

Universidade Federal de Uberlândia - UFU
Faculdade de Engenharia Elétrica
Programa de Pós-Graduação em Engenharia Elétrica



Operador de Recombinação para
Programação Genética baseado em
Regressão Linear Múltipla

Leonardo Garcia Marques

Uberlândia – MG

2019

Leonardo Garcia Marques

Operador de Recombinação para Programação Genética baseado em Regressão Linear Múltipla

Tese apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Uberlândia, como requisito parcial para a obtenção do título de Doutor em Ciências.

Orientador: PhD. Keiji Yamanaka
Coorientador: Dr. Igor Santos Peretta

Uberlândia – MG
2019

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da UFU, MG, Brasil.

M357o Marques, Leonardo Garcia, 1983-
2019 Operador de recombinação para Programação Genética baseado em
Regressão Linear Múltipla [recurso eletrônico] / Leonardo Garcia
Marques. - 2019.

Orientador: keiji Yamanaka.

Coorientador: Igor Santos Peretta.

Tese (Doutorado) - Universidade Federal de Uberlândia, Programa
de Pós-Graduação em Engenharia Elétrica.

Disponível em: <http://dx.doi.org/10.14393/ufu.te.2019.630>

Inclui bibliografia.

Inclui ilustrações.

1. Engenharia elétrica. 2. Programação genética (Computação). 3.
Análise de regressão - Processamento de dados. I. Yamanaka, keiji,
1956- (Orient.). II. Peretta, Igor Santos, 1974- (Coorient.). III.
Universidade Federal de Uberlândia. Programa de Pós-Graduação em
Engenharia Elétrica. IV. Título.

CDU: 621.3

Gerlaine Araújo Silva - CRB-6/1408



UNIVERSIDADE FEDERAL DE UBERLÂNDIA

ATA DE DEFESA

Programa de Pós-Graduação em:	Engenharia Elétrica				
Defesa de:	Tese de Doutorado, 237, COPEL				
Data:	22 de março de 2019	Hora de início:	13:00	Hora de encerramento:	16:10
Matrícula do Discente:	11423EEL007				
Nome do Discente:	Leonardo Garcia Marques				
Título do Trabalho:	Operador de recombinação para programação genética baseado em regressão linear múltipla				
Área de concentração:	Processamento da Informação				
Linha de pesquisa:	Inteligência Artificial				
Projeto de Pesquisa de vinculação:	Estudo e implementação de técnicas de inteligência computacional em problemas complexos				

Reuniu-se no Anfiteatro do Bloco 1E, Campus Santa Mônica, da Universidade Federal de Uberlândia, a Banca Examinadora, designada pelo Colegiado do Programa de Pós-graduação em Engenharia Elétrica, assim composta: Professores Doutores: Igor Santos Peretta - UFU; Wesley Pacheco Calixto - IFG; Frederico Gadelha Guimarães - UFMG; Gerson Flávio Mendes de Lima - ULBRA; e Keiji Yamanaka - UFU, orientador do candidato.

Iniciando os trabalhos o presidente da mesa, Dr. Keiji Yamanaka, apresentou a Comissão Examinadora e o candidato, agradeceu a presença do público, e concedeu ao Discente a palavra para a exposição do seu trabalho. A duração da apresentação do Discente e o tempo de arguição e resposta foram conforme as normas do Programa.

A seguir o senhor(a) presidente concedeu a palavra, pela ordem sucessivamente, aos(às) examinadores(as), que passaram a arguir o(a) candidato(a). Ultimada a arguição, que se desenvolveu dentro dos termos regimentais, a Banca, em sessão secreta, atribuiu o resultado final, considerando o(a) candidato(a):

Aprovado.

Esta defesa faz parte dos requisitos necessários à obtenção do título de **Doutor**.

O competente diploma será expedido após cumprimento dos demais requisitos, conforme as normas do Programa, a legislação pertinente e a regulamentação interna da UFU.

Nada mais havendo a tratar foram encerrados os trabalhos. Foi lavrada a presente ata que após lida e achada conforme foi assinada pela Banca Examinadora.



Documento assinado eletronicamente por **Frederico Gadelha Guimaraes, Usuário Externo**, em 22/03/2019, às 18:19, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do [Decreto nº 8.539, de 8 de outubro de 2015](#).



Documento assinado eletronicamente por **Gerson Flavio Mendes de Lima, Usuário Externo**, em 25/03/2019, às 10:59, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do [Decreto nº 8.539, de 8 de outubro de 2015](#).



Documento assinado eletronicamente por **Igor Santos Peretta, Professor(a) do Magistério Superior**, em 22/04/2019, às 15:52, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do [Decreto nº 8.539, de 8 de outubro de 2015](#).



Documento assinado eletronicamente por **Keiji Yamanaka, Membro de Comissão**, em 22/04/2019, às 16:23, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do [Decreto nº 8.539, de 8 de outubro de 2015](#).



Documento assinado eletronicamente por **Wesley Pacheco Calixto, Usuário Externo**, em 25/04/2019, às 08:20, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do [Decreto nº 8.539, de 8 de outubro de 2015](#).



A autenticidade deste documento pode ser conferida no site https://www.sei.ufu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **1086342** e o código CRC **B2D01B79**.

Resumo

Marques, L. **Operador de Recombinação para Programação Genética baseado em Regressão Linear Múltipla**. 168 p. Tese – Faculdade de Engenharia Elétrica, Universidade Federal de Uberlândia - UFU, 2019.

A Programação Genética (PG) é uma técnica de Computação Evolucionária que evolui indivíduos com genótipo de tamanho e formato variáveis. A expressividade na representação faz da PG uma ferramenta bastante útil na Engenharia, produzindo resultados competitivos com a inteligência humana. Uma de suas aplicações mais comuns é a descoberta automática de modelos a partir da análise de dados, o que é conhecido por Regressão Simbólica. A utilização da PG requer cuidadosa implementação de seus operadores genéticos, principalmente a recombinação e a mutação. Abordagens clássicas para a criação dos operadores costumam basear-se nas características sintáticas dos indivíduos, ao passo que técnicas recentes são direta ou indiretamente guiadas pela semântica. Há ainda abordagens que combinam Análise de Regressão e Computação Evolucionária para obter respostas com maior qualidade. De maneira geral, todos esses operadores estão sujeitos a um mesmo problema: a quantidade de genes que compõem os indivíduos começa a aumentar descontroladamente após algumas poucas gerações sem que haja melhorias na aptidão, comprometendo a qualidade das respostas produzidas. Este fenômeno é denominado efeito *bloat*. O objetivo deste trabalho é apresentar um novo operador para Programação Genética que possibilite evoluir populações que possam apresentar indivíduos mais acurados e de representação estrutural cujo tamanho seja naturalmente controlado. O operador desenvolvido simultaneamente atua na recombinação e na mutação, promovendo herança variacional e diversidade populacional. Por proporcionar a produção de indivíduos de alta qualidade enquanto evolui populações sem os efeitos nocivos associados ao *bloat*, o operador desenvolvido mostrou-se superior à clássica recombinação de subárvore, a novos operadores genéticos baseados em semântica e também a outras técnicas recentes baseadas em Análise de Regressão.

Palavras-chave: Programação Genética. Regressão Simbólica. Análise de Regressão. Efeito *Bloat*.

Abstract

Marques, L. **Recombination Operator for Genetic Programming based on Multiple Linear Regression**. 168 p. Ph.D. Thesis – Faculty of Electrical Engineering, Federal University of Uberlândia, 2019.

Genetic Programming (GP) is a technique of Evolutionary Computation that evolves individuals of variable size and shape. The expressiveness in the representation makes GP a very useful tool in Engineering, producing competitive results with human intelligence. One of its most common applications is the automatic discovery of models from data analysis, which is known as Symbolic regression. The use of GP requires careful implementation of its genetic operators, especially recombination and mutation. Classical approaches to operator creation are often based on the syntactic characteristics of individuals, while recent techniques are directly or indirectly guided by semantics. There are also approaches that combine Regression Analysis and Evolutionary Computation for higher quality responses. In general, all these operators are subject to the same problem: the number of genes that make up individuals begins to increase wildly after a few generations without improvements in aptitude, compromising the quality of the responses produced. This phenomenon is called bloat effect. The objective of this work is to present a new operator for Genetic Programming that allows the evolution of populations that can present more accurate and structural individuals whose size is naturally controlled. The developed operator simultaneously acts on recombination and mutation, promoting variational inheritance and population diversity. By providing the production of high-quality individuals while evolving populations without the harmful effects associated with bloat, the developed operator proved to be superior to classical subtree recombination, to new genetic operators based on semantics, and also to other recent techniques based on Analysis of Regression.

Keywords: Genetic Programming. Symbolic Regression. Regression Analysis. Bloat Effect.

Lista de ilustrações

Figura 1 – Árvore sintática para código em LISP.	30
Figura 2 – Árvore da expressão matemática $((y - x) + \sin(x * x)) * (x - 2)$	30
Figura 3 – Criação de uma árvore.	32
Figura 4 – Exemplos de árvores geradas pelos métodos <i>full</i> e <i>grow</i>	33
Figura 5 – Recombinação entre dois indivíduos.	36
Figura 6 – Nós viáveis e inviáveis em um árvore	41
Figura 7 – Árvores de diferentes formatos, mas com mesma aptidão.	43
Figura 8 – Gráfico de dispersão.	45
Figura 9 – Avaliação de uma árvore nó a nó.	52
Figura 10 – Superfície gerada pela Equação (3.1).	62
Figura 11 – Área de seleção de subárvores para o MRX.	65
Figura 12 – Exemplo com o passo a passo do operador MRX.	68
Figura 13 – Interface gráfica utilizada na realização dos experimentos.	74
Figura 14 – Gráfico de caixa (<i>boxplot</i>) para visualização da dispersão dos dados. . .	79
Figura 15 – Dispersão dos erros das respostas fornecidas no treinamento das funções do <i>benchmark</i> Koza: efeito do acréscimo de subárvores aleatórias. . . .	82
Figura 16 – Dispersão dos erros das respostas fornecidas no treinamento das funções do <i>benchmark</i> Ngueyn: efeito do acréscimo de subárvores aleatórias. . .	83
Figura 17 – Dispersão dos erros das respostas fornecidas no treinamento das funções do <i>benchmark</i> Keijzer: efeito do acréscimo de subárvores aleatórias. . .	84
Figura 18 – Dispersão dos erros das respostas fornecidas no treinamento das funções do <i>benchmark</i> Vladislavleva: efeito do acréscimo de subárvores aleatórias. .	85
Figura 19 – Dispersão dos erros das respostas fornecidas nos testes das funções do <i>benchmark</i> Koza: efeito do acréscimo de subárvores aleatórias.	87
Figura 20 – Dispersão dos erros das respostas fornecidas nos testes das funções do <i>benchmark</i> Ngueyn: efeito do acréscimo de subárvores aleatórias.	88
Figura 21 – Dispersão dos erros das respostas fornecidas nos testes das funções do <i>benchmark</i> Keijzer: efeito do acréscimo de subárvores aleatórias.	89

Figura 22 – Dispersão dos erros das respostas fornecidas nos testes das funções do <i>benchmark</i> Vladislavleva: efeito do acréscimo de subárvores aleatórias.	90
Figura 23 – Dispersão dos tamanhos das respostas fornecidas no treinamento das funções do <i>benchmark</i> Koza: efeito do acréscimo de subárvores aleatórias.	91
Figura 24 – Dispersão dos tamanhos das respostas fornecidas no treinamento das funções do <i>benchmark</i> Nguenyn: efeito do acréscimo de subárvores aleatórias.	93
Figura 25 – Dispersão dos tamanho das respostas fornecidas no treinamento das funções do <i>benchmark</i> Keijzer: efeito do acréscimo de subárvores aleatórias.	94
Figura 26 – Dispersão dos tamanhos das respostas fornecidas no treinamento das funções do <i>benchmark</i> Vladislavleva: efeito do acréscimo de subárvores aleatórias.	95
Figura 27 – O <i>bloat</i> em funções dos <i>benchmarks</i> Koza, Nguyen, Keijzer e Vladislavleva: efeito do acréscimo de subárvores aleatórias.	102
Figura 28 – Dispersão dos erros das respostas fornecidas no treinamento das funções do <i>benchmark</i> Koza: comparação entre o MRX e a recombinação de subárvores.	106
Figura 29 – Dispersão dos erros das respostas fornecidas no treinamento das funções do <i>benchmark</i> Nguenyn: comparação entre o MRX e a recombinação de subárvores.	107
Figura 30 – Dispersão dos erros das respostas fornecidas no treinamento das funções do <i>benchmark</i> Keijzer: comparação entre o MRX e a recombinação de subárvores.	108
Figura 31 – Dispersão dos erros das respostas fornecidas no treinamento das funções do <i>benchmark</i> Vladislavleva: comparação entre o MRX e a recombinação de subárvores.	109
Figura 32 – Dispersão dos erros das respostas fornecidas nos testes das funções do <i>benchmark</i> Koza: comparação entre o MRX e a recombinação de subárvores.	110
Figura 33 – Dispersão dos erros das respostas fornecidas nos testes das funções do <i>benchmark</i> Nguenyn: comparação entre o MRX e a recombinação de subárvores.	112
Figura 34 – Dispersão dos erros das respostas fornecidas nos testes das funções do <i>benchmark</i> Keijzer: comparação entre o MRX e a recombinação de subárvores.	113
Figura 35 – Dispersão dos erros das respostas fornecidas nos testes das funções do <i>benchmark</i> Vladislavleva: comparação entre o MRX e a recombinação de subárvores.	114

Figura 36 – Dispersão dos tamanhos das respostas fornecidas no treinamento das funções do <i>benchmark</i> Koza: comparação entre o MRX e a recombinação de subárvores.	115
Figura 37 – Dispersão dos tamanhos das respostas fornecidas no treinamento das funções do <i>benchmark</i> Ngueyn: comparação entre o MRX e a recombinação de subárvores.	117
Figura 38 – Dispersão dos tamanho das respostas fornecidas no treinamento das funções do <i>benchmark</i> Keijzer: comparação entre o MRX e a recombinação de subárvores.	118
Figura 39 – Dispersão dos tamanhos das respostas fornecidas no treinamento das funções do <i>benchmark</i> Vladislavleva.	119
Figura 40 – Efeito <i>bloat</i> em funções dos <i>benchmarks</i> Koza, Nguyen, Keijzer e Vladislavleva.	126
Figura 41 – Dispersão dos erros das respostas fornecidas no treinamento das funções do <i>benchmark</i> Koza.	129
Figura 42 – Dispersão dos erros das respostas fornecidas no treinamento das funções do <i>benchmark</i> Nguyen.	130
Figura 43 – Dispersão dos erros das respostas fornecidas no teste das funções do <i>benchmark</i> Koza.	132
Figura 44 – Dispersão dos erros das respostas fornecidas no teste das funções do <i>benchmark</i> Nguyen.	132
Figura 45 – Dispersão dos tamanhos (quantidade de nós) das respostas fornecidas no treinamento das funções do <i>benchmark</i> Koza.	134
Figura 46 – Dispersão dos tamanhos (quantidade de nós) das respostas fornecidas no treinamento das funções do <i>benchmark</i> Nguyen.	134
Figura 47 – Efeito <i>bloat</i> em funções dos <i>benchmarks</i> Koza e Nguyen, comparando o MRX e os operadores semânticos.	137
Figura 48 – Distribuição dos erros das respostas fornecidas pelo testes das funções dos <i>benchmarks</i> Koza e Nguyen.	140
Figura 49 – Distribuição dos tamanhos (quantidade de nós) observados nos melhores indivíduos encontrados ao final treinamento nas funções dos <i>benchmarks</i> Koza e Nguyen.	142
Figura 50 – Koza-2: comparação entre as curvas geradas pela recombinação de subárvore e pelo operador MRX.	146
Figura 51 – Ngueyn-5: comparação entre as curvas geradas pela recombinação de subárvore e pelo operador MRX.	147
Figura 52 – Comportamento do MRX para diferentes quantidades de subárvores. .	148

Lista de tabelas

Tabela 1 – Exemplos de funções comumente utilizadas em PG.	31
Tabela 2 – Exemplos de terminais comumente utilizadas em PG.	31
Tabela 3 – <i>Dataset</i> Nguyen-6.	34
Tabela 4 – Dados para Regressão Linear Múltipla	46
Tabela 5 – <i>Benchmarks</i> para regressão simbólica: Funções Koza	75
Tabela 6 – <i>Benchmarks</i> para regressão simbólica: Funções Nguyen	75
Tabela 7 – <i>Benchmarks</i> para regressão simbólica: Funções Keijzer	76
Tabela 8 – <i>Benchmarks</i> para regressão simbólica: Funções Vladislavleva	76
Tabela 9 – Conjuntos de funções e constantes utilizados por cada <i>benchmark</i>	76
Tabela 10 – Treinamento usando funções do <i>benchmark</i> Koza: efeito do acréscimo de subárvores aleatórias.	81
Tabela 11 – Treinamento usando funções do <i>benchmark</i> Nguyen: efeito do acréscimo de subárvores aleatórias.	81
Tabela 12 – Treinamento usando funções do <i>benchmark</i> Keijzer: efeito do acréscimo de subárvores aleatórias.	81
Tabela 13 – Treinamento usando funções do <i>benchmark</i> Vladislavleva: efeito do acrécimo de subárvores aleatórias.	82
Tabela 14 – Resultado dos testes do <i>benchmark</i> Koza: efeito do acréscimo de su- bárvores aleatórias.	86
Tabela 15 – Resultado dos testes do <i>benchmark</i> Nguyen: efeito do acréscimo de subárvores aleatórias.	86
Tabela 16 – Resultados dos testes do <i>benchmark</i> Keijzer: efeito do acréscimo de subárvores aleatórias.	86
Tabela 17 – Resultado dos testes do <i>benchmark</i> Vladislavleva: efeito do acréscimo de subárvores aleatórias.	87
Tabela 18 – Quantidade de nós observados nas funções Koza: efeito do acréscimo de subárvores aleatórias.	92

Tabela 19 – Quantidade de nós observados nas funções Nguyen: efeito do acréscimo de subárvores aleatórias.	92
Tabela 20 – Quantidade de nós observados nas funções Keijzer: efeito do acréscimo de subárvores aleatórias.	92
Tabela 21 – Quantidade de nós observados nas funções Vladislavleva: efeito do acréscimo de subárvores aleatórias.	93
Tabela 22 – Treinamento usando funções do <i>benchmark</i> Koza: comparação entre o MRX e a recombinação de subárvores.	105
Tabela 23 – Treinamento usando funções do <i>benchmark</i> Nguenyn: comparação entre o MRX e a recombinação de subárvores.	105
Tabela 24 – Treinamento usando funções do <i>benchmark</i> Keijzer: comparação entre o MRX e a recombinação de subárvores.	105
Tabela 25 – Treinamento usando funções do <i>benchmark</i> Vladislavleva: comparação entre o MRX e a recombinação de subárvores.	106
Tabela 26 – Resultado dos testes do <i>benchmark</i> Koza: comparação entre o MRX e a recombinação de subárvores.	111
Tabela 27 – Resultado dos testes do <i>benchmark</i> Nguenyn: comparação entre o MRX e a recombinação de subárvores.	111
Tabela 28 – Resultados dos testes do <i>benchmark</i> Keijzer: comparação entre o MRX e a recombinação de subárvores.	111
Tabela 29 – Resultado dos testes do <i>benchmark</i> Vladislavleva: comparação entre o MRX e a recombinação de subárvores.	112
Tabela 30 – Quantidade de nós observados nas funções Koza: comparação entre o MRX e a recombinação de subárvores.	116
Tabela 31 – Quantidade de nós observados nas funções Nguyen: comparação entre o MRX e a recombinação de subárvores.	116
Tabela 32 – Quantidade de nós observados nas funções Keijzer: comparação entre o MRX e a recombinação de subárvores.	116
Tabela 33 – Quantidade de nós observados nas funções Vladislavleva: comparação entre o MRX e a recombinação de subárvores.	117
Tabela 34 – Treinamento usando funções do <i>benchmark</i> Koza.	128
Tabela 35 – Treinamento usando funções do <i>benchmark</i> Nguyen.	129
Tabela 36 – Resultados dos testes do <i>benchmark</i> Koza.	131
Tabela 37 – Resultados dos testes do <i>benchmark</i> Nguyen.	131
Tabela 38 – Quantidade de nós observados nas funções Koza.	133
Tabela 39 – Quantidade de nós observados nas funções Nguyen.	133
Tabela 40 – Comparação entre os testes do <i>benchmark</i> Koza na GP-MRX e na MRGP.	139

Tabela 41 – Comparação entre os testes do <i>benchmark</i> Koza na GP-MRX e na MRGP.	140
Tabela 42 – Comparação entre os testes do <i>benchmark</i> Koza na GP-MRX e na MRGP.	141
Tabela 43 – Comparação entre os testes do <i>benchmark</i> Koza na GP-MRX e na MRGP.	141
Tabela 44 – Comparação com a Programação <i>Kaizen</i> : treinamento.	144
Tabela 45 – Comparação com a Programação <i>Kaizen</i> : teste.	145

