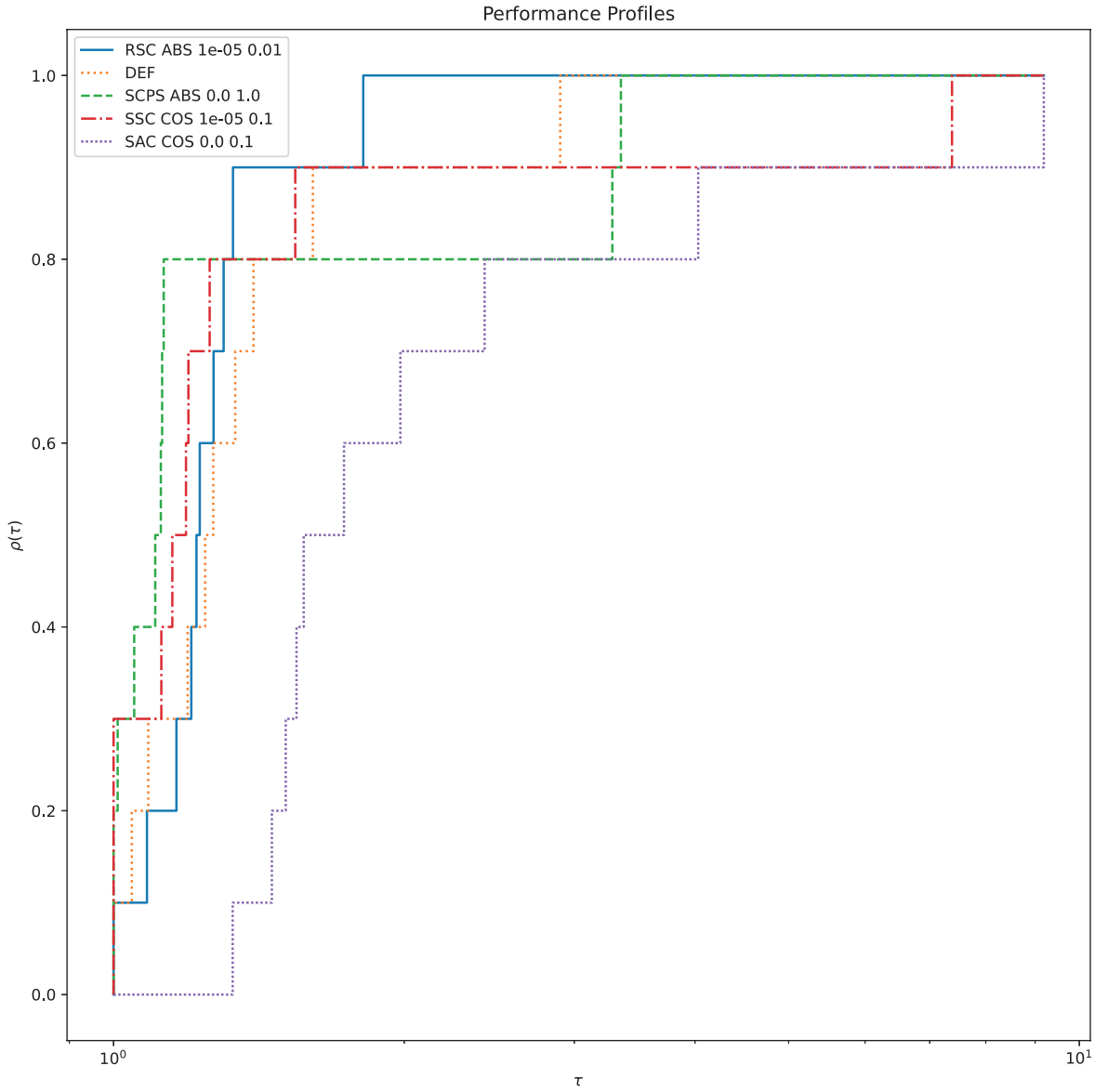
A quarta maior área pertence ao algoritmo SSC. Os algoritmos com maior área até então seguem a ordem RSC, DEF, SCPS e SSC, e formam um conjunto que tem comportamento similar no começo do PP, caminhando razoavelmente juntos até em torno de $\tau = 1.5$, onde começam a divergir. A menor área pertence ao SAC, o único algoritmo que não acompanha os outros no início do gráfico, ficando então defasado desde o início. Além disso, o algoritmo fica abaixo de todas as outras curvas, mostrando que todos os métodos o superam, inclusive a PGG padrão.

6.2.4 Avaliação de Sucessos

Uma forma de avaliar o desempenho nos problemas propostos é comparar os resultados obtidos com a solução ótima, ou contabilizar a quantidade de vezes que ele obtém sucesso. A medida de sucesso pode ser erro zero ou erro abaixo de um valor de ϵ estipulado. Neste trabalho, utiliza-se como medida de sucesso $\epsilon \leq 10^{-2}$ (Uy et al., 2010). A Tabela 5 exibe a contagem da quantidade de vezes em que cada algoritmo alcança um sucesso em cada problema.

O total de sucessos foi de 578 execuções entre as 74000, totalizando em torno de $0,7\%$. Primeiro, avaliamos a quantidade de sucessos por problema, tentando identificar alguma característica dos problemas que podem favorecer as chances de obter sucesso ou não. Claramente, os problemas F9 e F10 apresentam maior quantidade de sucessos, e o que os separa dos demais é serem problemas com 2 dimensões e terem mais pontos em seus conjuntos de dados. É possível associar que a maior quantidade de informação para treinamento torna a avaliação de soluções mais rica, sendo mais expressiva a diferença entre

Figura 19 – Perfis de Desempenho para as melhores recombinações de cada algoritmo.



Fonte: Elaborado pelo próprio autor (2024).

soluções boas e ruins. Por serem problemas com uma dimensão a mais, e terem operações trigonométricas que são mais sensíveis do que outras presentes em outras funções, F9 e F10 deveriam ter um desempenho pior, porém a maior riqueza de informações sobrepõe a dificuldade. O F7 foi o problema que teve a menor quantidade de sucessos, apenas 1 obtido pelo SCPS. Essa função é formada pela soma de dois logaritmos com diferentes argumentos, o que o torna difícil de ser resolvida, mesmo que em um domínio restrito.

O algoritmo que acumula maior quantidade de sucessos é o RSC, seguido pelo DEF com 4 sucessos a menos. Depois SCPS, SSC e SAC, seguindo assim a sequência de maior área no PP. Já o algoritmo que mais vezes consegue o melhor resultado em um problema

é o DEF, em 4 problemas, seguido por RSC e SCPS em 3, SSC em 1 e SAC em nenhum.

Tabela 5 – Contagem de sucessos. O algoritmo com mais sucessos é o RSC. O Algoritmo que obtém mais sucessos no maior número de problemas é o DEF. O problema com menor quantidade de sucessos é o F7.

Problemas	RSC	DEF	SCPS	SSC	SAC	Sucessos/problema
F1	16	**25**	**25**	17	9	92
F2	6	6	**11**	5	4	32
F3	**3**	**3**	2	2	0	10
F4	0	**2**	0	1	1	4
F5	6	**9**	3	1	0	19
F6	**10**	5	6	2	4	27
F7	0	0	1	0	0	1
F8	6	5	**7**	1	2	21
F9	**55**	51	44	36	33	219
F10	33	25	27	**44**	24	153
Sucessos/algoritmo	**135**	131	126	109	77	578
Total/algoritmo	3	**4**	3	1	0	

Fonte: Elaborado pelo próprio autor (2024).