Lista de abreviaturas e siglas

AG	Algoritmos Genéticos
AGX	<i>Semantic Aware Crossover</i>
CE	Computação Evolucionária
DS	Distância Semântica
GSGP	<i>Geometric Semantic Genetic Programming</i>
MAE	<i>Mean Absolute Error</i>
MRGP	<i>Multiple Regression Genetic Programming</i>
MRX	<i>Multiple Regression Crossover</i>
MSE	<i>Mean Squared Error</i>
PDCA	<i>Plan-Do-Check-Act</i>
PG	Programação Genética
PK	Programação Kaizen
RDO	<i>Random Desired Operator</i>
RMSE	<i>Root Mean Squared Error</i>
ROBDD	<i>Reduced Ordered Binary Decision Diagram</i>
SAE	<i>Sum of Absolute Errors</i>
SDC	<i>Semantically Driven Crossover</i>
SGM	<i>Semantic Mutation</i>

SGX *Semantic Crossover*

SSGX *Subtree Semantic Geometric Crossover*

Sumário

1	Introdução	23
1.1	Visão geral	23
1.2	Motivação	24
1.3	Objetivos	25
1.3.1	Objetivos específicos	25
1.4	Contribuições	26
1.5	Organização do trabalho	26
2	Aspectos teóricos e revisão bibliográfica	27
2.1	Computação Evolucionária	27
2.2	Programação Genética	29
2.2.1	Cromossomos	29
2.2.2	Geração da população inicial	32
2.2.3	O cálculo da aptidão	33
2.2.4	Seleção	35
2.2.5	Operadores genéticos	35
2.2.5.1	Abordagem sintática	36
2.2.5.2	Abordagem semântica	37
2.3	O efeito <i>bloat</i>	39
2.3.1	O <i>bloat</i> na recombinação	40
2.3.1.1	Defesa contra a recombinação	40
2.3.1.2	O Viés da remoção	41

2.3.1.3	Profundidade do ponto de cruzamento	42
2.3.2	Problemas relacionados à seleção e à busca direcionada pela aptidão	42
2.4	Análise de regressão em CE	44
2.4.1	Análise de regressão	44
2.4.1.1	Método dos mínimos quadrados	46
2.4.1.2	Coeficiente de determinação múltipla	50
2.4.1.3	Testes de hipóteses sobre os coeficientes de regressão	50
2.4.2	Técnicas de CE baseadas em Regressão Regressão	51
2.4.2.1	MRGP	51
2.4.2.2	Programação <i>Kaizen</i>	53
2.4.3	O problema das matrizes singulares	56
2.4.3.1	A pseudo inversa de Moore-Penrose	57
2.4.3.2	Generalização do método dos mínimos quadrados	58
2.5	Considerações sobre o capítulo	59
3	Definição de operador de recombinação com regressão múltipla	61
3.1	Subárvores como regressores	61
3.2	Operador MRX	62
3.2.1	Configuração da matriz do modelo	64
3.2.1.1	Escolha das subárvores	64
3.2.1.2	Criação da matriz	65
3.2.2	Criação do modelo linear	66
3.2.3	Identificação de modelos válidos	66
3.2.4	Exemplo	67
3.3	Melhorando a diversidade populacional	68
3.4	Controle de <i>bloat</i> pelo MRX	69
3.5	Considerações sobre o capítulo	70
4	Resultados e discussões	73
4.1	Preparação dos experimentos	73
4.1.1	Implementação e ambiente de teste	73
4.1.2	O <i>Benchmark</i>	74

4.1.3	Metodologia de teste validação dos resultados	77
4.1.3.1	Procedimentos para avaliação da qualidade	77
4.1.3.2	Procedimentos para avaliação do comportamento do <i>bloat</i>	78
4.2	Acréscimo de subárvores aleatórias ao modelo	79
4.2.1	Configuração dos experimentos	79
4.2.2	Treinamento	80
4.2.3	Teste das melhores respostas	86
4.2.4	Tamanho das melhores respostas	91
4.2.5	Efeito <i>bloat</i>	96
4.3	Comparação com operador sintático	102
4.3.1	Configuração dos experimentos	103
4.3.2	Treinamento	104
4.3.3	Teste das melhores respostas	110
4.3.4	Tamanho das melhores respostas	115
4.3.5	Efeito <i>bloat</i>	120
4.4	Comparação com operadores semânticos	126
4.4.1	Configuração dos experimentos	126
4.4.2	Treinamento	128
4.4.3	Teste das melhores respostas	131
4.4.4	Tamanho das melhores respostas	132
4.4.5	O efeito <i>bloat</i>	135
4.5	Comparação com a MRGP	138
4.5.1	Configuração dos experimentos	138
4.5.2	Teste das melhores respostas	139
4.5.3	Tamanho das melhores respostas	140
4.6	Programação <i>Kaizen</i>	142
4.6.1	Configuração do experimento	142
4.6.2	Resultados	143
4.7	Considerações sobre o capítulo	146

5	Conclusão	149
5.1	Considerações finais	149
5.2	Trabalhos futuros	150

Referências	153
------------------------------	------------

Apêndices	163
------------------	------------

APÊNDICE A	Principais técnicas de combate ao efeito <i>bloat</i>	165
A.1	Técnicas baseada na avaliação	165
A.2	Técnicas baseadas na seleção	166
A.3	Técnicas baseadas em recombinações especializadas	167
A.4	Técnicas baseadas em sobrevivência	168

Introdução

1.1 Visão geral

A descrição automática do comportamento de sistemas, sejam eles mecânicos, biológicos ou sociais, a partir da análise de uma massa de dados organizada como conjuntos de entrada/saída é área de grande interesse nas Ciências e nas Engenharias. É comum a utilização de Redes Neurais Artificiais na atividade de mapear modelos. Entretanto, caso o conhecimento das equações matemáticas que compõem o modelo seja, por alguma razão, de interesse do pesquisador, faz-se necessário adotar algum método de regressão.

Existem basicamente duas abordagens de regressão: 1) a clássica, fundamentada em conceitos estatísticos como a técnica dos mínimos quadrados; 2) a regressão simbólica, que pode ser obtida através da Computação Evolucionária. A regressão clássica consiste em adaptar um modelo matemático pré-estabelecido e ajustar parâmetros de forma a obter uma expressão que descreva adequadamente os dados observados. A função (1.1), por exemplo, pode ter seus parâmetros β_0 e β_1 ajustados de forma a descrever um conjunto de dados com uma entrada x e uma saída $f(x)$.

$$f(x) = \beta_0 + \beta_1 x \quad (1.1)$$

Se o ajuste de parâmetros não for perfeitamente adequado, o modelo baseado na função (1.1) torna-se sem utilidade. Além disso, um problema de modelagem pode ser suficientemente difícil a ponto de tornar árdua a tarefa de encontrar o tipo de função que seja melhor parametrizável, algo que não ocorre quando se aplica a regressão simbólica.

A regressão simbólica realiza a busca de uma estrutura de modelo apropriada aos dados ao invés de impor qualquer suposição anterior. Isso pode ser feito por meio de uma técnica de Computação Evolucionária conhecida como Programação Genética, que cria populações de equações e as fazem evoluir, seguindo em direção à equação mais apta à descrição do modelo.

A Programação Genética se destaca por possibilitar a resolução automática de problemas a partir de declarações de alto nível sobre o que precisa ser feito, sem exigir do usuário informações sobre forma e estrutura da solução (POLI; LANGDON; MCPHEE, 2008). Embora seja uma extensão dos Algoritmos Genéticos, a Programação Genética diferencia-se pela capacidade de manipular estruturas complexas, como listas, pilhas, árvores ou grafos em geral.

Existem diferentes abordagens para a utilização da Programação Genética. Estratégias tradicionais são guiadas pelas características sintáticas da população, criando novos indivíduos a partir da recombinação estrutural. Técnicas recentes utilizam conceitos semânticos, ou seja, o significado subjacente a cada indivíduo, para melhorar a capacidade de convergência. Outras técnicas utilizam propriedades estatísticas para fazer ajustes finos nas respostas obtidas. O que une todas essas abordagens, porém é a necessidade de lidar com a tendência de crescimento do tamanho médio dos indivíduos da população.

Desde as suas primeiras implementações, pesquisadores notam que, após algumas gerações, o número de nós que compõem as árvores de uma população começa a crescer aceleradamente e sem que esse crescimento apresente melhorias na aptidão. Este fenômeno, denominado efeito *bloat*, é ainda hoje objeto de investigação, pois não há consenso sobre suas origens nem tampouco técnica definitiva que o resolva.

Populações com árvores grandes são mais difíceis de evoluir, dada a alta demanda por poder computacional. Talvez as plataformas de computação de alto desempenho hoje disponíveis tenham a capacidade de atender às demandas de processamento adicionais. Entretanto, indivíduos extensos podem apresentar baixa capacidade de generalização e serem complexos demais para aplicações práticas, já que tornam-se difíceis de interpretar.

Nesse contexto, propomos neste trabalho uma técnica que tende a manter o tamanho das árvores em níveis controlados ao mesmo tempo que propicia maior capacidade de convergência na população.

1.2 Motivação

A motivação inicial para a pesquisa apresentada nesta tese teve como foco desenvolver uma nova forma de conter o efeito *bloat* e aumentar a capacidade de convergência da Programação Genética. Essa primeira abordagem baseou-se em características sintáticas das árvores. Nessa fase da pesquisa foram realizadas diversas tentativas de interpretação das características estruturais das árvores que compõem a população, com o objetivo de identificar quais seguimentos (ou sub-expressões ou subárvore) eram mais significativas em um indivíduo. Essa informação, acreditava-se, poderia ser útil para a criação de operadores genéticos que realizassem a recombinação de maneira mais eficiente. Infelizmente, poucos avanços foram conseguidos nesta fase.

O contato com técnicas Computação Evolucionária que utilizam Análise de Regressão forneceu novas possibilidades para a pesquisa. Dentre as abordagens, as que mais se destacaram foram a MRGP (ARNALDO; KRAWIEC; O'REILLY, 2014) e a Programação *Kaizen* (De Melo, 2014). Com o estudo dessas técnicas foi possível redirecionar o foco da pesquisa para o desenvolvimento de um novo operador genético: ao invés de gerar novos indivíduos formados pelas melhores subárvores dos pais, o operador recombina subárvores ponderadas por constantes reais cuidadosamente escolhidas, de forma a obter a melhor estrutura possível a partir das subestruturas aleatoriamente selecionadas.

1.3 Objetivos

O principal objetivo deste trabalho é apresentar um novo operador para Programação Genética que possibilite evoluir populações que possam apresentar indivíduos de maior acurácia e de representação estrutural cujo tamanho seja naturalmente controlado.

1.3.1 Objetivos específicos

No contexto da Programação Genética, buscamos nesta tese alcançar os seguintes objetivos específicos:

- Levantamento das principais definições necessárias ao entendimento da Programação Genética;
- Levantamento das principais teorias que explicam o efeito *bloat*;
- Entender a relação existente entre o efeito *bloat* e os operadores genéticos tradicionalmente utilizados;
- Levantamento o estado da arte dos operadores sintáticos e semânticos existentes;
- Conhecer a abordagem que as principais técnicas de Computação Evolucionária baseadas em Análise de Regressão utilizam para obter seus coeficientes e quais são suas limitações;
- Propor um operador para a Programação Genética baseado em regressão;
- Com base em métricas de qualidade baseadas nos erros e nos comprimentos observados nas melhores respostas obtidas no treinamento, na capacidade de generalização dessas respostas e no efeito *bloat* observado na população, comparar os resultados gerados pelo operador proposto com os resultados obtidos com a utilização de outros operadores genéticos e com outras técnicas de Computação Evolucionária baseadas em Análise de Regressão.

1.4 Contribuições

O presente trabalho trouxe as seguintes contribuições:

- A criação de novo operador genético capaz de gerar indivíduos de maior acurácia e com controle de *bloat* que atua naturalmente;
- Aplicação da matriz pseudo inversa em problemas de regressão linear múltipla relacionados à Programação Genética;
- Desenvolvimento de técnica de autoajuste da taxa de mutação que proporciona diversidade populacional e consequente melhoria no desempenho global da evolução;

1.5 Organização do trabalho

Este trabalho é organizado em cinco capítulos. O Capítulo 2 contempla uma revisão bibliográfica das principais teorias que embasam esta tese. O funcionamento básico da Programação Genética é descrito, assim como as teorias referentes às origens do efeito *bloat* que estão relacionadas com a aplicação dos operadores genéticos. Abordagens de Programação Genética baseadas em Análise de Regressão são também apresentadas nesse capítulo, juntamente com os fundamentos estatísticos necessários, incluindo a definição de matriz pseudo inversa.

O Capítulo 3 apresenta o operador genético MRX, responsável por criar novos indivíduos a partir da recombinação de subárvores ajustadas por meio de constantes reais obtidas via regressão linear múltipla. Além de descrever o funcionamento seu básico, são apresentadas também técnicas de melhoria na diversidade populacional que fazem do MRX um operador híbrido, unindo recombinação e mutação. O capítulo termina confrontando as teorias de origem do *bloat* diretamente ligadas aos operadores genéticos ao operador MRX, apontando indícios teóricos que o fazem resistente ao problema.

O Capítulo 4 apresenta uma série de experimentos realizados para comparar o desempenho do operador criado nesta tese com algumas abordagens existentes. São descritos os resultados de comparações com a Programação Genética clássica, com operadores semânticos recentes e com outras abordagens de Computação Evolucionária baseadas em Análise de Regressão. Analisam-se a qualidade das respostas obtidas por todos os métodos e a forma como *bloat* se comporta em cada um.

O Capítulo 5 discute as contribuições obtidas por meio das técnicas desenvolvidas nesta tese. São apresentadas também possíveis caminhos para se conseguir melhorias nas técnicas apresentadas por meio da escolha de novos parâmetros de execução, que pode ser feito em trabalhos futuros.

Aspectos teóricos e revisão bibliográfica

A Programação Genética é uma técnica de Computação Evolucionária com notável potencial de aplicações, principalmente devido à sua capacidade de evoluir estruturas complexas. Existem diferentes formas implementá-la, podendo-se basear em questões como sintaxe, semântica e características estatísticas inerentes. Este capítulo apresenta uma discussão sobre os principais conceitos de Programação Genética, descrevendo seu funcionamento, algumas variações e discutindo um dos seus principais problemas: o efeito *bloat*.

2.1 Computação Evolucionária

Computação Evolucionária (CE) é o termo utilizado para descrever sistemas artificiais inspirados na Teoria da Evolução de Charles Darwin. Ao longo das últimas décadas, pesquisadores propuseram diferentes técnicas de CE, das quais podemos citar a Busca Evolutiva (TURING, 1950), a Programação Evolutiva (FOGEL, 1962), os Algoritmos Genéticos (HOLLAND, 1975) e a Programação Genética (KOZA, 1992a). Embora cada técnica seja pautada por particularidades, De Jong (2006) chama a atenção para o fato de todas compartilharem pelo menos quatro características:

- A existência de uma população (ou de várias), formada por indivíduos que competem por recursos limitados;
- O comportamento dinâmico das populações, que são constantemente modificadas por eventos como nascimento e morte de indivíduos;
- Alguma medida de aptidão (*fitness*) associada à capacidade de um indivíduo sobreviver e/ou deixar descendentes;
- A herança variacional, ou seja, a prole herda características que a faz semelhante a seus pais, mas não idêntica a eles.

É comum em CE o emprego de termos próprios das Ciências Biológicas. Os mais comuns foram organizados e definidos por Peretta (2015):

Indivíduo: Uma possível solução para o problema em análise.

Cromossomo (ou genótipo): Representação do indivíduo dentro do espaço de busca. Esta representação é específica do problema em análise e pode corresponder a uma *string*, a um vetor ou a alguma estrutura de dados mais complexa, como grafos.

Genes: Elementos básicos que compõem o cromossomo. Se o cromossomo é um vetor \mathbf{v} de números inteiros, por exemplo, então o elemento da posição \mathbf{v}_i é um gene.

Fenótipo: Significado prático de um cromossomo. Indivíduos com diferentes genótipos podem apresentar mesmo fenótipo. Por exemplo, em uma população de equações matemáticas, diferentes equações podem, quando simplificadas, representar a mesma equação.

População: Conjunto de todos os indivíduos. Tradicionalmente, a população apresenta tamanho constante.

Meio ambiente: Espaço de busca ou domínio do problema em análise.

Aptidão (ou *fitness*): Métrica que indica o quanto um indivíduo é adequado à resolução do problema em foco.

Função de avaliação ou de adequação: Procedimento utilizado para calcular a aptidão do indivíduo.

Recombinação ou cruzamento: Operador genético que combina aleatoriamente os cromossomos de dois ou mais indivíduos (pais) para produzir um, dois ou mais indivíduos (filhos). Este operador deve ser capaz de promover a herança variacional.

Mutação: Operador genético que provoca modificações aleatórias nos cromossomos dos indivíduos. Este operador auxilia no aumento da diversidade populacional.

Seleção: Mecanismo que determina quais indivíduos serão escolhidos para serem submetidos aos operadores genéticos. A forma mais comum de seleção é a baseada na aptidão, mas a seleção randômica pode também ser empregada.

Geração: É um ciclo completo de execução da Computação Evolucionária. Esse ciclo consiste em selecionar indivíduos da população, submetê-los aos operadores genéticos, avaliá-los segundo algum critério estabelecido a priori e definir quais indivíduos serão utilizados para compor a população da próxima geração, caso nenhuma solução aceitável seja encontrada na geração atual.

Redução: Consiste na “morte” do(s) indivíduo(s) menos apto(s). Pode ser utilizada para manter a população com quantidade constante de indivíduos e/ou impedir que soluções inválidas sejam disseminadas e comprometam soluções candidatas.

Essas definições são de grande utilidade para a compreensão deste trabalho. Na seção seguinte, vários desses termos são utilizados para melhor definir a Programação Genética.

2.2 Programação Genética

A *Programação Genética* (PG) é uma técnica de Inteligência Artificial e Aprendizado de Máquina capaz de evoluir estruturas como equações matemáticas ou programas de computador a partir de declarações de alto nível sobre o que precisa ser feito, sem, contudo, exigir do usuário informações adicionais sobre o formato da solução (POLI; LANGDON; MCPHEE, 2008). Isso só é possível porque a PG mapeia os genótipos de seus indivíduos no formato de estruturas de dados complexas de tamanho e formato variáveis, como listas, árvores ou grafos mais gerais. Esta seção descreve os elementos mais essenciais à utilização de PG.

2.2.1 Cromossomos

Em PG, um cromossomo deve ser capaz de se manifestar sob diferentes formatos e tamanhos. Assim, estruturas de dados como listas encadeadas podem ser candidatas adequadas à utilização em PG¹. Apesar disso, são as árvores as estruturas de dados com maior utilização em Programação Genética.

A preferência pela utilização de árvores pode ser explicada pelo uso de linguagens funcionais desde as primeiras implementações de PG realizadas por Koza (1992a). Em LISP, por exemplo, programas podem ser mapeados diretamente em forma de árvore – uma árvore sintática. Dessa forma, o trecho `(* (sin (- (y x)) (if (> x 5.1) (+ 2.1 x) (* (sqrt y) (/ x 3.5))))` pode ser diretamente mapeado para a árvore da Figura 1.

Além de programas, expressões matemáticas podem também ser representadas na forma de árvores (ou na sintaxe LISP). Para tanto, basta convertê-las para a notação prefixa (GRAHAM, 1996), como exemplificado na Figura 2. Esta notação traz facilidades à compreensão do relacionamento de uma sub-expressão e de uma subárvore correspondente (POLI; LANGDON; MCPHEE, 2008).

Os nós de uma árvore correspondem aos genes do cromossomo em Programação Genética. Estes nós são divididos em dois grupos:

¹ E, de fato, são. Banzhaf (1993), por exemplo, utilizou listas para representar as sequências de comandos que forma um programa.

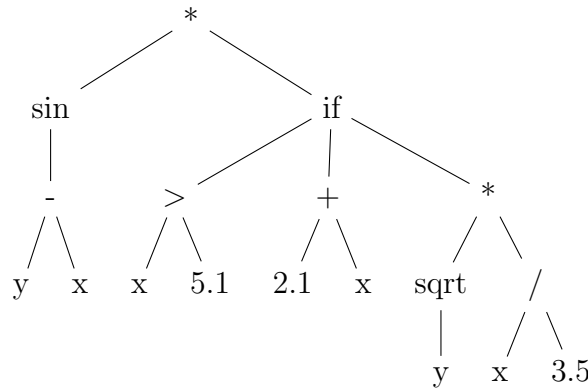


Figura 1 – Árvore sintática para código em LISP.

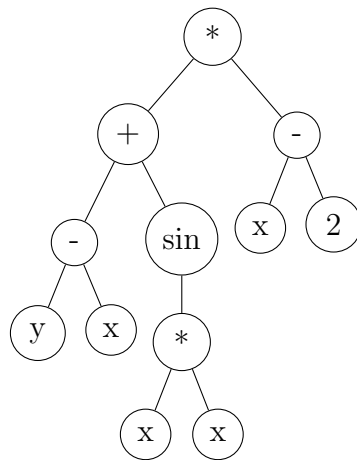


Figura 2 – Árvore da expressão matemática $((y - x) + \sin(x * x)) * (x - 2)$. Ao reescrevê-la na notação pré-fixa $(* (+ (- y x) (\sin (* x x))) (- x 2))$, a hierarquia associada a seus nós torna-se evidente.

Nós não terminais ou funções: São os nós internos à árvore. No caso de expressões matemáticas, correspondem às funções com argumento. Ao número de argumentos de uma função, dá-se nome de *aridade*. A Tabela 1 exemplifica algumas funções comumente usadas em árvores de PG.

Nós terminais: São os nós extremos das árvores (folhas), podendo representar variáveis, constantes ou funções sem argumento. Exemplos de terminais são apresentados na Tabela 2.