6.2.5 Avaliação Estatística

Os perfis de desempenho são ferramentas que apresentam uma visualização intuitiva e muito rica. No entanto, é possível que outras formas de avaliação dos resultados tragam outras interpretações. O teste estatístico de Kruskal-Wallis pode ajudar a indicar se há diferenças estatisticamente significativas entre os desempenhos entre o resultado dos algoritmos.

Para averiguar se há diferenças significativas, para cada problema, iremos comparar os resultados dos algoritmos com aquele que apresentou o melhor desempenho naquele problema. É criada a matriz M, com p linhas (problemas) e a colunas (algoritmos). Inicialmente M é preenchida com zero em todas as posições. Para cada problema é definido quais algoritmos alcançaram o melhor resultado, e atribuído o valor 1 em M. Em seguida, é realizado o teste estatístico $KW(a^p_{best}, a^p_j)$ onde a^p_{best} são os resultados do melhor algoritmo do problema p e a^p_j os resultados do j-ésimo algoritmo para o problema p. Caso o p-valor seja maior que $0,05$, não há significância estatística, mostrando que o algoritmo é similar ao melhor, e então é atribuído 1 na matriz. Ao final, a matriz terá para cada problema o valor 1 na posição do melhor algoritmo e daqueles que são similares a ele. Somando os valores de cada algoritmo pode-se ver seu desempenho geral para os problemas. O Algoritmo 3 detalha melhor o processo.

A Tabela 6 exibe as comparações estatísticas dos algoritmos. Os resultados mostram que o RSC, SCPS e DEF são estatisticamente similares em quase todos os problemas, sendo ou o melhor ou com resultados estatisticamente similares ao melhor algoritmo. Eles

Algoritmo 3: Teste Estatístico.

1 Matriz $M[p, a] = 0$
2 Para cada problema p_i encontra a_j com melhor desempenho e faz $M[p_i, a_j] = 1$
3 Para cada problema p_i, se $p_i \neq P_{best}$, calcula $KW(p_{best}, a_j)$ e se maior que 0.05 então $M[p_i, a_j] = 1$
4 Faz a contagem dos valores de cada algoritmo somando a coluna e verifica os que tem maior valor

aparecem 9 vezes como melhor ou similar, sendo que RSC e SCPS somam 1 em todos os problemas exceto o F10, e o SSC em todos exceto o F9. Em sequência, vem o SSC somando 6, e, por último, o SAC que soma 0, não sendo o melhor e nem similar ao melhor em nenhum problema.

Tabela 6 – Análise estatística.

Problemas	RSC	DEF	SCPS	SSC	SAC
F1	1	1	1	1	0
F2	1	1	1	0	0
F3	1	1	1	0	0
F4	1	1	1	1	0
F5	1	1	1	1	0
F6	1	1	1	1	0
F7	1	1	1	0	0
F8	1	1	1	1	0
F9	1	0	1	1	0
F10	0	1	0	0	0
Total	**9**	**9**	**9**	6	0

Fonte: Elaborado pelo próprio autor (2024).

Ambos RSC e SCPS têm os mesmos valores na Tabela 6, e não conseguiram os melhores resultados no problema F10. Esse problema é um dos dois problemas que têm 2 dimensões, é composto por operações de produto, seno e cosseno. Também contém a constante 2, enquanto não existem constantes na gramática, deixando para o processo evolutivo gerar genes que expressem esse valor através de operações aritméticas.

6.3 Discussão

Esta seção irá discutir as propostas do trabalho em relação a seus resultados. Inicialmente será discutido o impacto da escolha de medida de distância, e em seguida o impacto da variação paramétrica no RSC.

6.3.1 Impacto dos Métodos de Distância

Os métodos de cálculo de distância, como a distância absoluta e a similaridade cosseno, mostraram desempenhos diferentes entre os métodos de recombinação. A distância

absoluta geralmente apresentou melhor desempenho em comparação com a similaridade cosseno, especialmente em combinações com limites superiores pequenos. Ainda, o cálculo de COS envolve uma complexidade maior que o de ABS, que é mais amplamente utilizado na literatura. No entanto, COS aparece na melhor variação paramétrica de alguns algoritmos, mostrando que pode ser uma boa métrica de distância dependendo do método utilizado. A interpretação geométrica de COS é aferir o quanto dois vetores apontam para a mesma direção, independente de sua magnitude. Em recuperação da informação, essa informação indica que esses vetores representam uma informação similar.

6.3.2 Impacto dos Parâmetros do RSC

O RSC têm como parâmetros os limites superior e inferior delimitando uma região onde a distância é aceita para a recombinação. Os resultados evidenciam alguns comportamentos anteriormente previstos. Um exemplo é na utilização de limite inferior zero, que poderia levar a dominação da roleta por sub-árvores muito similares, diminuindo a efetividade da recombinação. Também valores muito altos de limite superior mostraram um desempenho inferior, dado que uma maior diferença semântica é prejudicial para a localidade. Além disso, é possível observar na Figura 15b que ambos parâmetros geram alteração nos resultados.

7 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho apresenta a aplicação de semântica na PGG, em especial, através do operador de recombinação. Através de diversas abordagens existentes na literatura, é evidenciado que o uso de semântica é benefício para técnicas de PG em geral. No entanto poucos trabalhos abordam o uso de semântica junto da PGG, o que motivou o desenvolvimento deste trabalho, a fim de verificar o desempenho ao resolver problemas de RS. Para isso, os operadores de recombinação da literatura SAC e SSC foram adaptados para funcionar com a PGG, e também foi testado o SCPS, além do operador proposto neste trabalho. O operador proposto RSC utiliza a informação semântica para criar uma roleta, onde as probabilidades de escolha são distribuídas baseado na similaridade semântica dos candidatos.

Os resultados sugerem que a incorporação de informações semânticas nos operadores de recombinação pode melhorar significativamente a eficiência da PGG em problemas de regressão simbólica. Em especial, métodos mais recentes de recombinação como o SCPS se mostraram superiores a outros métodos mais antigos da literatura, como SSC e SAC. No entanto, também foi observado que o *benchmark* utilizado, composto por 10 funções com uma a duas dimensões, pode apresentar resultados diferentes da literatura quando aplicada a estratégia de validação cruzada. Uma diferença em relação aos resultados da literatura é o desempenho da recombinação padrão da PGG, se apresentando como um método mais robusto que recombinações semânticas em alguns casos. O algoritmo aqui proposto, RSC, apresenta a maior robustez por ter a maior área sob a curva dos Perfis de Desempenho dentre os melhores algoritmos, mesmo que os resultados estatísticos apontem outros métodos como sendo similares à ele.

Suas variações paramétricas mostram que os melhores resultados são obtido nos menores valores de limite inferior. Isso indica que eles são os melhores valores, e que valores ainda menores podem ser investigados para avaliar se podem obter um desempenho ainda melhor do algoritmo. No entanto, valores ainda menores abrirão possibilidades de distâncias muito pequenas, o que fará com que sub-árvores similares sempre dominem a roleta. Já para o limite superior, um valor não muito alto se mostrou o ideal. Como quanto maior a distância, menor a chance de ser escolhido na roleta, essas sub-árvores tendem a não ser muito escolhidas, mas ainda assim a variação desse parâmetro causa impacto nas soluções como observado nas Figuras 15b e 17b.