Cada nó função em uma árvore utilizada em PG deve, garantidamente, ser capaz de aceitar e tratar qualquer outro nó como argumento, seja ele uma constante, uma variável ou outro nó função. Esta propriedade recebe o nome de *fechamento* (KOZA, 1992b). Para garantir o fechamento em todas as situações a evitar que o processamento seja interrompido pela ocorrência de algum erro (como uma divisão por zero ou o cálculo de uma raiz quadrada no domínio real com argumento negativo, por exemplo), costuma-se aplicar modificações pontuais aos operadores:

Tabela 1 – Exemplos de funções comumente utilizadas em PG.

Tipos de Primitivas	Exemplos
Aritméticas	$+$, $*$, $-$, $/$
Matemáticas	\sin , \cos , \tan , etc.
Booleanas	AND, OR, NOT
Condicional	if-then-else
Laços	for, repeat

Fonte: Poli, Langdon e McPhee (2008).

Tabela 2 – Exemplos de terminais comumente utilizadas em PG.

Tipos de Primitivas	Exemplos
Variáveis	x , y
Valores constantes	3.14, 100
Funções sem argumento	rand

Fonte: Poli, Langdon e McPhee (2008).

- Uma divisão por zero pode ser evitada implementando a *divisão protegida*, tradicionalmente denotada por $\%$. Este operador recebe como argumento um numerador e um denominador (num , den), retornando 1.0 sempre que $den = 0$ e num/den caso contrário.
- Um número *negativo* no argumento de uma função que calcula a *raiz quadrada* pode ser tratado calculando o *valor absoluto* do argumento antes de aplicá-lo à função. O mesmo vale para a função *logaritmo*, com a ressalva de ela retornar zero sempre que o argumento for zero.
- Se um valor *numérico* for aplicado em uma função que espera receber um valor *lógico*, pode-se adotar a convenção de considerar **false** qualquer valor negativo ou **true**, caso contrário. Essa escolha deve ser feita com cuidado, já que algumas linguagens de programação consideram **false** quando o valor numérico é 0 e **true** para qualquer outro valor.

Existem situações em que operações protegidas não são capazes de garantir resultados válidos. Exemplo é a função **exp(x)** (e^x , exponencial), cujo domínio é o conjunto \mathbb{R} , dos números reais. A expressão **exp(exp(exp(exp(exp(x)))))**, embora sintática e semanticamente válida, corresponde a valores tão elevados que a representação de sua avaliação gera erros na maioria das linguagens de programação.

Por fim, o valor de saída de uma árvore é obtido por meio do *caminho em pré-ordem* (POLI; LANGDON; MCPHEE, 2008), que pode ser naturalmente implementado por meio de chamadas recursivas. Este procedimento, detalhando no Algoritmo 1, é a forma mais

eficiente de realizar a avaliação, visto que possui tempo de execução $O(n)$, onde n é o número de nós da árvore (GOODRICH; TAMASSIA, 2003).

Algoritmo 1 `avalia(expr)`

```

1: se expr é uma função então
2:    $\text{proc} \leftarrow \text{expr}(1)$    (O primeiro elemento equivale à raiz)
3:    $\text{valor} \leftarrow \text{proc}(\text{avalia}(\text{expr}(2)), \text{avalia}(\text{expr}(3)), \dots)$    (Avalia seus argumentos)
4: senão
5:   se expr é uma variável or expr é uma constante então
6:      $\text{valor} \leftarrow \text{expr}$    (Obtém seu valor)
7:   senão
8:      $\text{valor} \leftarrow \text{expr}()$    (Função sem-argumentos: execute)
9:   fim se
10: fim se
11: retorne valor

```

2.2.2 Geração da população inicial

A criação de um indivíduo inicia-se com a escolha aleatória de uma função que representará a raiz da árvore. Seus argumentos são aleatoriamente selecionados dentro dos conjuntos de nós (terminais e não terminais) disponíveis e recursivamente inseridos. Este processo é exemplificado na Figura 3.

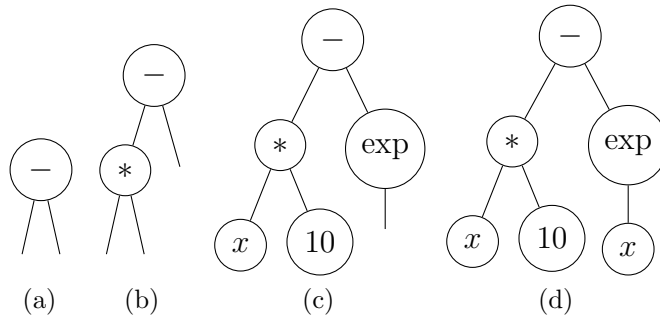


Figura 3 – Criação de um árvore: (a) escolha da função $-$, com dois argumentos, como raiz; (b) escolha da função $*$, com dois argumentos, para ser o primeiro argumento de $-$; (c) escolha dos terminais x e 10 para argumentos de $*$ e a função `exp`, com um argumento, para segundo argumento de $-$; (d) escolha do terminal x para argumento de `exp`.

Os métodos mais comuns de criação de árvores em PG são:

Full: Gera todas as árvores com profundidade d . Durante a criação, funções são escolhidas até uma profundidade $d - 1$. Neste momento, um terminal é selecionado (Figura 4a).

Grow: Gera árvores com profundidade *máxima* d . Após a escolha do nó raiz (uma função), escolhe-se aleatoriamente entre funções e terminais, de forma a não ultrapassar a profundidade d (Figura 4b).

Ramped half-and-half: Uma combinação dos métodos *full* e *grow*.

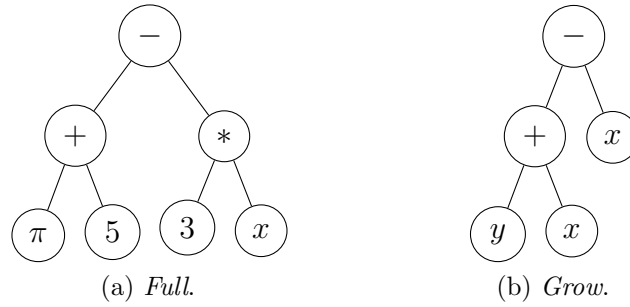


Figura 4 – Exemplos de árvores geradas pelos métodos *full* e *grow*, com profundidade $d = 3$.

Os métodos *full* e *grow* são descritos no Algoritmo 2, adaptado de Poli, Langdon e McPhee (2008), onde **funcoes_set** é o conjunto de funções, **terminais_set** é o conjunto de terminais, **max_d** é a profundidade máxima definida para a árvore em criação e **rand()** é uma função que retorna um número real randômico uniformemente distribuído entre 0 e 1.

Algoritmo 2 **gen_rnd_expr**(funcoes_set, terminais_set, max_d, metodo)

```

1: se max_d = 0 ou (metodo = grow e rand() <  $\frac{|\text{terminais\_set}|}{|\text{terminais\_set}| + |\text{funcoes\_set}|}$ ) então
2:   expr ← escolha_elemento_randomicamente(terminais_set)
3: senão
4:   func ← escolha_elemento_randomicamente(funcoes_set)
5:   para i ← 1 até aridade(func) faça
6:     arg_i ← gen_rnd_expr(funcoes_set, terminais_set, max_d - 1, metodo)
7:   fim para
8:   expr ← (func, arg_1, arg_2, ...)
9: fim se
10: retorne expr

```

2.2.3 O cálculo da aptidão

Em Programação Genética, a aptidão é calculada com base em casos de treinamento (ou conjunto de treinamento) que costumam ser organizados em arquivos específicos de dados denominados *datasets*. A Tabela 3 exemplifica um *dataset*, onde a primeira coluna representa a entrada x da função enquanto a segunda representa a saída $f(x)$. A manipulação de expressões matemáticas para descoberta de funções recebe o nome de *regressão simbólica*.

Ao substituir as variáveis em uma árvore por seus valores de entrada conforme o conjunto de treinamento e realizar sua avaliação, é possível comparar o quão distante está

Tabela 3 – *Dataset* Nguyen-6, definido pela função $f(x) = \sin(x) + \sin(x + x^2)$. Os valores de x são valores aleatórios uniformemente distribuídos dentro do intervalo $[-1, 1]$.

x	$f(x)$
-0.145476224769	-0.268956598372
0.286700923885	0.643377463608
0.788439715468	1.696366619411
0.903926012813	1.774500701538
0.947271488824	1.774576134040
-0.560386106268	-0.775382489529
0.889338306295	1.770669829983
0.413290613432	0.953073659382
0.341834109444	0.777984762076
-0.704394366239	-0.854294008024
0.772655120219	1.677877326528
-0.312736371232	-0.520944748362
-0.103689335315	-0.196307756783
0.404630803553	0.931927389799
-0.472375707538	-0.701667619584
-0.368915351794	-0.591323225850
0.615783149800	1.416341741364
-0.196850113951	-0.353023585378
-0.309735593887	-0.516981194422
0.749860288033	1.648273812511

a saída calculada da saída desejada. Este procedimento aplicado a todos os valores de entrada do *dataset* gera dados utilizados na determinação de alguma métrica de qualidade. Se n é o número de amostras do conjunto de treinamento no domínio de x , p_{ij} é o valor de saída determinado pela árvore i para a amostra j e s_j é o valor de $f(x)$ no conjunto de treinamento para a j -ésima amostra de x , podemos determinar as seguintes métricas:

Sum of Absolute Errors (SAE): A soma dos erros absolutos foi o método adotado por Koza (1992a) nas primeira implementações de PG é ainda bastante utilizada atualmente. Sua fórmula é descrita em (2.1):

$$SAE_i = \sum_{j=1}^n |p_{ij} - s_j| \quad (2.1)$$

Mean Absolute Error (MAE): Consiste do valor médio das diferenças absolutas, como definido em (2.2):

$$MAE_i = \frac{\sum_{j=1}^n |p_{ij} - s_j|}{n} \quad (2.2)$$

Mean Squared Error (MSE): A soma dos quadrados dos erros diferencia-se do MAE por utilizar o quadrado do erro ao invés do valor absoluto para obter resultados positivos, como apresentado em (2.3):

$$MSE_i = \frac{\sum_{j=1}^n (p_{ij} - s_j)^2}{n} \quad (2.3)$$

Root Mean Squared Error (RMSE): Fornece valores mais compactos do que o MSE e sua fórmula é dada em (2.4):

$$RMSE_i = \sqrt{\frac{\sum_{j=1}^n (p_{ij} - s_j)^2}{n}} \quad (2.4)$$

2.2.4 Seleção

Selecionar um indivíduo em CE consiste em aplicar alguma metodologia que permita uma escolha que seja ao mesmo tempo aleatória e enviesada segundo algum critério – geralmente associado à qualidade. Os métodos de seleção mais tradicionalmente utilizados são a seleção por aptidão proporcional (roleta viciada), *ranking* e torneio.

Na roleta viciada, indivíduos com maior aptidão têm maiores chances de serem selecionados. Nesse caso, a presença de um super indivíduo (algum elemento da população que possua aptidão desproporcionalmente superior quando comparado aos demais) faz com que a convergência ocorra prematuramente, diminuindo a diversidade populacional e dificultando a busca por soluções (LINDEN, 2008).

A seleção por *ranking* evita o problema do super indivíduo mantendo a pressão seletiva no mesmo nível em todas as gerações, independentemente do grau de convergência que a população já tenha alcançado (LINDEN, 2008). Esta técnica não utiliza diretamente o valor da função de aptidão: antes, é criado um *ranking* dos melhores indivíduos por meio do ordenamento crescente da população de acordo com os valores dados pela função de aptidão. O problema desta abordagem reside na necessidade de ordenar a população a cada geração, um procedimento que, em suas mais eficientes implementações, demanda tempo de execução $O(n \log n)$, sendo n o número de elementos a serem ordenados (CORMEN et al., 2001), ou seja, o tamanho da população.

Já o torneio é o método de seleção mais tradicionalmente utilizado em implementações de Programação Genética, desde a sua concepção por Koza (1992b). Consiste em escolher randomicamente k indivíduos e fazer com que eles compitam entre si. Vence o torneio aquele que possuir a melhor avaliação na função de aptidão (BLICKLE; THIELE, 1995).

Como todos os integrantes do grupo que compõe o torneio são escolhidos aleatoriamente com a mesma probabilidade, é possível que os melhores indivíduos (incluindo algum super indivíduo) não sejam escolhidos para competir. Assim, o torneio proporciona seleção com menor pressão seletiva do que o método proporcional sem a necessidade de ordenamento. O torneio é ainda capaz de emular todos os outros métodos de seleção, a depender apenas do número de competidores que participam da disputa (De Jong, 2006).

2.2.5 Operadores genéticos

Esta seção apresenta e discute os dois principais operadores genéticos utilizados em PG: a recombinação e a mutação. A discussão é feita considerando aspectos sintáticos e semânticos.

2.2.5.1 Abordagem sintática

Em PG, as primeiras implementações de recombinação baseavam-se simplesmente na estrutura (formato, ou sintaxe) das árvores envolvidas no processo. Um dos primeiros operadores criados foi denominado *recombinação de subárvore*.

Dadas duas árvores a e b , selecionadas por algum método descrito na Seção 2.2.4, escolhe-se um ponto (nó) em cada uma. Em a , descarta-se a subárvore cuja raiz é o nó selecionado, substituindo-a pela subárvore cuja raiz é o nó selecionado em b . Os nós de b não pertencentes à subárvore selecionada são descartados. Este processo pode ser visto na Figura 5.

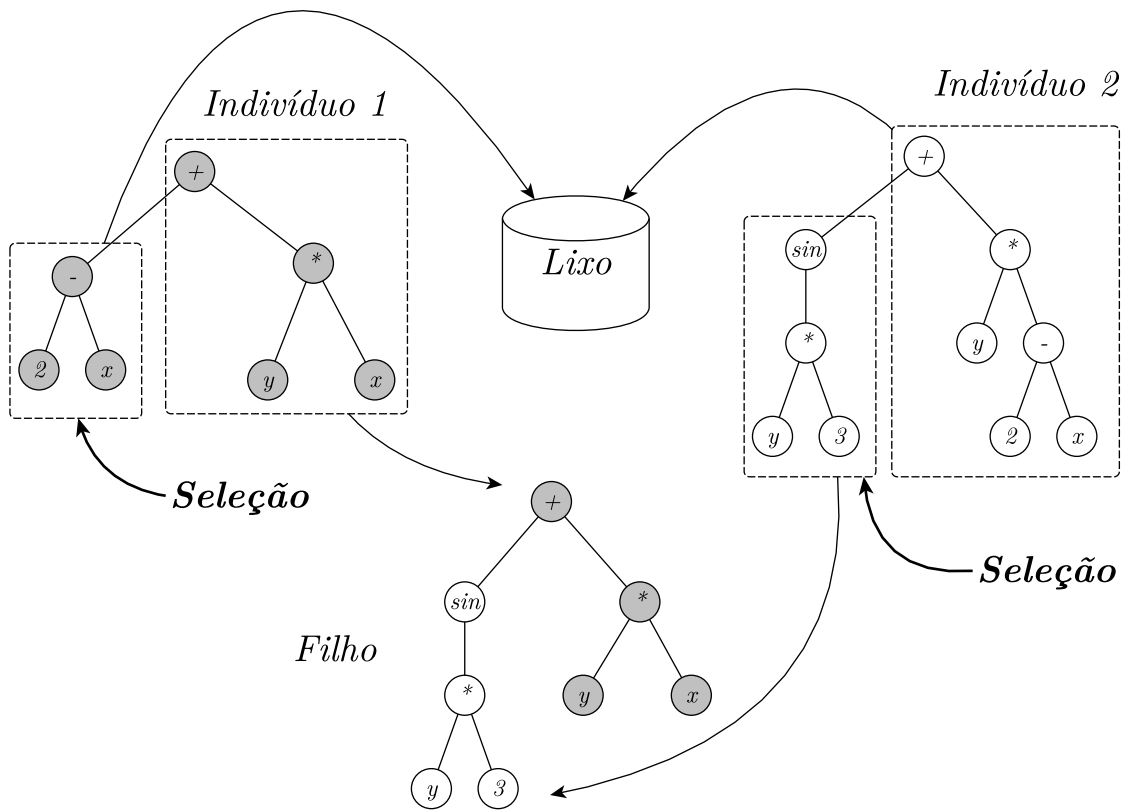


Figura 5 – Recombinação entre os indivíduos $((2-x) + (y*x))$ e $((\sin(y*3)) + (y*(2-x)))$, que resultou no filho $((\sin(y*3)) + (y*x))$.

Há em Poli, Langdon e McPhee (2008) a recomendação de que as partes do indivíduo envolvidas com o cruzamento sejam efetivamente copiadas, para não causar qualquer dano aos indivíduos originais. De fato, embora a realização da cópia possa causar sobrecarga no processamento computacional, sua utilização previne muitos erros comuns à manipulação de ponteiros e referências.

Embora qualquer nó possa ser selecionado como ponto de recombinação, normalmente adota-se a estratégia de escolher nós função em 90% das ocasiões, sendo os outros 10% reservados para nós terminais.

Outro operador utilizado em PG é a mutação, cuja primeira versão foi proposta por Koza (1992b) e é conhecida por *mutação de subárvore*. Consiste em selecionar aleatoriamente um nó da árvore e substituí-lo por uma subárvore criada também de forma aleatória. Uma versão semelhante foi proposta por Kinnear, Jr. (1994), que impôs uma restrição: o tamanho da nova árvore gerada não poderia ser 15% maior do que a original.

Embora Koza (1992b) tenha apresentado resultados que classificam a mutação como um operador secundário e desnecessário, por desempenhar pouca influência na evolução de funções booleanas, resultados descritos em O'Reilly e Oppacher (1994), Chellapilla (1997), Harries e Smith (1997) e Luke e Spector (1997) indicaram o contrário, alegando que a mutação, principalmente quando combinada com outros operadores, pode trazer benefícios para problemas específicos de PG.

2.2.5.2 Abordagem semântica

Em computação, o termo *semântica* pode ser entendido como o significado de programas (ou funções) sintaticamente corretos, sendo que dois programas sintaticamente iguais possuem a mesma semântica, embora não se possa garantir que dois programas que possuam a mesma semântica representam também a mesma sintaxe (UY et al., 2011).

Para a criação de operadores genéticos especializados, é necessário definir algum método para identificar a semântica dos indivíduos. Um dos primeiros esforços nesse sentido foi o SDC (*Semantically Driven Crossover*), proposto por Beadle e Johnson (2008). O SDC melhora a diversidade comportamental dos indivíduos da população impedindo que filhos semanticamente equivalentes aos pais sejam produzidos. A semântica de indivíduos desse domínio pode ser facilmente obtida pela utilização de diagramas de decisão binária ordenada reduzida ou ROBDD, do inglês *Reduced Ordered Binary Decision Diagram*.

Assim, determinar a semântica de funções booleanas consiste em enumerar todos os seus possíveis valores de saída. Para funções reais, porém, essa tarefa se torna mais complexa. Nguyen et al. (2016) propuseram uma forma de avaliar o significado de funções deste tipo: se $X = (x_1, x_2, \dots, x_n)$ são as entradas do conjunto de treinamento, então para um dado programa P a semântica do programa, denotado por $S(P)$, é o vetor

$$S(P) = (P(x_1), P(x_2), \dots, P(x_n)), \quad \text{para } i = 1, 2, \dots, n \quad (2.5)$$

Dessa forma, é possível definir também a *distância semântica* (DS) entre dois programas (P_1 e P_2) como:

$$DS(P_1, P_2) = \frac{|S(P_1) - S(P_2)|}{n} = \frac{\sum_{i=1}^n |P_1(x_i) - P_2(x_i)|}{n} \quad (2.6)$$

Operadores como o SDC analisam o quanto um filho é distante, em termos de significados (valores) dos pais, podendo aceitar ou não a sobrevivência do novo indivíduo na

nova população. Assim, a semântica não está diretamente ligada ao operador, fazendo dele um método indireto. A primeira versão de PG com operadores que tratam de semântica de forma direta, proposta por Moraglio, Krawiec e Johnson (2011), é denominada *Geometric Semantic Genetic Programming* (GSGP). Esta técnica propõe o operador de recombinação SGX (*Semantic Crossover*) e um operador de mutação SGM.

Para o caso de funções de domínio real, o SGX funciona combinando duas árvores A_1 e A_2 para criar uma árvore filha A_3 , tal que $A_3 = (A_1 \times A_R) + ((1 - A_R) \times A_2)$, sendo A_R uma constante real no intervalo $[0, 1]$ ou uma função real cuja imagem é o intervalo $[0, 1]$. O operador de mutação da GSGP atua sobre uma árvore A modificando-a para A_M , tal que $A_M = A + ms \times (A_{R_1} - A_{R_2})$, onde A_{R_1} e A_{R_2} são funções reais aleatórias com imagem em $[0, 1]$ e ms é o passo de mutação.

A GSGP produz filhos que contém em sua composição todos os genes dos pais, o que leva ao crescimento exponencial da população. Implementações eficientes, como em Vanneschi et al. (2013), foram propostos para amenizar a sobrecarga da memória durante o processamento, mas as respostas produzidas pela GSGP ainda são, em geral, grandes e complexas demais para serem entendidas por seres humanos.

Operadores AGX e RDO

Um operador que tenta reduzir o crescimento exponencial das árvores evoluídas pelo SGX foi proposto por Beadle e Johnson (2008) e denominado *Semantic Aware Crossover* (AGX). Após escolher os pontos de cruzamento nos pais A_1 e A_2 , o AGX seleciona uma árvore, dentro de uma coleção pré-estabelecida, uma cuja semântica seja intermediária às duas subárvores dos pais.