Apesar dos bons resultados alcançados pela recombinação RSC proposta aqui, ainda há espaço para explorar mais a relação da semântica na PGG. Trabalhos futuros podem explorar variações paramétricas adicionais e aplicar operadores de recombinação semântica em outras classes de problemas. Problemas de regressão envolvendo equações funcionais e a inferência de redes regulatórias gênicas, um problema que se mostra difícil de resolver

apenas com PGG, podem se beneficiar do uso desses operadores, pois representam uma rede onde a alteração de um gene de um elemento da rede altera todos outros elementos com quem ele é relacionado. Logo, alterações nos modelos realizadas através dos operadores genéticos que sejam brandas, mas ainda sejam significativas, são bem vindas. Além disso, é possível ainda explorar outras formas de recombinação semântica, como utilizando a semântica para fazer a seleção dos indivíduos que participarão da recombinação, explorar a semântica na mutação e também como estratégia de substituição da população. Uma possibilidade de recombinação semântica, similar à RSC mas com uma diferente estratégia é elaborar uma classificação de seleção de subárvores: ordenando as candidatas em relação à semântica. Em seguida são montadas as probabilidades com base no ranking de cada candidato, por exemplo com uma distribuição linear ou exponencial. Dessa forma, a chance de seleção de um candidato é influenciada pela semântica, mas não de forma proporcional como no RSC.

REFERÊNCIAS

Dimitrios Angelis, Filippos Sofos, and Theodoros E. Karakasidis. Artificial intelligence in physical sciences: Symbolic regression trends and perspectives. *Archives of Computational Methods in Engineering*, 30(6):3845–3865, April 2023. ISSN 1886-1784. doi: 10.1007/s11831-023-09922-z. URL http://dx.doi.org/10.1007/s11831-023-09922-z.

Dirk V. Arnold and Hans-Georg Beyer. A comparison of evolution strategies with other direct search methods in the presence of noise. *Computational Optimization and Applications*, 24(1):135–159, 2003. ISSN 0926-6003. doi: 10.1023/a:1021810301763. URL http://dx.doi.org/10.1023/A:1021810301763.

Helio J. C. Barbosa, Heder S. Bernardino, and André M. S. Barreto. Using performance profiles to analyze the results of the 2006 CEC constrained optimization competition. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8. IEEE, 2010.

Lawrence Beadle and Colin G. Johnson. Semantically driven crossover in genetic programming. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 111–116, 2008. doi: 10.1109/CEC.2008.4630784.

Lawrence Beadle and Colin G. Johnson. Semantically driven mutation in genetic programming. In *2009 IEEE Congress on Evolutionary Computation*, pages 1336–1342, 2009. doi: 10.1109/CEC.2009.4983099.

H.S. Bernardino. *Programação imunológica gramatical para inferência automática de modelos e projeto ótimo de estruturas*. PhD thesis, Laboratório Nacional de Computação Científica (LNCC/MCTI), 2012.

Josh Bongard and Hod Lipson. Automated reverse engineering of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 104(24):9943–9948, June 2007. ISSN 1091-6490. doi: 10.1073/pnas.0609476104. URL http://dx.doi.org/10.1073/pnas.0609476104.

Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986. doi: 10.1109/TC.1986.1676819.

Xi Cheng, Bohdan Khomtchouk, Norman Matloff, and Pete Mohanty. Polynomial regression as an alternative to neural nets, 2019. URL https://arxiv.org/abs/1806.06850.

Robert Cleary and Michael O'Neill. *An Attribute Grammar Decoder for the 01 Multi-Constrained Knapsack Problem*, page 34–45. Springer Berlin Heidelberg, 2005. ISBN 9783540319962. doi: 10.1007/978-3-540-31996-2_4. URL http://dx.doi.org/10.1007/978-3-540-31996-2_4.

Charles Darwin. *On the Origin of Species by Means of Natural Selection.* Murray, London, 1859. or the Preservation of Favored Races in the Struggle for Life.

J.M. de Freitas, H.S. Bernardino, J.N. Guerreiro, and H.J.C. Barbosa. Aplicação de uma programação genética gramatical multiobjetivo na inferência da máxima deformação longitudinal de dutos com amassamento. In *Proc. of the Iberian Latin American Congress on Computational Methods in Engineering (CILAMCE)*, 2015.

João Marcos de Freitas, Felipe Rafael de Souza, and Heder S. Bernardino. Evolving controllers for mario ai using grammar-based genetic programming. In *2018 IEEE Congress on Evolutionary Computation (CEC)*, page 1–8. IEEE Press, 2018. doi: 10.1109/CEC.2018.8477698. URL https://doi.org/10.1109/CEC.2018.8477698.

João Marcos de Freitas, Heder Soares Bernardino, Luciana Brugiolo Gonçalves, and Stěnio Sã Rosário Furtado Soares. Human activity recognition using grammar-based genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '22, page 699–702, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392686. doi: 10.1145/3520304.3529076. URL https://doi.org/10.1145/3520304.3529076.

Kenneth De Jong. Evolutionary computation: a unified approach. In *Proceedings of the 12th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '10, page 2289–2302, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781450300735. doi: 10.1145/1830761.1830896. URL https://doi.org/10.1145/1830761.1830896.

Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Math. Program.*, 91(2):201–213, Jan 2002.

Issam El Naqa and Martin J. Murphy. *What Is Machine Learning?*, page 3–11. Springer International Publishing, 2015. ISBN 9783319183053. doi: 10.1007/978-3-319-18305-3_1. URL http://dx.doi.org/10.1007/978-3-319-18305-3_1.

Stefan Forstenlechner. *Program Synthesis with Grammars and Semantics in Genetic Programming.* PhD thesis, University College Dublin, Ireland, January 2019. URL https://www.smurfitschool.ie/facultyresearch/phdresearch/phdgraduates/stefanforstenlechner/.

Richard Forsyth. BEAGLE a Darwinian approach to pattern recognition. *Kybernetes*, 10 (3):159–166, 1981. ISSN 0368-492X. doi: doi:10.1108/eb005587. URL http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/kybernetes_forsyth.pdf.

John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* The MIT Press,

April 1992. ISBN 9780262275552. doi: 10.7551/mitpress/1090.001.0001. URL http://dx.doi.org/10.7551/mitpress/1090.001.0001.

Hsin-Hsiung Huang and Qing He. *Nonlinear regression analysis*, page 558–567. Elsevier, 2023. ISBN 9780128186299. doi: 10.1016/b978-0-12-818630-5.10068-5. URL http://dx.doi.org/10.1016/B978-0-12-818630-5.10068-5.

Colin G. Johnson. *Deriving Genetic Programming Fitness Properties by Static Analysis*, page 298–307. Springer Berlin Heidelberg, 2002. ISBN 9783540459842. doi: 10.1007/3-540-45984-7_29. URL http://dx.doi.org/10.1007/3-540-45984-7_29.

Colin G. Johnson. *Genetic Programming with Guaranteed Constraints*, page 95–100. Springer Berlin Heidelberg, 2004. ISBN 9783540452409. doi: 10.1007/978-3-540-45240-9_14. URL http://dx.doi.org/10.1007/978-3-540-45240-9_14.

Colin G. Johnson. Genetic programming with fitness based on model checking. In Marc Ebner, Michael O'Neill, Anikó Ekárt, Leonardo Vanneschi, and Anna Isabel Esparcia-Alcázar, editors, *Genetic Programming*, pages 114–124, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-71605-1.

Kush Juvekar and Anupam Purwar. Cos-mix: Cosine similarity and distance fusion for improved information retrieval, 2024. URL https://arxiv.org/abs/2406.00638.

Maarten Keijzer. *Improving Symbolic Regression with Interval Arithmetic and Linear Scaling*, page 70–82. Springer Berlin Heidelberg, 2003. ISBN 9783540365990. doi: 10.1007/3-540-36599-0_7. URL http://dx.doi.org/10.1007/3-540-36599-0_7.

Donald E. Knuth. Semantics of context-free languages. *Mathematical Systems Theory*, 2(2):127–145, June 1968. ISSN 1433-0490. doi: 10.1007/bf01692511. URL http://dx.doi.org/10.1007/BF01692511.