A forma como AGX determina a semântica de suas subárvores consiste em armazenar todos os valores intermediários obtidos na avaliação, entrada por entrada. Assim, cada nó possui um vetor com saídas locais. Técnica similar é aplicada ao *Random Desired Operator* (RDO), um operador proposto por Pawlak, Wieloch e Krawiec (2015) que diferencia-se do AGX por agir sob um único indivíduo, substituindo uma subárvore por outra (selecionada de um repositório) que possua semântica situada em um ponto entre a árvore em operação e a saída desejada do conjunto de treinamento.

Operador SSGX

Outro operador semântico recentemente criado é o *Subtree Semantic Geometric Crossover* (SSGX), proposto por Nguyen et al. (2016). Ele se assemelha ao SGX, com a diferença de agir no nível das subárvores. Dados dois pais A_1 e A_2 , seleciona-se um ponto de recombinação para cada pai: P_1 e P_2 , que são na verdade raízes de subárvores. É escolhida de um repositório uma subárvore S_{A_1} que possui similaridade com P_1 e que obedece uma restrição de tamanho. O mesmo procedimento é feito P_2 , escolhendo uma subárvore S_{A_2} .

Por fim, dois filhos são criados: $A_R \times S_{A_1} + (1 - A_R) \times S_{A_2}$ e $(1 - A_R) \times S_{A_1} + A_R \times S_{A_2}$, onde A_R é uma função aleatória com imagem em $[0, 1]$.

Durante a evolução, o SSGX não é utilizado em todas as ocasiões. Nguyen et al. (2016) relatam que a aplicação do operador em 100% dos casos faz com que a população rapidamente colapse para indivíduos muito pequenos. Por essa razão, o SSGX é utilizado com probabilidade de 30%, sendo os outros 70% por conta da recombinação sintática de subárvore.

2.3 O efeito *bloat*

Desde as primeiras implementações de PG, pesquisadores notaram a natural tendência ao crescimento do tamanho médio das árvores que formam a população ao longo das gerações. O crescimento, a partir de certo ponto, não costuma ser acompanhado de melhorias significativas nos valores de aptidão dos indivíduos, que muitas vezes mantém-se praticamente estacionário. Esse fenômeno é denominado efeito *bloat*.

O *bloat* caracteriza-se pela presença de trechos de código desnecessários e/ou redundantes que podem ser suprimidos do programa sem afetar a semântica. Angeline (1994) chamou esses trechos de *introns*, em analogia ao que ocorre em sistemas biológicos. Os *introns* costumam ser divididos em duas categorias: código inviável, cuja remoção não modifica o valor de saída, e código não otimizado, que contribui na avaliação da árvore mas que poderia ser substituído por construções simplificadas e de mesmos significados (BRAMEIER; BANZHAF, 2003).

Uma das primeiras teorias acerca do aparecimento de *introns* em populações evoluídas por PG remete à Hipótese dos Blocos Construtivos. Tal teoria, conhecida por *Hitchhiking*, foi proposta por Tackett (1994) e sugere que a proliferação de *introns* ocorre porque estes são adjacentes a blocos construtivos de alta qualidade. Ou seja, os blocos construtivos correspondentes aos *introns* simplesmente “pegam carona” com as melhores subexpressões.

Para que a Programação Genética seja capaz de obter soluções simples e compactas, é necessário manter o controle sobre o tamanho médio dos indivíduos que compõem a população. Para tanto, é desejável que o sistema naturalmente direcione a busca para regiões do espaço onde se encontram as árvores mais compactas e mais aptas. Além de aumentar a clareza das respostas obtidas, o tratamento do *bloat* também evita problemas que podem ocorrer em tempo de execução, uma vez que evoluir árvores de grandes dimensões demanda mais memória física e maior poder de processamento (LUKE; PANAIT, 2006).

Não há consenso entre os pesquisadores com relação à fundamentação teórica do aparecimento do *bloat*. Entretanto, esse efeito parece ter suas origens intrinsecamente

relacionada à própria forma de operação da PG. Assim, são elencadas a seguir teorias que relacionam conceitos básicos de PG, como recombinação e seleção, ao crescimento descontrolado do número de genes nos indivíduos.

2.3.1 O *bloat* na recombinação

Esta seção detalha as teorias que associam o *bloat* o operador de recombinação: a *defesa contra a recombinação*, o *viés da remoção* e a *profundidade do ponto de cruzamento*.

2.3.1.1 Defesa contra a recombinação

A forma como o operador de recombinação sintática age faz com que ele seja potencialmente destrutivo, com a possibilidade de quebrar seguimentos de código de alta qualidade e gerar filhos potencialmente piores que os pais (NORDIN; FRANCONI; BANZHAF, 1995). Esse esse efeito é mais intensamente sentido nos estágios avançados da evolução.

Em Altenberg (1994) é apresentado um estudo sobre a evolucionabilidade² aplicada à Programação Genética que atenta para o fato de que nos estágios iniciais da evolução de uma população um bloco de código sobrevive por meio da capacidade de aumentar sua adequação através da recombinação com outros blocos de código na mesma medida que a média da aptidão. Entretanto, o autor observou que nos últimos estágios da evolução a população chega a um estado quase estacionário de aptidão média. Neste momento, a recombinação deixa de ser o principal fator de melhoria da aptidão e a vantagem passa a ser dos indivíduos que possuem aptidão pelo menos tão boa quanto a dos pais, ou seja, aqueles que foram criados por variações neutras.

Uma variação neutra é criada quando a recombinação insere *introns* nas árvores filhas, de forma a não modificar o código efetivo já existente. Isso parece explicar o fato de a aptidão média da população permanecer praticamente inalterada ao mesmo tempo em que o tamanho médio dos indivíduos aumenta exponencialmente. Assim, esse comportamento, denominado “defesa contra a recombinação”, figura entre as teorias candidatas a explicar o efeito *bloat*.

A defesa contra a recombinação foi também examinada por Blickle e Thiele (1994), que fizeram uma descrição matemática do efeito da redundância de seguimentos de código em PG. McPhee e Miller (1995) apresentaram também análises teóricas e experimentais cujos resultados indicaram que, para certos casos específicos, a pressão para replicação induz o aumento no tamanho das árvores geradas. Já em Nordin, Banzhaf e Francone (1999), foi desenvolvido o operador recombinação homóloga para diminuir o crescimento dos indivíduos de uma população ao evoluir códigos de máquinas para arquitetura CISC.

² Capacidade de uma população produzir variantes melhores do que qualquer outra que já existe.

Este operador efetua trocas de blocos sempre nas mesmas posições relativas em ambas as árvores e mostrou-se menos destrutivo. Outros trabalhos, como Smith e Harries (1998) e Soule (1998), por exemplo, indicam a efetividade da defesa contra a recombinação como causa do *bloat*, tendo sido, por essa razão, a teoria mais amplamente aceita por muitos anos.

Porém, resultados posteriores colocaram essa teoria em dúvida. Luke (2003) realizou experimentos em que os pontos de cruzamento ocorriam somente dentro das regiões consideradas inviáveis, causando diminuição dessas regiões. Notou-se porém que trechos de código não otimizado levam também ao crescimento das árvores. Contatou-se que a teoria de defesa parece apresentar-se correta quando aplicada aos trechos de código redundantes, mas não aparenta ter o mesmo impacto quando aplicada aos códigos não otimizados de PGs baseados em árvores (SILVA, 2008).

2.3.1.2 O Viés da remoção

De acordo com Soule e Foster (1998), os nós de uma árvore podem ser divididos em viáveis e inviáveis (Figura 6), sendo que:

1. Todos os descendentes de um nó inviável são também inviáveis;
2. Os ancestrais de um nó viável são nós viáveis;
3. Se qualquer nó em uma árvore for viável, então o nó raiz será viável.

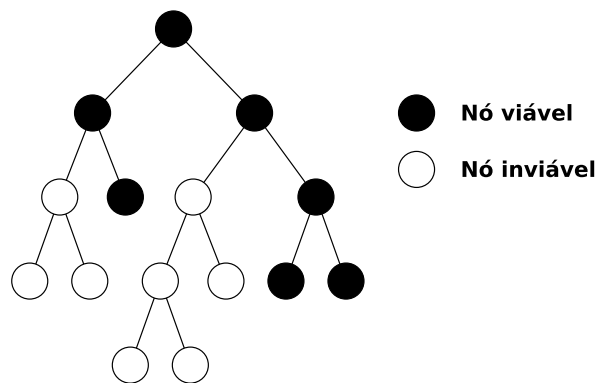


Figura 6 – As árvores apresentam nós viáveis (preenchidos) e nós inviáveis (não preenchidos). Operadores genéticos destrutivos que atuam nos nós viáveis dos pais podem promover o crescimento dos filhos. Adaptado de Soule e Foster (1998, p. 2).

Soule e Foster (1998) observaram que indivíduos que possuem subárvores inviáveis são mais propensos a sobreviver, devido principalmente a uma espécie de assimetria entre os tamanhos das subárvores removidas e inseridas. Observou-se que para garantir

a sobrevivência, a subárvore removida de um indivíduo não deve ser maior do que sua subárvore inviável, caso contrário há a possibilidade de sua aptidão diminuir, dados os efeitos destrutivos da recombinação. Dessa forma, há um viés no sentido de remover subárvores pequenas. Entretanto, não há qualquer viés associado à inserção da nova subárvore: uma subárvore inserida dentro da região inviável, independentemente do tamanho, não modificará a aptidão do indivíduo original, permitindo que as árvores cresçam sem limites.

Para testar sua teoria, Soule e Foster (1998) utilizaram uma versão menos destrutiva da recombinação, proposta por O'Reilly (1995), na qual filhos com aptidão menor que a do pai são descartados e substituídos pelos pais. Uma versão mais rigorosa desta técnica rejeita filhos com aptidão menor ou igual à dos pais. Ambas as versões se mostraram eficazes em retardar o crescimento das árvores, com maior destaque para a última. Resultados semelhantes foram relatados por Langdon e Poli (1998c). Entretanto, evidências encontradas por Luke (2003) indicam que talvez a principal causa para a diminuição do crescimento médio da população seja, na verdade, a taxa de replicação dos pais para as próximas gerações. De acordo também com esses resultados, o *bloat* pode não estar sendo impedido, mas apenas retardado.

2.3.1.3 Profundidade do ponto de cruzamento

Generalizando o conceito de viés de remoção, observa-se que existe forte correlação entre a profundidade do ponto de cruzamento e o efeito sobre a aptidão do filho gerado. De maneira geral, pontos mais profundos minimizam as diferenças de aptidão entre pai e filho.

Assim, o efeito destrutivo do operador da recombinação faz com que árvores maiores e com maior quantidade de pontos mais profundos disponíveis tendam a sofrer pouca modificação e seguirem em frente na evolução, visto que os ramos removidos também são menores neste caso, como é relatado em Igel e Chellapilla (1999), e de Luke (2000a), Luke (2000b) e Luke (2003).

2.3.2 Problemas relacionados à seleção e à busca direcionada pela aptidão

De acordo com Langdon (1998), qualquer técnica de busca que use função de avaliação estática e representações discretas de tamanho variável está sujeita ao *bloat*. No caso da PG, diferentes indivíduos, com diferentes formas e tamanhos, podem compartilhar da mesma medida de aptidão, conforme apresentado na Figura 7. De fato, há diversas maneiras de representar um mesmo indivíduo e apenas a minoria delas corresponde a representações curtas, o que leva à predominância de soluções longas (LANGDON; POLI, 1998b).

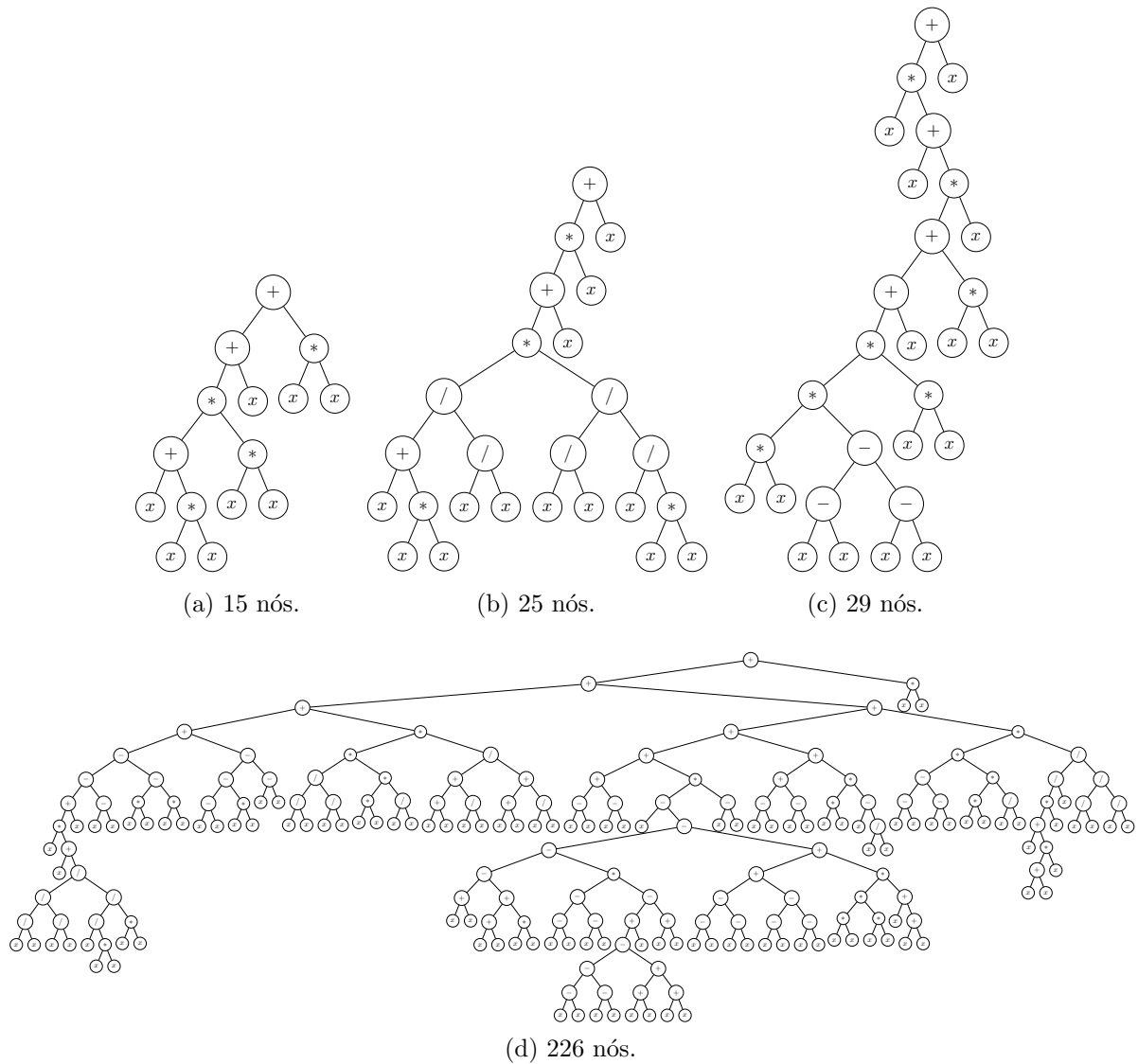


Figura 7 – Árvores de diferentes formatos, mas com mesma aptidão. Função: $f(x) = x^4 + x^3 + x^2 + x$, utilizando 20 pontos $(x, f(x))$, com $x \in [-1, 1]$. Todas essas árvores representam $f(x)$.

Esta teoria não coloca os *introns* como causa primordial para o *bloat* (SILVA, 2008), levando Langdon (1998) a apresentar dois importantes resultados:

1. Operadores de busca desprovidos de viés de comprimento tendem a amostrar árvores maiores;
2. Programas mais longos são favorecidos em competições, já que eles geralmente podem se reproduzir com maior precisão.

Conforme discutido na Seção 2.3.1.1, durante o processo de evolução chega-se a um ponto de convergência em que aos melhores indivíduos são aqueles com aptidão semelhante os das gerações anteriores. De forma geral, o comprimento variável possibilita

maior número de representações longas do que representações curtas para a mesma solução. Assim, espera-se que soluções mais longas ocorram com maior frequência e que estas tenham maior probabilidade de serem selecionadas para aplicação dos operadores genéticos. Ou seja, a seleção baseada puramente em aptidão leva ao *bloat* (LANGDON; POLI, 1998a).

Os resultados apresentados em Langdon e Banzhaf (2000) confirmam a influência da seleção no crescimento dos indivíduos. Um ambiente em que os piores indivíduos são desfavorecidos direciona a evolução rumo a alternativas com melhor aptidão – justamente onde se situam as representações maiores. Quando a seleção é feita de maneira totalmente aleatória, o *bloat* não ocorre, mas há, naturalmente, prejuízos no que diz respeito à qualidade da população. Estudos estatísticos desenvolvidos em Gelly et al. (2005) e em Gelly et al. (2006) também apoiam esta teoria.

2.4 Análise de regressão em CE

Há na literatura diversas pesquisas que combinam Programação Genética e técnicas estatísticas. Em Iba, deGaris e Sato (1995), por exemplo, é apresentado um sistema adaptativo chamado STROGANOFF (*STructured Representation On Genetic Algorithms for Nonlinear Functions Fitting*), descrito como uma abordagem de PG que utiliza os princípios de Computação Evolucionária associados a Análise de Regressão Múltipla. O STROGANOFF é utilizado para ajustar parâmetros internos da PG, mas sem considerar a complexidade do modelo e sem a intenção de mapear o comportamento de todo o programa.

Já em McKay et al. (1999) é apresentada uma combinação de PG e Regressão Contínua para desenvolver dois algoritmos de regressão contínua não linear que são aplicados à construção de modelos de processos químicos. Relevantes contribuições são também apresentadas em Keijzer (2003), Sobester, Nair e Keane (2004), Worm e Chiu (2013), dentre outros. Esta seção discute duas técnicas de Computação Evolucionária que utilizam propriedades de regressão linear múltipla. Antes, porém, são apresentados os fundamentos matemáticos da análise de regressão.³

2.4.1 Análise de regressão

É comum nas Ciências e nas Engenharias a necessidade de explorar relações entre duas ou mais variáveis. Na física clássica, em áreas como cinemática ou dinâmica, é possível estabelecer relações determinísticas cujas fórmulas podem ser facilmente encontradas. Entretanto, em diversas situações observa-se comportamento não determinístico, o que

³ São discutidos apenas os aspectos estatísticos fundamentais ao entendimento do presente trabalho. Estudos mais aprofundados podem ser encontrados em Montgomery e Runger (2010) e em Hines, Montgomery e Borror (2008).

leva o problema para o campo estatístico. À coleção de ferramentas estatísticas que são utilizadas para modelar e explorar relações entre variáveis que estão relacionadas de maneira não determinística dá-se o nome de *análise de regressão*.

O caso mais simples de regressão envolve o estudo com apenas duas variáveis, geralmente denotadas por x e y . Quando pontos pertencentes a um conjunto de duas variáveis são plotados no plano cartesiano bidimensional como, na Figura 8, algum padrão de comportamento pode ser observado.

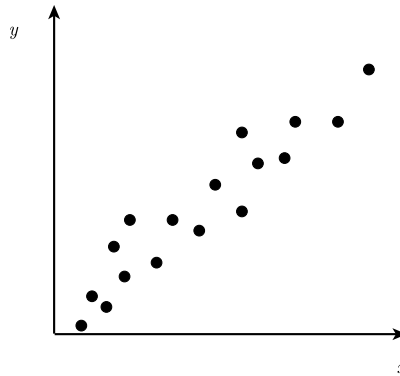


Figura 8 – Gráfico de dispersão.

O gráfico da Figura 8 recebe o nome de *diagrama de dispersão*. Muito embora não pareça haver alguma curva simples que passe exatamente por todos os pontos, é razoável admitir que estes repousam aleatoriamente dispersos em torno de uma linha reta. Se Y é uma variável aleatória que modela este comportamento, sua relação com x pode ser descrita como na Equação (2.7), em que os valores β_i são chamados coeficientes de regressão. O valor β_0 (denominado intercepto) é o ponto no eixo das ordenadas interceptado pela reta (valor esperado quando $x = 0$) e β_1 é a inclinação da reta (tangente do ângulo que a reta faz com o eixo das abscissas).

$$E(Y|x) = \mu_{Y|x} = \beta_0 + \beta_1 x \quad (2.7)$$

É importante observar que a média de Y é uma função linear de x , mas o valor real observado y pode não cair exatamente na linha reta. Assim, para o valor real de Y é necessário considerar também um termo de erro aleatório ϵ , como na Equação (2.8).

$$Y = \beta_0 + \beta_1 x + \epsilon \quad (2.8)$$

Modelos como o descrito em (2.8), que consideram um único regressor ou preditor x e uma variável aleatória dependente ou variável resposta Y , recebem o nome de *regressão*

linear simples. Embora seja ferramenta útil em diversas situações, há casos em que mais de um regressor se faz necessário, levando a modelos como o descrito em (2.9).

$$Y = \beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k + \epsilon \quad (2.9)$$

Uma regressão que contém mais de um regressor, como o descrito em (2.9), é chamada de *regressão linear múltipla*. Este modelo generaliza a regressão linear simples de forma que cada variável dependente (ou de resposta) Y pode estar relacionada a k variáveis independentes ou regressoras que descrevem um hiperplano no espaço k -dimensional das variáveis regressoras $\{x_j\}$. O termo *linear* deve-se ao fato de ser uma função linear de parâmetros desconhecidos β_i . Para que seja possível definir um modelo como esse, é necessário encontrar alguma forma de estimar os valores dos parâmetros β_i e isso pode ser feito por meio do *método dos mínimos quadrados*.