John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992. ISBN 0-262-11170-5. URL http://mitpress.mit.edu/books/genetic-programming.

Krzysztof Krawiec and Pawel Lichocki. Approximating geometric crossover in semantic space. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, GECCO '09, page 987–994, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605583259. doi: 10.1145/1569901.1570036. URL https://doi.org/10.1145/1569901.1570036.

John Langford. *The Cross Validation Problem*, page 687–688. Springer Berlin Heidelberg, 2005. ISBN 9783540318927. doi: 10.1007/11503415_47. URL http://dx.doi.org/10.1007/11503415_47.

Todd D. Little. *The Oxford Handbook of Quantitative Methods in Psychology: Vol. 2: Statistical Analysis*. Oxford University Press, 03 2013. ISBN 9780199934898. doi: 10.1093/oxfordhb/9780199934898.001.0001. URL https://doi.org/10.1093/oxfordhb/9780199934898.001.0001.

Wolfgang Banzhaf Markus F. Brameier. *Genetic and Evolutionary Computation*. Springer US, 2007. ISBN 9780387310299. doi: 10.1007/978-0-387-31030-5. URL http://dx.doi.org/10.1007/978-0-387-31030-5.

Robert I. McKay, Nguyen Xuan Hoai, Peter Alexander Whigham, Yin Shan, and Michael O'Neill. Grammar-based genetic programming: a survey. *Genetic Programming and Evolvable Machines*, 11(3–4):365–396, May 2010. ISSN 1573-7632. doi: 10.1007/s10710-010-9109-y. URL http://dx.doi.org/10.1007/s10710-010-9109-y.

Nicholas Freitag McPhee, Brian Ohs, and Tyler Hutchison. *Semantic Building Blocks in Genetic Programming*, page 134–145. Springer Berlin Heidelberg, 2008. ISBN 9783540786719. doi: 10.1007/978-3-540-78671-9_12. URL http://dx.doi.org/10.1007/978-3-540-78671-9_12.

Julian F Miller, Peter Thomson, Terence Fogarty, et al. Designing electronic circuits using evolutionary algorithms. arithmetic circuits: A case study. *Genetic algorithms and evolution strategies in engineering and computer science*, pages 105–131, 1997.

Frederico José Dias Möller, Heder Soares Bernardino, Stênio Sã Rosário Furtado Soares, and Lucas Augusto Müller de Souza. An adaptive mutation for cartesian genetic programming using an ϵ-greedy strategy. *Applied Intelligence*, 53(22):27290–27303, September 2023. ISSN 1573-7497. doi: 10.1007/s10489-023-04951-4. URL http://dx.doi.org/10.1007/s10489-023-04951-4.

Flávio A.A. Motta, João M. De Freitas, Felipe R. De Souza, Heder S. Bernardino, Itamar L. De Oliveira, and Helio J.C. Barbosa. A hybrid grammar-based genetic programming for symbolic regression problems. In *2018 IEEE Congress on Evolutionary Computation (CEC)*, page 1–8. IEEE Press, 2018. doi: 10.1109/CEC.2018.8477826. URL https://doi.org/10.1109/CEC.2018.8477826.

Saptarshi Mukherjee and Ruhi Sonal. A reconciliation between cosine similarity and euclidean distance in individual decision-making problems. *Indian Economic Review*, 58 (2):427–431, December 2023. ISSN 2520-1778. doi: 10.1007/s41775-023-00206-8. URL http://dx.doi.org/10.1007/s41775-023-00206-8.

Quang Uy Nguyen, Xuan Hoai Nguyen, and Michael O'Neill. *Semantic Aware Crossover for Genetic Programming: The Case for Real-Valued Function Regression*, page 292–302. Springer Berlin Heidelberg, 2009. ISBN 9783642011818. doi: 10.1007/978-3-642-01181-8_25. URL http://dx.doi.org/10.1007/978-3-642-01181-8_25.

Patryk Orzechowski, William La Cava, and Jason H. Moore. Where are we now?: a large benchmark study of recent symbolic regression methods. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '18. ACM, July 2018. doi: 10.1145/3205455.3205539. URL http://dx.doi.org/10.1145/3205455.3205539.

Conor Ryan, J. J. Collins, and Michael O'Neill. Grammatical evolution: Evolving programs for an arbitrary language. In Wolfgang Banzhaf, Riccardo Poli, Marc Schoenauer, and Terence C. Fogarty, editors, *Proceedings of the First European Workshop on Genetic Programming*, volume 1391 of *LNCS*, pages 83–96, Paris, 14-15 April 1998. Springer-Verlag. ISBN 3-540-64360-5. doi: doi:10.1007/BFb0055930.

Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, April 2009. ISSN 1095-9203. doi: 10.1126/science.1165893. URL http://dx.doi.org/10.1126/science.1165893.

Jan Scholz. Genetic algorithms and the traveling salesman problem a historical review. *ArXiv*, abs/1901.05737, 2019. URL https://api.semanticscholar.org/CorpusID:58014207.

Hossein Sharifipour, Mojtaba Shakeri, and Hassan Haghighi. Structural test data generation using a memetic ant colony optimization based on evolution strategies. *Swarm and Evolutionary Computation*, 40:76–91, June 2018. ISSN 2210-6502. doi: 10.1016/j.swevo.2017.12.009. URL http://dx.doi.org/10.1016/j.swevo.2017.12.009.

Bharath Srinivasan. Explicit treatment of non-michaelis-menten and atypical kinetics in early drug discovery **. *ChemMedChem*, 16, 2020. URL https://api.semanticscholar.org/CorpusID:227157473.

W.A. Tackett and A. Carmi. The unique implications of brood selection for genetic programming. In *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, pages 160–165 vol.1, 1994. doi: 10.1109/ICEC.1994.350023.

Nguyen Quang Uy, Nguyen Xuan Hoai, Michael O'Neill, R. I. McKay, and Edgar Galván-López. Semantically-based crossover in genetic programming: application to real-valued symbolic regression. *Genetic Programming and Evolvable Machines*, 12(2): 91–119, July 2010. ISSN 1573-7632. doi: 10.1007/s10710-010-9121-2. URL http://dx.doi.org/10.1007/s10710-010-9121-2.

R. V. Veiga, J. M. de Freitas, H. S. Bernardino, H. J. C. Barbosa, and N. M. Alcântara-Neves. Using grammar-based genetic programming to determine characteristics of multiple infections and environmental factors in the development of allergies and asthma. In *2015 IEEE Congress on Evolutionary Computation (CEC)*, pages 1604–1611, May 2015. doi: 10.1109/CEC.2015.7257079.

Rafael V. Veiga, Helio J. C. Barbosa, Heder S. Bernardino, João M. Freitas, Caroline A. Feitosa, Sheila M. A. Matos, Neuza M. Alcântara-Neves, and Maurício L. Barreto. Multiobjective grammar-based genetic programming applied to the study of asthma and allergy epidemiology. *BMC Bioinformatics*, 19(1), June 2018. ISSN 1471-2105. doi: 10.1186/s12859-018-2233-z. URL `http://dx.doi.org/10.1186/s12859-018-2233-z`.

Kit Po Wong and ZhaoYang Dong. Differential evolution, an alternative approach to evolutionary algorithm, June 2007. URL `http://dx.doi.org/10.1002/9780470225868.ch9`.

Man Leung Wong and Kwong Sak Leung. An induction system that learns programs in different programming languages using genetic programming and logic grammars. In *Proceedings of 7th IEEE International Conference on Tools with Artificial Intelligence*, pages 380–387, 1995. doi: 10.1109/TAI.1995.479782.