2.4.1.1 Método dos mínimos quadrados

Uma forma de estimar os valores dos coeficientes do modelo de regressão linear múltipla é o método dos mínimos quadrados. Seja $n > k$ observações disponíveis e seja x_{ij} a i -ésima observação ou nível da variável x_j . As observações são

$$(x_{i1}, x_{i2}, \dots, x_{ik}, y_i), \quad i = 1, 2, \dots, n \quad \text{e} \quad n > k.$$

Os dados da regressão múltipla costumam ser representados de forma tabular, como apresentado na Tabela 4.

Tabela 4 – Dados para Regressão Linear Múltipla

x_1	x_2	\cdots	x_k	y
x_{11}	x_{12}	\cdots	x_{1k}	y_1
x_{21}	x_{22}	\cdots	x_{2k}	y_2
\vdots	\vdots		\vdots	\vdots
x_{n1}	x_{n2}	\cdots	x_{nk}	y_n

Em (2.10), cada observação $(x_{i1}, x_{i2}, \dots, x_{ik}, y_i)$ satisfaz o modelo na Equação (2.9):

$$\begin{aligned} y_i &= \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_k x_{ik} + \epsilon_i \\ &= \beta_0 + \sum_{j=1}^k \beta_j x_{ij} + \epsilon_i \quad i = 1, 2, \dots, n \end{aligned} \quad (2.10)$$

A função de mínimos quadrados é dada pela Equação (2.11):

$$L = \sum_{i=1}^n \epsilon_i^2 = \sum_{i=1}^n \left(y_i - \beta_0 - \sum_{j=1}^k \beta_j x_{ij} \right)^2 \quad (2.11)$$

Queremos minimizar L com relação a $\beta_0, \beta_1, \dots, \beta_k$. As estimativas de mínimos quadrados de $\beta_0, \beta_1, \dots, \beta_k$ têm que satisfazer às seguintes derivadas:

$$\left. \frac{\partial L}{\partial \beta_0} \right|_{\hat{\beta}_0, \hat{\beta}_1, \dots, \hat{\beta}_k} = -2 \sum_{i=1}^n \left(y_i - \hat{\beta}_0 - \sum_{j=1}^k \hat{\beta}_j x_{ij} \right) x_{ij} = 0 \quad (2.12a)$$

$$\left. \frac{\partial L}{\partial \beta_j} \right|_{\hat{\beta}_0, \hat{\beta}_1, \dots, \hat{\beta}_k} = -2 \sum_{i=1}^n \left(y_i - \hat{\beta}_0 - \sum_{j=1}^k \hat{\beta}_j x_{ij} \right) x_{ij} = 0 \quad j = 1, 2, \dots, k \quad (2.12b)$$

Reescrevendo as Equações (2.12), obtemos as Equações Normais (2.13) de mínimos quadrados:

$$\begin{array}{ccccccccc} n\hat{\beta}_0 & + & \hat{\beta}_1 \sum_{i=1}^n x_{i1} & + & \hat{\beta}_2 \sum_{i=1}^n x_{i2} & + & \cdots & + & \hat{\beta}_k \sum_{i=1}^n x_{ik} & = & \sum_{i=1}^n y_i \\ \hat{\beta}_0 \sum_{i=1}^n x_{i1} & + & \hat{\beta}_1 \sum_{i=1}^n x_{i1}^2 & + & \hat{\beta}_2 \sum_{i=1}^n x_{i1}x_{i2} & + & \cdots & + & \hat{\beta}_k \sum_{i=1}^n x_{i1}x_{ik} & = & \sum_{i=1}^n y_i x_{i1} \\ \vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\ \hat{\beta}_0 \sum_{i=1}^n x_{ik} & + & \hat{\beta}_1 \sum_{i=1}^n x_{ik}x_{i1} & + & \hat{\beta}_2 \sum_{i=1}^n x_{ik}x_{i2} & + & \cdots & + & \hat{\beta}_k \sum_{i=1}^n x_{ik}^2 & = & \sum_{i=1}^n y_i x_{ik} \end{array} \quad (2.13)$$

Na Equação (2.13), há $p = k + 1$ equações normais, uma para cada um dos coeficientes desconhecidos de regressão. A solução para as equações normais serão os *estimadores de mínimos quadrados* $\hat{\beta}_0, \hat{\beta}_1, \dots, \hat{\beta}_k$ dos coeficientes de regressão. As equações normais podem ser resolvidas por qualquer método apropriado para solução de sistemas de equações lineares.

Outra forma de representar um modelo de regressão linear múltipla é expressar as operações matemáticas com a notação matricial. Se houver k variáveis regressoras e n observações, $(x_{i1}, x_{i2}, \dots, x_{ik}, y_i)$, $i = 1, 2, \dots, n$, o modelo de relacionando os regressores à resposta é:

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_k x_{ik} + \epsilon_i \quad i = 1, 2, \dots, n$$

Esse modelo é um sistema de n equações, que pode ser expresso na notação matricial como

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon} \quad (2.14)$$

sendo

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \quad \mathbf{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1k} \\ 1 & x_{21} & x_{22} & \cdots & x_{2k} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{nk} \end{bmatrix} \quad \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_k \end{bmatrix} \quad \text{e} \quad \boldsymbol{\epsilon} = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix}$$

Em geral, \mathbf{y} é um vetor ($n \times 1$) das observações, \mathbf{X} é uma matriz ($n \times p$) dos níveis das variáveis independentes, $\boldsymbol{\beta}$ é um vetor ($p \times 1$) dos coeficientes de regressão e $\boldsymbol{\epsilon}$ é um vetor ($n \times 1$) dos erros aleatórios. A matriz \mathbf{X} é frequentemente chamada de *matriz do modelo*.

Deseja-se encontrar o vetor dos coeficientes de mínimos quadrados $\hat{\boldsymbol{\beta}}$ que minimiza a soma dos quadrados dos erros. Se esse valor for representado por L , então

$$\begin{aligned}
 L &= \sum_{i=1}^n \epsilon_i^2 \\
 &= \boldsymbol{\epsilon}'\boldsymbol{\epsilon} \\
 &= (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})'(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) \\
 &= \mathbf{y}'\mathbf{y} - \boldsymbol{\beta}'\mathbf{X}'\mathbf{y} - \mathbf{y}'\mathbf{X}\boldsymbol{\beta} + \boldsymbol{\beta}'\mathbf{X}'\mathbf{X}\boldsymbol{\beta} \\
 &= \mathbf{y}'\mathbf{y} - 2\boldsymbol{\beta}'\mathbf{X}'\mathbf{y} + \boldsymbol{\beta}'\mathbf{X}'\mathbf{X}\boldsymbol{\beta}
 \end{aligned} \tag{2.15}$$

A forma apresentada pela Equação (2.15) deve-se ao fato de $\boldsymbol{\beta}'\mathbf{X}'\mathbf{y}$ ser uma matriz (1×1), sendo portando um escalar, e sua transposta $(\boldsymbol{\beta}'\mathbf{X}'\mathbf{y})' = \mathbf{y}'\mathbf{X}\boldsymbol{\beta}$ ser o mesmo escalar. O estimador de mínimos quadrados $\hat{\boldsymbol{\beta}}$ é a solução para $\boldsymbol{\beta}$ nas equações

$$\begin{aligned}
 \frac{\partial L}{\partial \boldsymbol{\beta}} &= \mathbf{0} \Rightarrow -2\mathbf{X}'\mathbf{y} + 2\mathbf{X}'\mathbf{X}\hat{\boldsymbol{\beta}} = \mathbf{0} \\
 &\Rightarrow \mathbf{X}'\mathbf{X}\hat{\boldsymbol{\beta}} = \mathbf{X}'\mathbf{y}
 \end{aligned} \tag{2.16}$$

As estruturas presentes em (2.16) são as equações normais de mínimos quadrados na forma matricial. Com o objetivo de determinar os valores estimados da matriz $\boldsymbol{\beta}$, ou seja, $\hat{\boldsymbol{\beta}}$, as seguintes operações podem ser aplicadas na Equação (2.16):

$$\begin{aligned}
 \mathbf{X}'\mathbf{X}\hat{\boldsymbol{\beta}} &= \mathbf{X}'\mathbf{y} \Rightarrow (\mathbf{X}'\mathbf{X})^{-1}(\mathbf{X}'\mathbf{X})\hat{\boldsymbol{\beta}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y} \\
 &\Rightarrow \mathbf{I}\hat{\boldsymbol{\beta}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y} \quad (\mathbf{I} = \text{matriz identidade}) \\
 &\Rightarrow \hat{\boldsymbol{\beta}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}
 \end{aligned} \tag{2.17}$$

A Equação (2.17) resultante corresponde à *estimativa de mínimos quadrados* de $\boldsymbol{\beta}$, havendo $p = k + 1$ equações normais para $p = k + 1$ incógnitas (valores de $\hat{\beta}_0, \hat{\beta}_1, \dots, \hat{\beta}_k$). A Equação (2.16), descrita de forma matricial, corresponde a:

$$\begin{bmatrix}
 n & \sum_{i=1}^n x_{i1} & \sum_{i=1}^n x_{i2} & \cdots & \sum_{i=1}^n x_{ik} \\
 \sum_{i=1}^n x_{i1} & \sum_{i=1}^n x_{i1}^2 & \sum_{i=1}^n x_{i1}x_{i2} & \cdots & \sum_{i=1}^n x_{i1}x_{ik} \\
 \vdots & \vdots & \vdots & \ddots & \vdots \\
 \sum_{i=1}^n x_{ik} & \sum_{i=1}^n x_{ik}x_{i1} & \sum_{i=1}^n x_{ik}x_{i2} & \cdots & \sum_{i=1}^n x_{ik}^2
 \end{bmatrix}
 \begin{bmatrix}
 \hat{\beta}_0 \\
 \hat{\beta}_1 \\
 \vdots \\
 \hat{\beta}_k
 \end{bmatrix}
 =
 \begin{bmatrix}
 \sum_{i=1}^n y_i \\
 \sum_{i=1}^n x_{i1}y_i \\
 \vdots \\
 \sum_{i=1}^n x_{ik}y_i
 \end{bmatrix}$$

Dessa forma, a execução desta multiplicação matricial resultará a forma das equações normais presentes na Equação (2.13).

As matrizes que compõem a Equação (2.16) trazem consigo muitas informações. Os elementos da diagonal de $\mathbf{X}'\mathbf{X}$ são as somas dos quadrados dos elementos nas colunas de \mathbf{X} e os demais são as somas dos produtos cruzados dos elementos nas colunas de \mathbf{X} . Por fim, os elementos de $\mathbf{X}'\mathbf{y}$ são as somas dos produtos cruzados das colunas de \mathbf{X} e das observações $\{y_i\}$.

O modelo de regressão é

$$\hat{y}_i = \hat{\beta}_0 + \sum_{j=1}^k \hat{\beta}_j x_{ij} \quad i = 1, 2, \dots, n \quad (2.18)$$

Em notação matricial:

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\boldsymbol{\beta}} \quad (2.19)$$

A diferença entre as observações y_i e o valor ajustado \hat{y}_i é um *resíduo*, $e_i = y_i - \hat{y}_i$. O vetor coluna de resíduos é denotado por

$$\mathbf{e} = \mathbf{y} - \hat{\mathbf{y}} \quad (2.20)$$

Se aceitamos as suposições de que os erros ϵ_i são estatisticamente independentes, com média zero e variância σ^2 , pode-se considerar que os estimadores de mínimos quadrados $\hat{\beta}_0, \hat{\beta}_1, \dots, \hat{\beta}_k$ são estimadores não tendenciosos de $\beta_0, \beta_1, \dots, \beta_k$. Um estimador para a variância, necessário para discussões posteriores sobre testes de hipóteses, pode ser obtido por:

$$\hat{\sigma} = \frac{SQ_E}{n - p} \quad (2.21)$$

O termo SQ_E da Equação (2.21) recebe o nome de *soma dos quadrados dos resíduos* e é definido em:

$$\begin{aligned} SQ_E &= \sum_{i=1}^n (y_i - \hat{y}_i)^2 \\ &= \sum_{i=1}^n e_i^2 \\ &= \mathbf{e}'\mathbf{e} \\ &= (\mathbf{y} - \hat{\mathbf{y}})'(\mathbf{y} - \hat{\mathbf{y}}) \\ &= \mathbf{y}'\mathbf{y} - \hat{\boldsymbol{\beta}}'\mathbf{X}'\mathbf{y} + \mathbf{y}'\mathbf{X}\hat{\boldsymbol{\beta}} + \hat{\boldsymbol{\beta}}'\mathbf{X}'\mathbf{X}\hat{\boldsymbol{\beta}} \\ &= (\mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}})'(\mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}}) \\ &= \mathbf{y}'\mathbf{y} - 2\hat{\boldsymbol{\beta}}'\mathbf{X}'\mathbf{y} + \hat{\boldsymbol{\beta}}'\mathbf{X}'\mathbf{X}\hat{\boldsymbol{\beta}} \\ &= \mathbf{y}'\mathbf{y} - \hat{\boldsymbol{\beta}}'\mathbf{X}'\mathbf{y} \end{aligned} \quad (2.22)$$

Já o denominador da Equação (2.21), $n - p$, é chamado *grau de liberdade do erro*, sendo p o número de parâmetros do modelo de regressão linear múltipla.

2.4.1.2 Coeficiente de determinação múltipla

Uma estatística global para avaliar o ajuste do modelo é o *coeficiente de regressão múltipla* R^2 , cuja fórmula é descrita em (2.23).

$$R^2 = \frac{SQ_R}{SQ_T} = 1 - \frac{SQ_E}{SQ_T}, \quad (2.23)$$

onde

$$SQ_R = \hat{\beta}' \mathbf{X}' \mathbf{y} - \frac{(\sum_{i=0}^n y_i)^2}{n}$$

$$SQ_T = \mathbf{y}' \mathbf{y} - \frac{(\sum_{i=1}^n y_i)^2}{n}$$

R^2 pode assumir valores reais entre 0 e 1, onde 1 indica que modelo ajustou-se perfeitamente aos valores amostrados. Naturalmente, 0 indica que o modelo falhou completamente.

Uma variação de R^2 frequentemente utilizada é o *coeficiente de determinação múltipla ajustado*, ou, simplesmente, $R_{ajustado}^2$, apresentado em (2.24).

$$R_{ajustado}^2 = 1 - \frac{SQ_E/(n-p)}{SQ_T/(n-1)} \quad (2.24)$$

Esta métrica permite avaliar a qualidade do modelo mediante o número de parâmetros p (regressores) presentes. Como é descrito em (2.24), o valor de $R_{ajustado}^2$ crescerá apenas se a adição de um novo termo (parâmetro) reduzir significativamente a média quadrática dos erros. Assim, a adição ao modelo de termos que não sejam significantes à modelagem da resposta será penalizada.

2.4.1.3 Testes de hipóteses sobre os coeficientes de regressão

Para determinar o valor potencial de cada uma das variáveis regressoras em um modelo de regressão, é comum utilizar *testes de hipóteses* para seus coeficientes. Como, é descrito em (2.18), os coeficientes em um modelo de regressão podem ser tratados como medida de “peso” para os regressores, ponderando a somatória. Dessa forma, decidir se um coeficiente é ou não, dentro de certo nível de significância α , estatisticamente igual a zero pode levar à diminuição do número de regressores em um modelo.

De fato, a adição de uma variável ao modelo sempre faz com que a soma de quadrados para a regressão aumente, ao passo que a soma dos quadrados dos erros diminua. Entretanto, a adição de uma variável sem importância pode aumentar a média quadrática dos erros, o que faz diminuir a utilidade do modelo.

Para testar se β_j é, dentro de certo nível de significância, estatisticamente igual a 0, utiliza-se as hipóteses (nula e alternativa) descritas em (2.25), com a estatística de

testes apresentada na Equação (2.26).

$$\begin{aligned} H_0 : \beta_j &= 0 \\ H_1 : \beta_j &\neq 0 \end{aligned} \quad (2.25)$$

$$t_0 = \frac{\hat{\beta}_j}{\sqrt{\sigma^2 C_{jj}}} \quad (2.26)$$

Em (2.26), o termo C_{jj} é o elemento da diagonal de $(\mathbf{X}'\mathbf{X})^{-1}$ correspondente a $\hat{\beta}_j$. Como raramente o valor de σ^2 é conhecido, na prática ele é substituído pelo valor estimado $\hat{\sigma}^2$, já apresentado na Equação (2.21). A hipótese nula H_0 será rejeitada caso $|t_0| > t_{\alpha/2, n-p}$.

2.4.2 Técnicas de CE baseadas em Regressão Regressão

Nesta seção serão apresentadas duas técnicas recentes de Computação Evolucionária que utilizam conceitos de Regressão Linear Múltipla: a Programação Genética com Regressão Múltipla e Programação *Kaizen*.

2.4.2.1 MRGP

A Programação Genética com Regressão Múltipla (ou MRGP, do inglês *Multiple Regression Genetic Programming*) foi apresentada por Arnaldo, Krawiec e O'Reilly (2014) como forma de melhorar a qualidade de produção da PG. Esta abordagem decompõe e combina linearmente as sub-expressões de um programa por meio de regressão linear múltipla na variável alvo, produzindo assim a previsão do modelo de regressão múltipla resultante. Nesta técnica, avalia-se o modelo, em detrimento do resultado da execução do programa.

Arnaldo, Krawiec e O'Reilly (2014) utilizaram a MRGP para melhorar a adequação de uma solução final evoluída, obtendo soluções mais eficientes do que os gerados por PG ou por Regressão Múltipla. Foi realizada também a melhoria de indivíduos ao longo da evolução, o que proporcionou resultados melhores do que a própria MRGP executada apenas no final.

A principal diferença entre a PG e a MRGP é o fato de eliminar a comparação entre o resultado final do programa com a variável alvo \mathbf{y} . A técnica assume que todas as sub-expressões de um programa podem ser ajustadas na forma de uma combinação linear conforme regride para a variável alvo. Assume-se também que o modelo de regressão resultante seja superior à saída do programa, já que esse desconsidera o potencial das saídas aninhadas. A variável \mathbf{y} é comparada, portanto, ao modelo de regressão.

Tal como a PG, a MRGP trabalha a partir de um conjunto de dados (conjunto de treinamento ou de “casos de adequação” ou *dataset*), composto por m linhas e n

colunas. Existe também um vetor de destino, com n posições. O MRGP é apresentado no Algoritmo 3.

Algoritmo 3 Algoritmo básico do MRGP

- 1: Crie uma matriz \mathbf{F}_{nk} , onde n é a quantidade de elementos do *dataset* e k é o número de nós da árvore
- 2: **para** i de 1 até n **faça**
- 3: Aplique a entrada i do *dataset* à árvore
- 4: Execute o programa (com a análise da árvore convencional) e armazenamos a saída de cada subexpressão depois que ela é executada (ver Figura 9)
- 5: Armazene o resultado de cada nó j na posição $i \times j$ de \mathbf{F}
- 6: **fim para**
- 7: Crie \mathbf{F}' acrescentando a \mathbf{F} uma coluna de 1's
- 8: Mapeie os valores de \mathbf{F}' para a saída desejada y pela técnica de Regressão Linear Múltipla, que produz uma combinação linear que minimiza os erros de predição de \hat{y} .

$$y = \mathbf{F}'\boldsymbol{\beta} + \epsilon \quad (2.27)$$

- 9: Avalie a qualidade do modelo regredido calculando sua saída $\hat{y} = \mathbf{F}\boldsymbol{\beta}$ e comparando com a saída desejada y :

$$(\mathbf{y} - \hat{\mathbf{y}})^T(\mathbf{y} - \hat{\mathbf{y}}) = \boldsymbol{\epsilon}^T \boldsymbol{\epsilon}$$

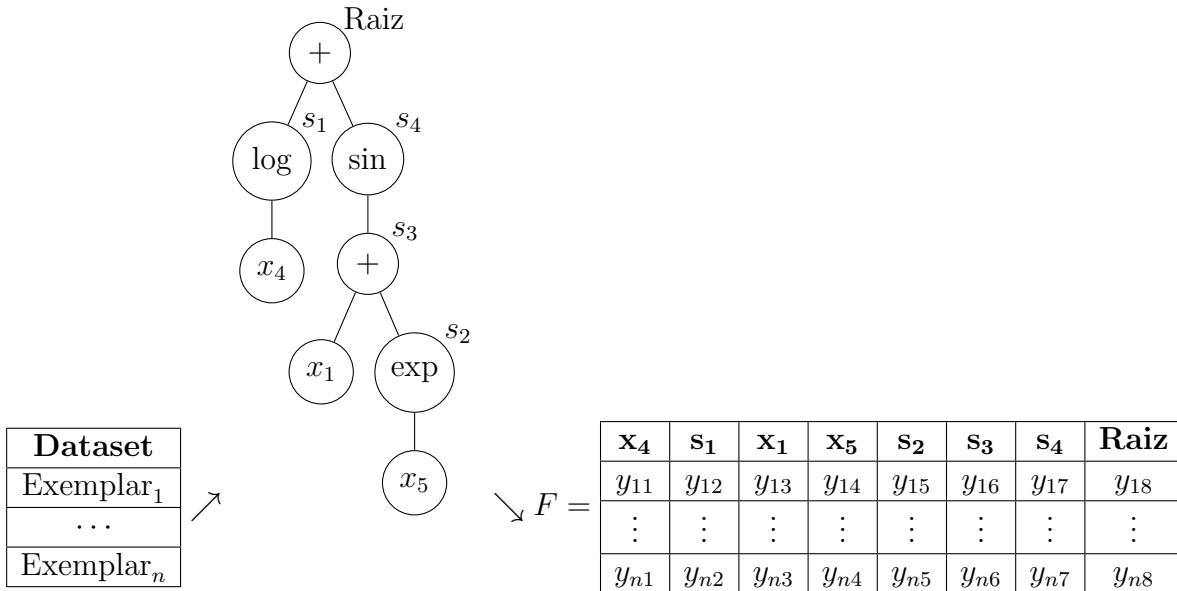


Figura 9 – Avaliação de uma árvore nó a nó. Cada valor do *dataset* é submetido à árvore e a computação é executada passo a passo. Os valores locais são armazenados para formar uma matriz.

Existem dois casos que podem comprometer o adequado funcionamento da MRGP:

- 1) quando a matriz \mathbf{F} possuir número de características k grande em relação ao número

de exemplares n ; 2) se um número suficiente de colunas de \mathbf{F} não forem linearmente independentes.

Como o número de regressores corresponde à quantidade de subexpressões, deve-se manter as árvores da população com o menor número possível de nós. Assim, o *bloat* deve ser combatido já nas primeiras gerações, para aumentar a eficiência e tentar minimizar a quantidade de árvores que se enquadrem no caso 1. Arnaldo, Krawiec e O'Reilly (2014) utilizam o algoritmo NSGA-II (DEB et al., 2002) para criar uma versão de Programação Genética Multiobjetivo que evolui árvores com base nos objetivos de minimização do erro e da complexidade do modelo. Neste contexto, entende-se complexidade como a soma da quantidade de sub-expressões, como definido em Vladislavleva (2008).

Já o caso 2 é tratado com a utilização do Algoritmo LARS (EFRON et al., 2004), no qual variáveis são gradualmente inseridas ao modelo enquanto os coeficientes são continuamente movidos em direção ao seu valor de mínimo quadrado, até que todas as variáveis tenham sido incluídas. As etapas que este algoritmo envolve são melhor detalhadas em Hastie, Tibshirani e Friedman (2009).

2.4.2.2 Programação *Kaizen*

A Programação *Kaizen* (KP) é uma heurística de Computação Evolucionária desenvolvida por De Melo (2014). Enquanto a Programação Genética baseia-se na Teoria da Evolução para encontrar o modelo mais adequado à resolução um problema, a Programação *Kaizen* utiliza-se dos conceitos de Melhoria Contínua da Metodologia Japonesa *Kaizen* para construir uma resposta. Diferentemente da Programação Genética, em que cada indivíduo é considerado uma solução potencial dentro de uma população de soluções individuais, a KP analisa partes dessa solução, que são combinadas ao final do processo para verificação da aderência do modelo.

Essa heurística é baseada no conceito de Eventos *Kaizen*, em que pequenos times formados por funcionários e gerentes (especialistas no negócio escolhidos por seu conhecimento na área) reúnem-se com o intuito de encontrar alguma forma de solucionar determinado problema. Ideias são propostas pelos especialistas para resolver partes dos problemas, ao invés de constituírem soluções integrais. A resposta será dada pela junção de todas as ideias parciais. Em cada ciclo, ideias que não se provarem capazes de melhorar a solução já existente são descartadas. Se ao final de um ciclo não forem observadas melhorias significativas no modelo vigente, volta-se à solução anterior, considerada a melhor até o momento. Um novo modelo só é aceito, portanto, quando ele for comprovadamente melhor do que o existente.

Os eventos *Kaizen* costumam empregar a metodologia *Plan-Do-Check-Act*, também conhecida por PDCA (GITLOW; HOWARD, 1989), para guiar o processo de melhoria contínua. O PDCA consiste em um ciclo onde ações são planejadas, executadas e verificadas,

levando a novas ações que são direcionadas com base nos resultados obtidos. A experiência proporcionada por cada ciclo faz com que os especialistas aprendam ainda mais sobre o problema, podendo, em ciclos posteriores, oferecer ideias de maior qualidade e evitar ideias que não tragam contribuições reais. Seguindo a analogia da aplicação do ciclo PDCA, a Programação *Kaizen* aplica cada etapa desta metodologia, conforme apresentado no Algoritmo 4.

Algoritmo 4 Programação *Kaizen*, abstração

- 1: crie uma equipe de especialistas;
 - 2: defina um objetivo a ser alcançado;
 - 3: **enquanto** o objetivo não for alcançado **faça**
 - 4: PLAN: a equipe realiza um *brainstorming* e cada especialista propõe uma ideia para resolver parte do problema;
 - 5: DO: as ideias atuais e as novas (criadas a partir da segunda iteração) são aplicadas ao problema e unidas para se tornarem uma solução completa;
 - 6: CHECK: avaliar a solução, então cada ideia (antiga e nova) é analisada e sua contribuição para resolver o problema é medida;
 - 7: ACT: se a solução é melhorada, então as melhores ideias são selecionadas e se tornam o novo padrão, que é apresentado à equipe junto com cada contribuição, melhorando o conhecimento do problema. Crie outro Evento *Kaizen* com uma nova equipe se o atual ficar preso em um ótimo local;
 - 8: **fim enquanto**
 - 9: **retorne** as ideias com contribuição significativa e a solução final.
-

Na fase PLAN (Planejamento), um time de especialistas fornece ideias para resolver o problema proposto. Como no começo há pouco ou nenhum conhecimento sobre os detalhes do problema, é certo que não haja boas ideias logo de início. Isso será melhorado ao longo do processo.

Em um problema de regressão simbólica, os especialistas são regressores K_j , com $j = 1, \dots, t$, sendo t o tamanho do time de especialistas. Cada integrante do time fornecerá partes de expressões matemáticas que, quando combinadas, poderão resolver o problema. Se $t = 3$, as ideias fornecidas por cada especialista K_j podem ser, por exemplo, $K_1 = 1/x$; $K_2 = \sin(x^2)$; $K_3 = \cos(x) + e^x$.

Em cada ciclo, existe um modelo (uma solução), considerado o melhor até o momento, que almeja-se melhorar. Este modelo é formado por regressores que já foram propostos e testados. Se cada regressor for representado como a coluna de uma matriz e cada linha corresponder à resposta dada por ele quando submetido a uma entrada (linha) do *dataset*, uma matriz que representa este padrão $STD_{n,t}$ (são t especialistas e o *dataset* possui n linhas) pode ser construída:

$$STD = \begin{bmatrix} STD_{11} & \dots & STD_{1t} \\ \vdots & \dots & \vdots \\ STD_{n1} & \dots & STD_{nt} \end{bmatrix}$$

Na fase DO (Faça), cada especialista apresenta sua ideia para que ela possa ser aplicada ao problema, ou seja, calculada. Com os resultados é possível criar, com o mesmo método utilizado para construir $STD_{n,t}$, uma matriz $TRIAL_{n,w}$, em que w é a quantidade de novos regressores:

$$TRIAL = \begin{bmatrix} TRIAL_{11} & \dots & TRIAL_{1w} \\ \vdots & \dots & \vdots \\ TRIAL_{n1} & \dots & TRIAL_{nw} \end{bmatrix}$$

As soluções parciais representadas na matriz $TRIAL$ são obtidas pela geração de novas ideias aleatórias por parte dos especialistas, pela combinação de soluções parciais propostas por diferentes especialistas ou pela combinação de novas ideias com as já existentes no modelo – este último passo faz uso do conhecimento já adquirido pelo modelo. Este processo de combinar ideias assemelha-se ao (na verdade, costuma ser usado como) operador de recombinação em Programação Genética baseada em árvores. Dessa forma, novas soluções parciais podem ser combinadas com ideias pré-existentes pela simples junção destas duas matrizes, obtendo-se assim uma matriz K :

$$K = \begin{bmatrix} STD_{11} & \dots & STD_{1t} & TRIAL_{11} & \dots & TRIAL_{1w} \\ \vdots & \dots & \vdots & \vdots & \dots & \vdots \\ STD_{n1} & \dots & STD_{nt} & TRIAL_{n1} & \dots & TRIAL_{nw} \end{bmatrix}$$

Na fase CHECK, a Programação *Kaizen* faz uso das ideias criadas pelos especialistas, em conjunto com as ideias pré-existentes, para criar um modelo linear com base nas saídas esperadas (y) existentes no *dataset*. Para tanto, utiliza-se o Método dos Mínimos Quadrados para criar um modelo linear. Se o time for composto por três especialistas, então o seguinte modelo será obtido:

$$\hat{y}_i = \hat{\beta}_1 K_{i,1} + \hat{\beta}_2 K_{i,2} + \hat{\beta}_3 K_{i,3}, \quad (2.28)$$

onde $\hat{y}_i, i = 1, \dots, n$ é a saída calculada com base nos coeficientes $\hat{\beta}_1, \hat{\beta}_2$ e $\hat{\beta}_3$, que são os coeficientes estimados em (2.29):

$$\hat{\beta} = (K'K)^{-1}K'y \quad (2.29)$$

O modelo gerado é linear nos parâmetros $\hat{\beta}_i$, mas não nos regressores gerados por K_i . Além disso, embora o intercepto não seja definido explicitamente, ele pode estar presente, uma vez que algum regressor pode vir a gerar uma constante.

Por meio do modelo linear, a Programação *Kaizen* consegue identificar quais ideias têm maior impacto na construção da solução final. Executando testes de hipóteses sobre os coeficientes é possível saber qual não é estatisticamente diferente de zero, de forma que esses podem ser suprimidos da solução final sem impactar negativamente o modelo. Tal teste é efetuado considerando o *valor-p*, obtido a partir de um teste de hipóteses bilateral em uma distribuição *t* de *Student*. O teste de hipóteses é executado com significância α tipicamente igual a 5%. Estabelece-se também um limiar θ que define o valor mínimo que um coeficiente pode ter para não ser considerado zero. Além disso, ideias duplicadas são também penalizadas.

Após a determinação de quais ideias são de fato relevantes, o modelo linear é construído com os regressores que sobraram e seus respectivos coeficientes. A qualidade do modelo é mensurada com base no coeficiente de determinação R^2 , muito utilizado em Estatística para medir o ajustamento de um modelo que estima dados de um *dataset*.

Na fase ACT, a comparação entre o modelo desenvolvido na fase CHECK e o modelo atual é realizada. Caso o novo modelo apresente maior qualidade do que o existente, a substituição será feita.

Uma situação também possível é a estagnação em ótimos locais. Neste caso, um novo time de especialistas deve ser criado. Além disso, um modelo estagnado pode ser provisoriamente abandonado para dar lugar a um novo ciclo de busca, com novos especialistas e novas ideias, na tentativa de encontrar respostas de melhor qualidade.

A Programação *Kaizen* já foi empregada com sucesso em diferentes contextos. Ela foi utilizada para prever a resistência à compressão do concreto de alto desempenho, apresentando resultados superiores a outras técnicas já empregadas e disponíveis na literatura (MELO; BANZHAF, 2017); na construção de modelos para problemas de classificação (de Melo; BANZHAF, 2015); no auxílio na decisão de concessão de créditos (de Melo; BANZHAF, 2016), dentre outras aplicações.

2.4.3 O problema das matrizes singulares

A determinação dos coeficientes de mínimos quadrados é de fundamental importância para o funcionamento da MRGP e da Programação *Kaizen*. Quando se utiliza o método apresentado na Seção 2.4.1.1, deve ser cumprido um pré-requisito sobre a matriz do modelo: ela deve, necessariamente, ser *não singular*⁴, para que \mathbf{X}^{-1} possa ser calculada. Esta questão está estritamente ligada ao valor do posto⁵ de \mathbf{X} , representado aqui por $\text{Posto}(\mathbf{X})$. Considerando que \mathbf{X} possui n linhas e k colunas, se $n \geq k$ e $\text{Posto}(\mathbf{X}) = k$, então \mathbf{X} é não singular e há garantias de inversa para $\mathbf{X}'\mathbf{X}$. Porém, se $n \leq k$ e $\text{Posto}(\mathbf{X}) = n$, então

⁴ Matrizes não singulares têm o determinante diferente de zero. Este é um pré-requisito para a inversão da matriz.

⁵ O posto de uma matriz \mathbf{X} consiste no número de linhas linearmente independentes de \mathbf{X} .

o número de equações não supera o número de regressores. Se $\text{Posto}(\mathbf{X}) = \min(n, k)$, então dizemos que \mathbf{X} é *posto completo*.

Embora esta preocupação não esteja presente na maioria dos problemas de análise de regressão, visto que dificilmente os dados da regressão representarão matrizes com posto incompleto, o mesmo não pode ser dito em relação às matrizes geradas pela MRGP e pela PK. Em ambos os casos, trabalha-se com dados gerados a partir das saídas de árvores e não diretamente com os valores do conjunto de treinamento. Basta, por exemplo, que duas árvores gerem colunas com valores linearmente dependentes para uma matriz singular ser criada.

Conforme já discutido, a MRGP utiliza o Algoritmo LARS para contornar este problema. Já a PK não trata diretamente deste assunto, a não ser pela preocupação de penalizar ideias iguais presentes no modelo⁶. Um método mais geral para o cálculo da inversa (e do problema dos mínimos quadrados) é apresentado a seguir.

2.4.3.1 A pseudo inversa de Moore-Penrose

A matriz pseudo inversa de Moore-Penrose (MOORE, 1920; PENROSE, 1955) é um método geral para solucionar sistemas de equações lineares do tipo $\mathbf{b} = \mathbf{A}\mathbf{y}$, onde \mathbf{b} é uma matriz $m \times 1$, \mathbf{A} é $m \times n$ e \mathbf{y} é $n \times 1$. Tal solução é fornecida pela Equação (2.30).

$$\mathbf{y} = \mathbf{A}^\dagger \mathbf{b} \quad (2.30)$$

Em (2.30), \mathbf{A}^\dagger é a pseudo inversa de Moore-Penrose, que obedece às seguintes propriedades:

- I) $\mathbf{A}\mathbf{A}^\dagger\mathbf{A} = \mathbf{A}$;
- II) $\mathbf{A}^\dagger\mathbf{A}\mathbf{A}^\dagger = \mathbf{A}^\dagger$;
- III) $(\mathbf{A}\mathbf{A}^\dagger)' = \mathbf{A}\mathbf{A}^\dagger$;
- IV) $(\mathbf{A}^\dagger\mathbf{A})' = \mathbf{A}^\dagger\mathbf{A}$.

Dependendo da relação entre m e n , há três situações possíveis:

- $m = n$: Neste caso, $\mathbf{A}^\dagger = \mathbf{A}^{-1}$, desde que \mathbf{A} possua posto completo, ou seja, $\text{Posto}(\mathbf{A}) = n$;

⁶ Ideias iguais correspondem a subárvores iguais. Na prática, isso geraria colunas com valores iguais, uma forma básica de dependência linear que resultaria em matrizes singulares.

- $m > n$: A pseudo inversa oferece a solução fornecida pela pseudo inversa minimiza $\|\mathbf{b} - \mathbf{A}\mathbf{y}\|$. Em situações como esta, mesmo havendo mais equações do que variáveis livres é encontrada a solução de mínimos quadrados mais adequada possível;
- $m < n$: Embora exista, neste caso, infinitas soluções, a pseudo inversa fornece uma solução particular que representa a norma mínima de \mathbf{y} .

Em casos em que a matriz \mathbf{A} possui posto completo, a pseudo inversa pode ser calculada da seguinte forma:

- $\mathbf{A}^\dagger = \mathbf{A}'(\mathbf{A}\mathbf{A}')^{-1}$, se $m < n$;
- $\mathbf{A}^\dagger = (\mathbf{A}'\mathbf{A})^{-1}\mathbf{A}'$, se $m > n$.

Porém, se \mathbf{A} não possuir posto completo, é necessário recorrer às propriedades de um tipo especial de fatoração de matrizes chamada decomposição de valores singulares ou SVD – do inglês *Singular Value Decomposition*. A SVD da matriz \mathbf{A} , $m \times n$, é apresentada em (2.31):

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}' \quad (2.31)$$

Em (2.31), \mathbf{U} é uma matriz ortogonal $m \times m$ cujas colunas são autovetores de $\mathbf{A}\mathbf{A}'$. \mathbf{V} é ortogonal de dimensão $n \times n$ cujas colunas correspondem aos autovetores de $\mathbf{A}'\mathbf{A}$. Já $\mathbf{\Sigma}$ é uma matriz diagonal, de dimensão $m \times n$. Os elementos de sua diagonal, denominados $\sigma_1, \dots, \sigma_r$ são raízes quadradas dos autovetores não nulos de $\mathbf{A}\mathbf{A}'$ e $\mathbf{A}'\mathbf{A}$. Eles são os vetores valores singulares da matriz \mathbf{A} e preenchem os r primeiros lugares da diagonal principal de $\mathbf{\Sigma}$. Observa-se que $\text{Posto}(\mathbf{A}) = r$. É possível assim determinar a matriz pseudo inversa por meio da Expressão (2.32):

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}' \Rightarrow \mathbf{A}^\dagger = \mathbf{V}\mathbf{\Sigma}^\dagger\mathbf{U}' \quad (2.32)$$

Em (2.32), $\mathbf{\Sigma}^\dagger$ é calculada da seguinte forma:

$$\mathbf{\Sigma} = \left[\begin{array}{ccc|c} \sigma_1 & & & 0 \\ & \ddots & & \\ & & \sigma_r & 0 \\ \hline 0 & & & 0 \end{array} \right] \Rightarrow \mathbf{\Sigma}^\dagger = \left[\begin{array}{ccc|c} \frac{1}{\sigma_1} & & & 0 \\ & \ddots & & \\ & & \frac{1}{\sigma_r} & 0 \\ \hline 0 & & & 0 \end{array} \right] \quad (2.33)$$

Em (2.33), sempre que σ_i for suficientemente pequeno (menor que certo limiar pré-estabelecido), ele poderá ser considerado zero.

2.4.3.2 Generalização do método dos mínimos quadrados

O estabelecimento do conceito da inversa de Moore-Penrose nos permite encontrar o vetor $\hat{\beta}$ que minimiza a diferença dos quadrados de $\mathbf{y} = \mathbf{X}\beta$ para o caso de $n \leq k$ e $\text{Posto}(\mathbf{X}) = n$. Situações como esta, em que o posto de \mathbf{X} é dito “deficiente”, o seguinte resultado pode ser utilizado:

$$\hat{\beta} = \mathbf{X}^\dagger \mathbf{y} \quad (2.34)$$

Se \mathbf{X} possui inversa, então a Equação (2.34) fornecerá o mesmo resultado que a Equação (2.17). Porém, se \mathbf{X} for singular ou não quadrada, a pseudo-inversa proporcionará a *melhor* solução para $\mathbf{y} = \mathbf{X}\beta$, ou seja, a solução, dentre as infinitas soluções possíveis, que minimiza as diferenças dos quadrados dos erros (DEUFLHARD; SAUTTER, 1980).

2.5 Considerações sobre o capítulo

Este capítulo discutiu os princípios básicos de Programação Genética, tratando temas como semântica e métodos baseados em regressão. Discutiu-se também o efeito *bloat*, que é um dos principais problemas encontrados na evolução de indivíduos de tamanho e formato variáveis.

Embora seja uma técnica aplicada com sucesso na solução de diversos problemas práticos, há ainda alguns desafios a serem superados. Questões relacionadas à capacidade de generalização da PG, complexidade dos resultados gerados, formas adequadas para medir e/ou prever a capacidade da PG solucionar problemas, formas de utilizar a consciência semântica para melhorar os resultados estão entre os problemas ainda sem solução (O’NEILL et al., 2010)

No que se refere à complexidade dos resultados gerados, vemos o controle do *bloat* como grande desafio a ser vencido. Há em Silva (2008) interessante discussão acerca das causas do *bloat* que culmina em irônico resultado: o único elemento da evolução que, quando retirado, elimina instantaneamente o *bloat* é justamente aquele que não pode ser ignorado, ou seja, a busca por indivíduos com melhor aptidão. De fato, se a priorização pela sobrevivência do mais apto não existisse, *introns* impregnados no código de alta qualidade não proliferariam, não haveria problemas de defesa contra a recombinação, o viés da remoção não seria um problema e indivíduos com grande quantidade de nós teriam a mesma chance de sobreviver que aqueles de tamanho reduzido. Assim, a Programação Genética ainda carece de alguma técnica que consiga gerar indivíduos de alta qualidade e que controle o *bloat* de forma natural para que sua utilização possa ser ainda mais disseminada nas Ciências e nas Engenharias.

Já no que tange à capacidade de convergência e qualidade das respostas obtidas, alguma atenção deve ser dada à maneira como os operadores genéticos são aplicados. Os tradicionais operadores de recombinação e de mutação utilizam-se dos aspectos sintáticos dos indivíduos para gerar novos membros da população. Em outras palavras, as representações sintáticas são as principais diretrizes da busca, mas no final o que determina o quão apto é um indivíduo é sua semântica (MORAGLIO; KRAWIEC; JOHNSON, 2011). Operadores semânticos, como os discutidos na Seção 2.2.5.2, contribuem para a diminuição do erro mas, como veremos em capítulos posteriores, podem não tratar o *bloat* adequadamente.

Métodos que combinam GP e técnicas estatísticas podem figurar como uma alternativa para a solução dos problemas aqui elencados. Estes métodos acrescentam informações ao processo de evolução, levando a compreensão semântica dos indivíduos gerados a um novo nível.

Definição de operador de recombinação com regressão múltipla

Em Beadle e Johnson (2008), são definidas três estratégias básicas normalmente utilizadas para melhorar o mecanismo de treinamento em PG. A primeira consiste em modificar o método de seleção dos pais, o que pode ser feito com torneios especializados (LUKE; PANAIT, 2002; LUKE; PANAIT, 2006) ou abordagens multiobjetivo (BLEULER et al., 2001). A segunda é adotar critérios específicos para escolha dos pontos de recombinação, cuja forma mais difundida é padrão de escolha de nós função com 90% de probabilidade e os 10% restante em nós folha (KOZA, 1992a) ou ainda as técnicas de recombinação homóloga (LANGDON, 2000). A terceira é a adoção de algum sistema de avaliação do indivíduo que atue antes e depois da recombinação, como feito por O'Reilly e Oppacher (1995).

Essas três formas têm em comum o fato de considerarem primariamente questões sintáticas na recombinação genética. As abordagens mais recentes têm considerado também os aspectos semânticos dos indivíduos, utilizando-os de forma direta ou indireta. Neste Capítulo, apresentamos um novo método de recombinação que faz uso de sintáticas e semânticas. Como nas abordagens tradicionais, selecionamos aleatoriamente subárvores dos pais e as utilizamos como ramos que, combinados, formam os filhos. Acrescentamos, porém, uma constante de ponderação associada a cada ramo, de forma a indicar a importância de cada parte em relação ao todo, sendo possível, ainda, descartar ramos tidos como irrelevantes. O algoritmo contém também uma nova forma de aumentar a diversidade populacional.

3.1 Subárvores como regressores

Conforme é descrito por Montgomery e Runger (2010), modelos de regressão linear podem gerar superfícies cujas formas não são lineares (planos ou hiperplanos). De fato,

qualquer modelo de regressão que seja linear nos parâmetros (os β 's) é um modelo de regressão linear, independente da forma da superfície que ele gere. Um exemplo é a Equação (3.1), cujo gráfico é apresentado na Figura 10.

$$Y = 0,1 + 2x^2 + 7y \sin(x) + 5x \cos(y) \quad (3.1)$$

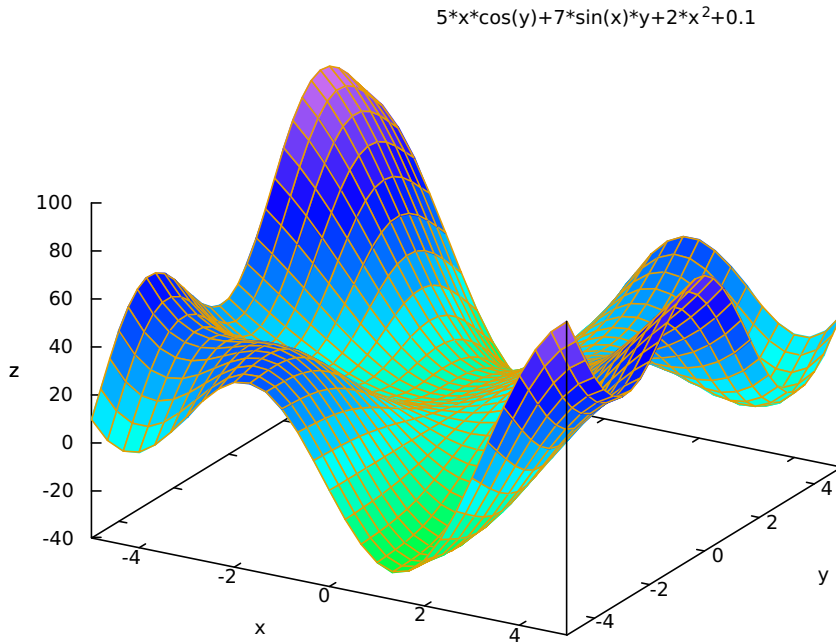


Figura 10 – Superfície gerada pela Equação (3.1), que é um modelo de regressão linear nos parâmetros β .

Em (3.1), os valores β_i e x_i são identificados da seguinte forma:

$$Y = \underbrace{0,1}_{\beta_0} + \underbrace{2}_{\beta_1} \underbrace{x^2}_{x_1} + \underbrace{7}_{\beta_2} \underbrace{y \sin(x)}_{x_2} + \underbrace{5}_{\beta_3} \underbrace{x \cos(y)}_{x_3}$$

Essa característica foi utilizada para a criação do operador de recombinação proposto, descrito e testado no presente trabalho, que trata cada regressor x_i como uma subárvore vinda dos pais no momento da recombinação, enquanto os valores β_i correspondem ao “peso” ou à “relevância” da subárvore na nova árvore gerada.

3.2 Operador MRX

As versões tradicionais de algoritmos de recombinação em árvores, como o apresentado Seção 2.2.5, criam novos indivíduos a partir da troca de subárvores em pontos aleatoriamente selecionados. Embora seja esperado que a árvore filha apresente qualidade

igual ou superior aos pais devido ao material genético herdado, na prática não há garantias quanto a isso. Estudos antigos, como O'Reilly e Oppacher (1994) e Nordin, Francone e Banzhaf (1995) já apontavam para os efeitos destrutivos do operador de recombinação, que podem levar à queda da média da qualidade da população ou mesmo ao efeito *bloat*.

Os recentes operadores baseados em semântica tentam acrescentar alguma informação sobre significado que cada árvore representa, com a finalidade de criar filhos de maior qualidade. Infelizmente, o efeito colateral de quase todas essas técnicas é o rápido crescimento do tamanho médio da população, levando também ao efeito *bloat* (NGUYEN et al., 2016).

É neste contexto que apresentamos o operador de recombinação com regressão linear múltipla, que denominamos MRX (*Multiple Regression Crossover*). Este operador atua sob subárvores distintas retiradas dos pais e constrói o filho unindo-os por meio de uma soma ponderada. As constantes reais utilizadas nesta ponderação são cuidadosamente escolhidas por meio da aplicação do método dos mínimos quadrados e correspondem aos coeficiente estimados, dados pela matriz $\hat{\beta}$, apresentada na Seção 2.4.1.1. Assim, a árvore gerada obedece à estrutura apresentada na Fórmula 3.2, onde os valores s_{a_i} e s_{b_i} representam subárvores de árvores a e b .

$$\hat{\beta}_0 + \hat{\beta}_1 s_{a_1} + \cdots + \hat{\beta}_n s_{a_n} + \hat{\beta}_{n+1} s_{b_1} + \cdots + \hat{\beta}_{2n} s_{b_n} \quad (3.2)$$

O Algoritmo 5 apresenta a descrição em alto nível do método proposto. Os detalhes de implementação, etapa por etapa, são fornecidos nas seções seguintes.

Algoritmo 5 Algoritmo Básico do MRX

Entrada: Duas árvores Arv_a e Arv_b e um *dataset* de treinamento.

Saída: Uma árvore filha com combinação otimizada de suas subárvores.

```

1:  $n \leftarrow \text{rand}(2, 6)$ 
2: Selecione  $n$  subárvores distintas de  $Arv_a$  e  $n$  subárvores distintas de  $Arv_b$ .
3: se não foi possível retirar pelo menos duas subárvores de cada árvore então
4:   Marque o modelo como inválido e finalize o algoritmo.
5: fim se
6: Crie a matriz do modelo  $\mathbf{X}$  utilizando o entradas do dataset submetidas às subárvore
7: se em uma coluna de  $\mathbf{X}$  houver pelo menos um valor inválido (NaN ou  $\infty$ ) então
8:   remova esta coluna de  $\mathbf{X}$ 
9: fim se
10: se todas as colunas foram retiradas pelo passo 8 então
11:   Marque o modelo como inválido e finalize o algoritmo.
12: fim se
13: Crie a matriz  $\mathbf{y}$  com as saídas do dataset
14: Encontre os coeficientes do modelo linear por meio do método dos quadrados mínimos
15: Execute o testes de hipóteses sobre os coeficientes e encontre a estatística  $t$  (com
    significância 0,01) para cada um
16: Retire todas as colunas de  $\mathbf{X}$  cujos coeficientes correspondentes são zero ou, de acordo
    com a estatística  $t$ , não diferentes de zero.
17: se todas as colunas foram retiradas então
18:   Marque o modelo como inválido e finalize o algoritmo.
19: fim se
20: se pelo menos uma coluna foi retirada então
21:   Aplique novamente o método dos quadrados mínimos na matriz modificada.
22: fim se
23: Crie o modelo linear
24: retorne a árvore associada ao modelo

```

3.2.1 Configuração da matriz do modelo

O MRX combina subárvores vindas de duas árvores pais para formar um novo indivíduo por meio da construção de um modelo linear. Cada subárvore s_{a_i} de um árvore a , assim como cada subárvore s_{b_i} de uma árvore b , será tratada como regressor de um modelo linear, como na Equação (2.14). Dessa forma, a determinação dos pesos β_i inicia-se com a construção a matriz \mathbf{X} do modelo.

3.2.1.1 Escolha das subárvores

Para que diferentes características dos pais sejam mapeadas nos modelos lineares, subárvores distintas devem ser aleatoriamente selecionadas. A quantidade de subárvores¹ que cada pai fornece para formar o filho é aleatoriamente definida, sendo no mínimo duas

¹ O parâmetro “quantidade de subárvores” pode refletir na qualidade e no tamanho das respostas geradas pelo MRX, conforme será discutido ao final do Capítulo 4.

e no máximo seis. Dessa forma, se em uma mesma geração um par de pais for selecionado mais de uma vez, a probabilidade de gerar filhos distintos é maior.

Embora qualquer nó não-terminal possa ser utilizado como raiz de uma subárvore (incluindo a raiz da árvore original), adotamos nesta seleção uma restrição relacionada o nível que o nó se encontra. Para inibir o aparecimento de subárvores excessivamente grandes, nós localizados nos níveis superiores não participam da seleção. Convencionamos assim que somente nós pertencentes à região correspondente à quarta parte superior da árvore não participam da seleção, como apresentado exemplificado na Figura 11. Esta restrição somente não é observada quando a profundidade da árvore é um valor menor ou igual quatro.

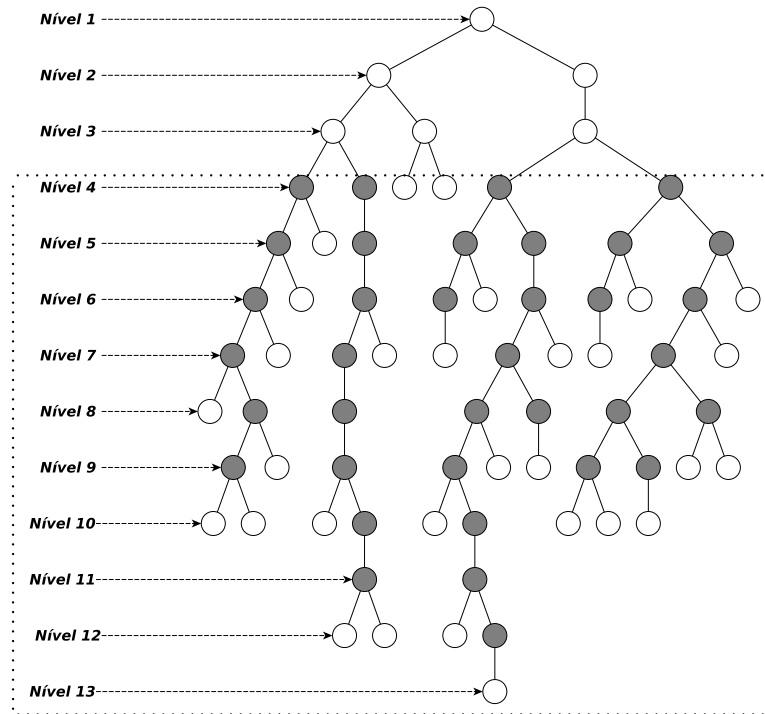


Figura 11 – Área de seleção de subárvores para o MRX. Os nós não terminais (preenchidos) localizados dentro da região tracejada (a partir do nível 4) são possíveis raízes das subárvores aleatoriamente selecionadas.

3.2.1.2 Criação da matriz

Para o preenchimento da matriz \mathbf{X} , cada subárvore é alimentada com cada entrada do *dataset* de treinamento. Assim, a matriz resultante é organizada de forma que cada uma das suas colunas representa uma subárvore s_{a_i} ou uma subárvore s_{b_i} e cada linha contenha o correspondente valor de saída de uma subárvore quando aplicada à entrada x_d do *dataset*. Se houver n subárvores de cada árvore e m entradas no *dataset*, representadas pelo vetor \mathbf{x}_i , então a matriz \mathbf{X} será configurada como apresentado em (3.3), onde a primeira coluna,

totalmente preenchida com 1's, servirá para encontrar o intercepto do modelo linear.

$$\mathbf{X} = \begin{bmatrix} 1 & s_{a_1}(\mathbf{x}_1) & \cdots & s_{a_n}(\mathbf{x}_1) & s_{b_1}(\mathbf{x}_1) & \cdots & s_{b_n}(\mathbf{x}_1) \\ 1 & s_{a_1}(\mathbf{x}_2) & \cdots & s_{a_n}(\mathbf{x}_2) & s_{b_1}(\mathbf{x}_2) & \cdots & s_{b_n}(\mathbf{x}_2) \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ 1 & s_{a_1}(\mathbf{x}_m) & \cdots & s_{a_n}(\mathbf{x}_m) & s_{b_1}(\mathbf{x}_m) & \cdots & s_{b_n}(\mathbf{x}_m) \end{bmatrix} \quad (3.3)$$

Estando a matriz completamente preenchida, verifica-se se alguma coluna contém um NaN ² ou um valor ∞ . Colunas que contenham qualquer um destes dois valores e em qualquer quantidade devem ser retiradas da matriz, com objetivo de inibir a proliferação de indivíduos incapazes de mapear todos os casos de treinamento.

3.2.2 Criação do modelo linear

Como a construção da matriz do modelo no MRX ocorre de forma semelhante aos processos executados na MRGP e na Programação *Kaizen*, não há garantias de se obter uma matriz não singular, conforme discutido na Seção 2.4.3. Por essa razão, o cálculo dos coeficientes de regressão linear é feito utilizando a pseudo inversa, com aplicação da Fórmula (2.34).

Resta ainda verificar o quão relevante é cada regressor no modelo linear criado. Os coeficientes da regressão indicam o quão pertinente é cada subárvore para o modelo criado. Assim, se um coeficiente não for considerado estatisticamente diferente de zero, utilizando os testes de hipóteses descritos na Seção 2.4.1.3, sua subárvore correspondente poderá ser eliminada. O mesmo procedimento é feito para os casos em que $\hat{\beta}_i$ for suficientemente próximo do *epsilon da máquina*³.

Os coeficientes $\hat{\beta}_i$ passam a fazer parte das subárvores correspondentes, sendo inseridas como nós constantes que multiplicam as raízes das subárvores. A seguir, todas as subárvores são unidas por nós função **Soma**. Dessa forma, a nova árvore criada pela junção dos genes dos pais são unidas ponderadamente, de acordo com os ajustes de uma regressão linear múltipla. À árvore criada, junta-se também, com a **Soma**, o coeficiente $\hat{\beta}_0$, que representa o intercepto.

3.2.3 Identificação de modelos válidos

Existe a possibilidade de o MRX, em seu processo de criação de árvores filhas como modelos lineares das subárvores, não completar sua execução. Isso pode ocorrer, por exemplo, se na etapa de criação da matriz \mathbf{X} todos as subárvores gerarem pelo menos

² *NaN*: Not a Number.

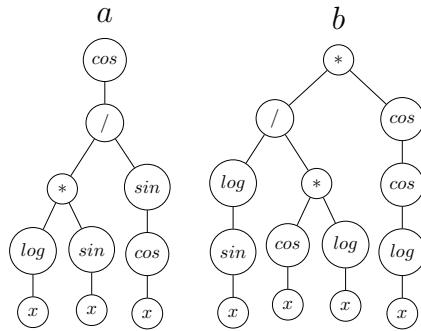
³ No contexto de operações numéricas de ponto flutuante, o *epsilon da máquina* é menor ϵ tal que $x + \epsilon = x$, $x \in \mathbb{R}$.

um valor ∞ ou NaN . Neste caso, não haverá elementos na matriz e o processo falha. Outro momento crítico é o teste dos coeficiente, momento em que é verificado que todos os valores são suficientemente próximos de zero ou estatisticamente iguais a zero. Neste caso, também, o modelo ficará sem regressores.

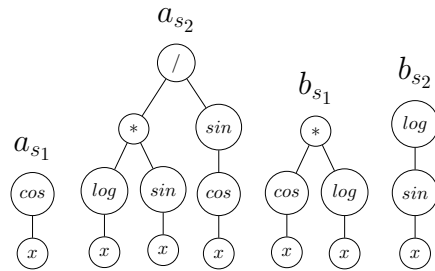
Para os casos em que o operador falha, adotamos a estratégia de selecionar novas subárvores e gerar novos modelos, repetindo este processo determinado número de vezes. Se ainda assim não for obtido algum modelo válido, um dos dois pais é copiado para a próxima geração.

3.2.4 Exemplo

O processo do algoritmo básico do MRX é exemplificado na Figura 12, onde a e b foram selecionadas (Figura 12a) para aplicação do operador. Selecionamos árvores pequenas para facilitar a visualização. Neste exemplo, duas subárvores de cada árvore são selecionadas (Figura 12b), submetidas ao conjunto de treinamento para preenchimento da matriz do modelo, os coeficientes de regressão são calculados por meio dos mínimos quadrados e com auxílio da pseudo inversa (Figura 12c) e os coeficientes são unidos às subárvores (Figura 12d) para formar as árvores (Figura 12d). Verificações e testes de hipóteses são realizados sobre os coeficientes e aqueles não diferentes de zero são retirados, para que o método dos mínimos quadrados possa ser novamente aplicado, criando a árvore filha (Figura 12f).



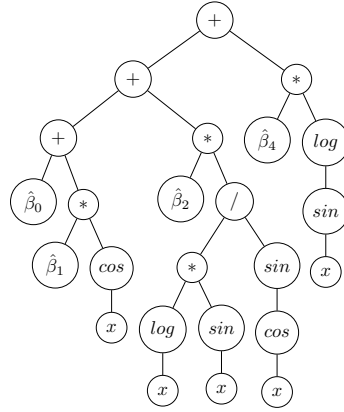
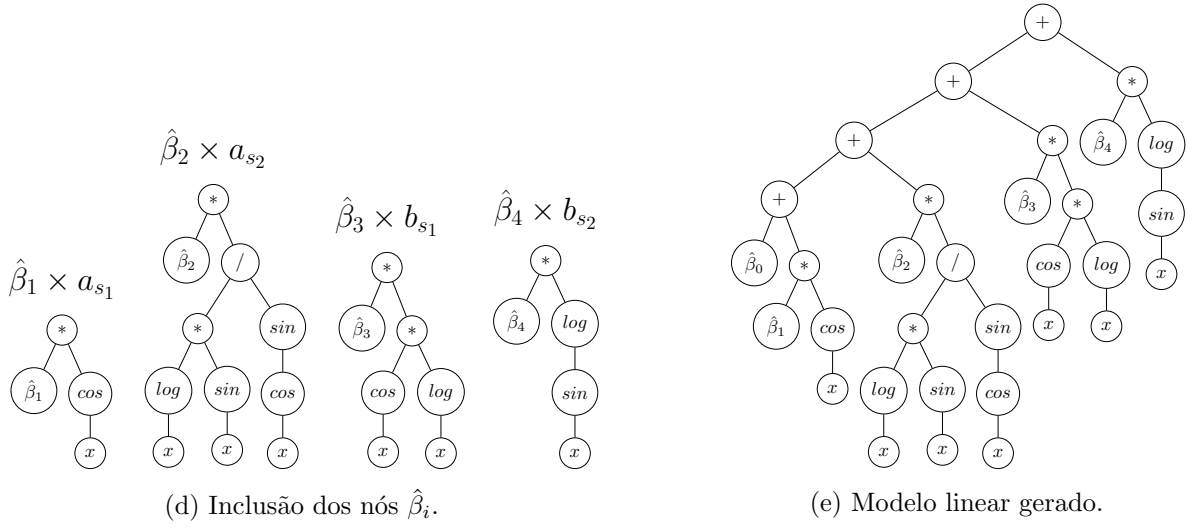
(a) Pais selecionados.



(b) Subárvores aleatoriamente selecionadas.

$$\begin{bmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \\ \hat{\beta}_2 \\ \hat{\beta}_3 \\ \hat{\beta}_4 \end{bmatrix} = \begin{bmatrix} 1 & a_{s1}(x_1) & a_{s2}(x_1) & b_{s1}(x_1) & b_{s1}(x_1) \\ 1 & a_{s1}(x_2) & a_{s2}(x_2) & b_{s1}(x_2) & b_{s1}(x_2) \\ 1 & a_{s1}(x_3) & a_{s2}(x_3) & b_{s1}(x_3) & b_{s1}(x_3) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & a_{s1}(x_n) & a_{s2}(x_n) & b_{s1}(x_n) & b_{s1}(x_n) \end{bmatrix}^{\dagger} \times \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix}$$

(c) $\hat{\beta} = X^{\dagger}y$: determinando os coeficientes da regressão.



(f) Filho gerado, supondo que o teste de hipóteses indicou que $\hat{\beta}_3$ não é estatisticamente diferente de zero. Considere que os valores $\hat{\beta}_i$ foram reavaliados pelo método dos mínimos quadrados após a exclusão de $\hat{\beta}_3$.

Figura 12 – Exemplo com o passo a passo do operador MRX.

3.3 Melhorando a diversidade populacional

Em Computação Evolucionária, entende-se diversidade como a diferença (estrutural e/ou comportamental) de indivíduos em uma população. Este conceito motiva muitas pesquisas em PG, já que a diversidade populacional atua na fase de exploração, tendendo a impedir que a população convirja prematuramente para ótimos locais (NGUYEN; NGUYEN, 2006). Como descrito por Burke, Gustafson e Kendall (2004), é comum que ocorra a perda da diversidade já nas primeiras gerações da PG. Para aumentar a diversidade no MRX, acrescentamos ao conjunto de subárvores selecionados dos pais algumas subárvores geradas aleatoriamente.

Como descrito na Seção 3.2.1.1, são selecionadas entre duas e seis subárvores de cada pai envolvido na recombinação. Assim, após escolhidas as subárvores do dois pais, escolhe-se também uma quantidade aleatória (entre 2 e 6) de subárvores aleatoriamente

geradas pelo método *grow* (Seção 2.2.2), com profundidades variando entre os valores mínimo e máximo das subárvores já selecionadas dos pais.

Esse método de acréscimo de diversidade faz do MRX um operador híbrido, capaz de criar novos indivíduos por recombinação e por mutação ao mesmo tempo. A vantagem em relação à mutação tradicional é que o componente só permanecerá no filho se for relevante. Outra consequência direta é não ser mais necessário determinar a priori as taxas de recombinação e mutação: o MRX é aplicado em 100% das vezes e a taxa efetiva de mutação é autoajustável para cada aplicação do operador.

3.4 Controle de *bloat* pelo MRX

A pesquisa que resultou na criação do operador MRX foi inicialmente direcionada à busca por um método de controle de *bloat*. Conforme será discutido em capítulos posteriores, onde resultados experimentais são apresentados, obtivemos de fato sucesso no controle no crescimento do tamanho médio das árvores geradas. Porém, ao analisar os métodos tradicionais de tratamento e/ou prevenção do *bloat*⁴, observamos que o MRX não se enquadra perfeitamente em nenhum deles.

O operador que propomos tem como principal objetivo gerar modelos de alta qualidade em que os genes selecionados dos pais são combinados com o melhor ajuste possível com referência às teorias de regressão linear múltipla. O controlado tamanho das árvores que acabam constituindo a população é um agradável efeito colateral. De fato, quando analisamos algumas teorias relacionadas à causa do *bloat*, como as discutidas na Seção 2.3, observamos que o MRX trata naturalmente cada uma delas.

Os efeitos da teoria do *hitchhiking* tendem a ser amenizados, visto que a qualidade de cada subárvore envolvida na recombinação é avaliada. Dessa forma, trechos que não representam efeitos significativos sobre a avaliação da árvore têm maiores chances de serem descartados. A decisão de escolher não apenas uma, mas várias subárvores de cada pai também auxilia na prevenção do *hitchhiking*, já que diversos trechos potencialmente “bons” e “ruins” podem ser testados.

A teoria da defesa contra a recombinação (BLICKLE; THIELE, 1994; MCPHEE; MILLER, 1995) também não parece se aplicar ao MRX, pois filhos que representam variações neutras obtidas pela introdução de *introns* não costumam ser gerados, já que os *introns* são trechos prováveis de serem retirados via testes de hipóteses. Além disso, a recombinação MRX representa mais um benefício do que uma ameaça aos descendentes criados, dado seu potencial de calibração entre os ramos.

A medida de qualidade das subárvores utilizada pelo MRX previne também os

⁴ O Apêndice A lista uma série de estratégias de controle de *bloat* popularmente conhecidas.

efeitos do viés de remoção, que é fundamentado na existência de nós viáveis e nós inviáveis (SOULE; FOSTER, 1998). Como os coeficientes da regressão determinam quais nós (raízes de subárvores) permanecem no modelo, quando um nó considerado inviável for selecionado ele tenderá a ser descartado, seja pelo valor do coeficiente, seja pela aplicação dos testes de hipóteses. Assim, por mais que um nó inviável seja escolhido, ele dificilmente figurará na resposta final.

A teoria da profundidade do ponto de cruzamento (IGEL; CHELLAPILLA, 1999) baseia-se no fato de nós mais profundos, quando operados pela recombinação tradicional, causarem pouco ou nenhum impacto na avaliação da árvore, podendo gerar variações neutras pela inserção de *introns*. Para o MRX, a profundidade do nó é fator irrelevante: qualquer nó, encontrando-se a qualquer profundidade, é tratado de forma igual pelo algoritmo de recombinação.

Por fim, mesmo representando uma técnica de busca que utiliza avaliação estática e com representação discreta de tamanho variável, como descrito por Langdon (1998), a recombinação MRX apresenta robustez em seus resultados. Ainda que indivíduos de representação longa sejam selecionados em detrimento a outros, de representação mais compacta, como descrito na Seção 2.3.2, seus descendentes não terão, necessariamente, representação longa. Dessa forma, no MRX a aptidão não pode ser considerada uma razão para a causa do *bloat*, diferente do que se observa em versões tradicionais da PG, como descrito por Langdon e Poli (1998a).

3.5 Considerações sobre o capítulo

Mesmo sendo um dos operadores mais antigos e mais utilizados na Programação Genética, os mecanismos que levam a recombinação a direcionar a busca rumo à solução ainda não são completamente compreendidos. Um esforço feito no sentido de elucidar esta questão foi realizado por O'Reilly (1995), que adaptou a Teoria dos Esquemas dos Algoritmos Genéticos ao contexto da PG. Mesmo sendo um avanço para a época, demonstrações matemáticas já indicaram que esta teoria não é completa em fornecer tal entendimento (De Jong, 2006).

Em uma análise empírica, McPhee, Ohs e Hutchison (2007) realizaram o estudo da semântica de cada subárvore presente nos pais utilizando um mecanismo baseado em blocos de construção semânticos. Em um problema de domínio booleano, eles enumeraram cada possível saída de cada subárvore presente no indivíduo, chegando à conclusão de que mais de 75% das recombinações não causam efeitos imediatos na busca por solução.

É com o intuito de proporcionar recombinações com alguma garantia de efeito prático que propomos o operador MRX. Uma vez que os pesos dos ramos do filho gerado são obtidos por meio do método dos mínimos quadrados, há de se esperar que o modelo

formado seja o mais adequado possível e sem perder a aleatoriedade (no momento da escolha das subárvores dos pais) pressuposta a um método de Computação Evolucionária. Além disso, a aleatoriedade proporcionada pelo acréscimo de novas subárvores ao modelo pode direcionar a busca para regiões que não seriam convencionalmente exploradas.

O algoritmo do MRX possibilita configurações variadas, dado que alguns parâmetros podem ser modificados. A escolha da quantidade e da região permitida para seleção das subárvores retiradas dos pais foram determinadas de maneira empírica neste trabalho. Assim, outros valores podem ser adotados em outras implementações, caso se constate melhorias nos resultados. Outra decisão que pode ser tomada por quem desejar utilizar o operador é o método de regressão linear adotado. Utilizamos aqui o método dos mínimos quadrados implementado com a pseudo inversa para determinar os coeficientes de regressão, mas outros algoritmos, como o Lasso, utilizado pela MRGP, podem também ser aplicados.

Este capítulo apresentou o MRX e discutiu suas características de forma teórica. No capítulo seguinte apresentaremos resultados de experimentos que compararam nosso operador com operadores já conhecidos e também a outras técnicas de CE também baseadas em Análise de Regressão.

Resultados e discussões

Variados testes foram executados para analisar as características principais do operador MRX. Foram realizadas comparações com a versão canônica da PG, com implementações que utilizam operadores semânticos e com outras técnicas baseadas em Regressão Linear Múltipla. Este capítulo apresenta e discute os resultados obtidos. Todas as comparações realizadas utilizaram a versão do MRX que acrescenta subárvores aleatórias aos novos modelos criados, conforme foi discutido na Seção 3.3. Assim, antes de apresentar os resultados comparando as diferentes técnicas, reportamos os experimentos feitos para verificar o quão efetivo é esse acréscimo de aleatoriedade.

4.1 Preparação dos experimentos

Para realização e validação dos testes desenvolvidos ao longo deste capítulo, três itens fazem-se necessários: 1) a implementação do método em alguma linguagem de programação; 2) um conjunto de dados para ser usado como *benchmark*; 3) uma metodologia de teste pré-definida. Tais itens são descritos nesta seção.

4.1.1 Implementação e ambiente de teste

Para realização dos experimentos, desenvolvemos uma aplicação de Programação Genética, denominada GP, que teve como base os conceitos canônicos apresentados na Seção 2.2. Pelo processo de herança em Programação Orientada a Objetos, estendemos a GP para criar outra aplicação denominada GP-MRX, que sobrescreve o método de recombinação da GP, substituindo-o pelo MRX, e também o método de mutação, tornando-o sem efeito.

O programa foi desenvolvido em linguagem Java, versão 1.8.0_201. Além das bibliotecas padrão da linguagem, foi utilizada também a *Commons Math*¹ para o processamento

¹ Disponível em <<http://commons.apache.org/proper/commons-math/>>

eficiente de matrizes e, principalmente, na determinação da pseudo inversa. Os experimentos foram realizados com auxílio da interface gráfica apresentada na Figura 13, que permite gravar em disco, no formato de arquivo texto, grande quantidade de resultados obtidos.



Figura 13 – Interface gráfica utilizada na realização dos experimentos.

O processamento dos dados gerados pelos experimentos foi feito utilizando a linguagem Python. Além das bibliotecas padrão da linguagem, utilizamos também a *Matplotlib*² para confecção dos gráficos e a *SciPy*³, para auxílio na realização das validações estatísticas.

Os *datasets* utilizados para a realização dos experimentos são descritos logo adiante, na Seção 4.1.2. Todas as execuções foram realizadas em um computador com as seguintes configurações:

- Sistema operacional: *Linux Mint 18.3 Cinnamon 64 bits*;
- Kernel *Linux: 4.15.0-43-generic*;
- Processador: *Intel i7-4770*, 3.40GHz – quatro núcleos físicos e quatro núcleos lógicos;
- Memória: 16 GB DDR3.

4.1.2 O Benchmark

McDermott et al. (2012) realizaram detalhada revisão da literatura especializada, discutindo sobre os principais *benchmarks* normalmente utilizados em trabalhos de Programação Genética. Como resultado, delimitou-se grupos de funções (*datasets*) que têm sido

² Disponível em: <<https://matplotlib.org/index.html>>

³ Disponível em: <<https://www.scipy.org/>>

amplamente utilizados por variados autores nos últimos anos. Dos *benchmarks* organizados, selecionamos alguns que são constituídos por funções polinomiais, trigonométricas, logarítmicas e exponenciais: Koza, Nguyen, Keijzer e Vladislavleva, apresentadas nas tabelas 5, 6, 7 e 8, respectivamente. Nessas tabelas, as seguintes notações são adotadas:

- $U[a, b, c]$: c valores aleatórios com distribuição uniforme são escolhidos no intervalo fechado de a a b ;
- $E[a, b, c]$: consiste em uma grade de pontos igualmente espaçados (para cada variável), iniciando em a e finalizando em b e com distância c entre eles. .

Tabela 5 – *Benchmarks* para regressão simbólica: Funções Koza

Nome	Função Objetivo	Treinamento	Teste
Koza-1	$x^4 + x^3 + x^2 + x$	$U[-1, 1, 20]$	$U[-1, 1, 20]$
Koza-2	$x^5 - 2x^3 + x$	$U[-1, 1, 20]$	$U[-1, 1, 20]$
Koza-3	$x^6 - 2x^4 + x^2$	$U[-1, 1, 20]$	$U[-1, 1, 20]$

Fonte: Adaptado de McDermott et al. (2012)

Tabela 6 – *Benchmarks* para regressão simbólica: Funções Nguyen

Nome	Função Objetivo	Treinamento	Teste
Nguyen-1	$x^3 + x^2 + x$	$U[-1, 1, 20]$	$U[-1, 1, 20]$
Nguyen-3	$x^5 + x^4 + x^3 + x + 2 + x$	$U[-1, 1, 20]$	$U[-1, 1, 20]$
Nguyen-4	$x^6 + x^5 + x^4 + x^3 + x + 2 + x$	$U[-1, 1, 20]$	$U[-1, 1, 20]$
Nguyen-5	$\sin(x^3) \cos(x) - 1$	$U[-1, 1, 20]$	$U[-1, 1, 20]$
Nguyen-6	$\sin(x) + \sin(x + x^2)$	$U[-1, 1, 20]$	$U[-1, 1, 20]$
Nguyen-7	$\ln(x + 1) + \ln(x^2 + 1)$	$U[0, 2, 20]$	$U[0, 2, 20]$
Nguyen-8	\sqrt{x}	$U[0, 4, 20]$	$U[0, 4, 20]$
Nguyen-9	$\sin(x) + \sin(y^2)$	$U[-1, 1, 100]$	$U[-1, 1, 100]$
Nguyen-10	$2 \sin(x) \cos(y)$	$U[-1, 1, 100]$	$U[-1, 1, 100]$

Fonte: Adaptado de McDermott et al. (2012)

Para cada função é definido um domínio para treinamento, empregado na evolução, e outro para teste, que é utilizado para medir a capacidade que um método tem de generalizar resultados, ou seja, o quão capaz ele é de fornecer saídas corretas para entradas que não constaram em seu treinamento. Originalmente, não é definido o domínio de teste para as funções Koza e Nguyen, mas para nossos experimentos criamos esses conjuntos com valores diferentes (mas dentro do mesmo intervalo) dos empregados nos conjuntos de treinamento.

A Tabela 9 indica os conjuntos de funções e constantes utilizados como genes em cada *benchmark*. As constantes para os grupo Koza e Nguyen são tradicionalmente opcionais. O símbolo % representa a operação divisão protegida. As “constantes” presentes no grupo Vladislavleva são na verdade funções com argumento, onde ϵ é um valor definido no início da execução, sendo aleatório e uniformemente distribuído dentro do intervalo

Tabela 7 – *Benchmarks* para regressão simbólica: Funções Keijzer

Nome	Função Objetivo	Treinamento	Teste
Keijzer-1	$0.3x \sin(2\pi x)$	E[-1, 1, 0.1]	E[-1, 1, 0.001]
Keijzer-2	$0.3x \sin(2\pi x)$	E[-2, 2, 0.1]	E[-2, 2, 0.001]
Keijzer-3	$0.3x \sin(2\pi x)$	E[-3, 3, 0.1]	E[-3, 3, 0.001]
Keijzer-4	$x^3 e^{-x} \cos(x) \sin(x) (\sin^2(x) \cos(x) - 1)$	E[0, 10, 0.05]	E[0.05, 100.5, 0.05]
Keijzer-5	$\frac{30xz}{(x-10)y^2}$	x, z : U[-1, 1, 1000] y : U[1, 2, 1000]	x, z : U[-1, 1, 10000] y : U[1, 2, 10000]
Keijzer-6	$\sum_{i=1}^x \frac{1}{i}$	E[1, 50, 1]	E[1, 120, 1]
Keijzer-7	$\ln(x)$	E[1, 100, 1]	E[1, 100, 0.1]
Keijzer-8	\sqrt{x}	E[0, 100, 1]	E[1, 100, 0.1]
Keijzer-9	$\ln(x + \sqrt{x^2 + 1})$	E[0, 100, 1]	E[1, 100, 0.1]
Keijzer-10	x^y	U[0, 1, 100]	E[0, 1, 0.01]
Keijzer-11	$xy + \sin((x-1)(y-1))$	U[-3, 3, 20]	E[-3, 3, 0.01]
Keijzer-12	$x^4 + x^3 + \frac{y^2}{2} - y$	U[-3, 3, 20]	E[-3, 3, 0.01]
Keijzer-13	$6 \sin(x) \cos(y)$	U[-3, 3, 20]	E[-3, 3, 0.01]
Keijzer-14	$\frac{8}{2+x^2+y^2}$	U[-3, 3, 20]	E[-3, 3, 0.01]
Keijzer-15	$\frac{x^3}{5} + \frac{y^4}{2} - y - x$	U[-3, 3, 20]	E[-3, 3, 0.01]

Fonte: Adaptado de McDermott et al. (2012)

Tabela 8 – *Benchmarks* para regressão simbólica: Funções Vladislavleva

Nome	Função Objetivo	Treinamento	Teste
Vladislavleva-1	$\frac{e^{-(x-1)^2}}{1.2+(y-2.5)^2}$	U[0.3, 4, 100]	E[-0.2, 4.2, 0.1]
Vladislavleva-2	$e^{-x} x^3 (\cos x \sin x) (\cos x \sin^2 x - 1)$	E[0.05, 10, 0.1]	E[.0.5, 10.5, 0.05]
Vladislavleva-3	$e^{-x} x^3 (\cos x \sin x) (\cos x \sin^2 x - 1)(y - 5)$	x : E[0.05, 10, 0.1] y : E[0.05, 10.05, 2]	x : E[-0.5, 10.5, 0.05] y : E[-0.5, 10.5, 0.5]
Vladislavleva-4	$\frac{10}{5+(x-3)^2+(y-3)^2+(z-3)^2+(v-3)^2+(w-3)^2}$	U[0.05, 6.05, 1024]	U[-0.25, 6.35, 5000]
Vladislavleva-5	$30 \frac{(x-1)(z-1)}{y^2(x-10)}$	x : U[0.05, 2, 300] y : U[1, 2, 300] z : U[0.05, 2, 300]	x : E[-0.05, 2.1, 0.15] y : E[0.95, 2.05, 0.1] z : E[-0.05, 2.1, 0.15]
Vladislavleva-6	$6 \sin(x) \cos(y)$	U[0.1, 5.9, 30]	E[-0.05, 6.05, 0.02]
Vladislavleva-7	$(x-3)(y-3) + 2 \sin((x-4)(y-4))$	U[0.05, 6.05, 300]	U[-0.25, 6.05, 1000]
Vladislavleva-8	$\frac{(x-3)^4+(y-3)^3-(y-3)}{(y-2)^4+10}$	U[0.05, 6.05, 50]	E[-0.25, 6.35, 0.2]

Fonte: Adaptado de McDermott et al. (2012)

$[-5, 5]$. Em Keijzer, as constantes são aleatoriamente selecionadas de uma distribuição normal com média zero e desvio-padrão 5. Os nomes adotados para as variáveis são, em ordem, são: x, y, z .

Tabela 9 – Conjuntos de funções e constantes utilizados por cada *benchmark*.

Nome	Funções	Constantes
Koza	$+, -, \%, \times, \sin, \cos, e^x, \ln(x)$	$[-1, 1]$
Nguyen	$+, -, \%, \times, \sin, \cos, e^x, \ln(x)$	$[-1, 1]$
Keijzer	$+, \times, \frac{1}{x}, -x, \sqrt{x}$	Valores aleatórios do intervalo $N(\mu = 0, \sigma = 5)$
Vladislavleva 4, 5 e 8	$+, -, \times, \%, x^2$	$x^\epsilon, x + \epsilon, x\epsilon$
Vladislavleva 1 e 6	$+, -, \times, \%, x^2, e^x, e^{-x}$	$x^\epsilon, x + \epsilon, x\epsilon$
Vladislavleva 2, 3 e 7	$+, -, \times, \%, x^2, e^x, e^{-x}, \sin, \cos$	$x^\epsilon, x + \epsilon, x\epsilon$

Fonte: Adaptado de McDermott et al. (2012)

4.1.3 Metodologia de teste validação dos resultados

Os *datasets* apresentados na Seção 4.1.2 são utilizados nas análises experimentais apresentadas no decorrer deste capítulo. Realizamos experimentos comparando a GP-MRX com:

- a implementação GP, apresentada na seção anterior, que utiliza operador sintático “recombinação de subárvore”, discutido na Seção 2.2.5.1;
- implementações que utilizam os operadores semânticos AGX, SSGX e RDO, discutidos na Seção 2.2.5.2;
- as versões de CE baseadas em Regressão Linear Múltipla MRGP e KP, discutidas nas seções 2.4.2.1 e 2.4.2.2, respectivamente.

Existem diferentes formas de comparar os resultados obtidos por versões distintas de Programação Genética. Neste trabalho, optamos por focar a maioria de nossas análises em dois critérios principais:

- A qualidade das respostas, mensurada para três critérios: 1) o erro apresentado pelo melhor indivíduo ao final da execução; 2) o erro observado no melhor indivíduo quando este é submetido ao conjunto de teste; 3) o tamanho (quantidade de nós) que o melhor indivíduo apresenta;
- Comportamento do *bloat* observado ao longo das gerações, considerado a evolução do tamanho médio de toda a população e a média de erro do melhor indivíduo.

4.1.3.1 Procedimentos para avaliação da qualidade

Optamos por utilizar, sempre que possível, a versão do programa desenvolvido pelo próprio autor da técnica em estudo, como é o caso da MRGP⁴ e do SSGX⁵, sendo que o último forneceu também a implementação do AGX e do RDO. Como o autor não encontrou implementação disponível para a KP, executamos experimentos utilizando o mesmo *benchmark* utilizado em De Melo (2014) e apresentamos a comparação entre os resultados obtidos pela GP-MRX e os resultados reportados no artigo. Essas implementações definem também qual métrica de erro é adotada. A MRGP utiliza a MSE, a PK baseia-se na RMSE, enquanto as demais análises utilizam a SAE. Dois aspectos chave foram analisados: a acurácia do modelo e a evolução do *bloat* ao longo das gerações

⁴ Disponibilizado por Arnaldo, Krawiec e O’Reilly (2014) em <<https://flexgp.github.io/gp-learners/mrgp.html>>.

⁵ Em <<https://github.com/jmmcd/GP-SSGX>> estão disponíveis os códigos que geraram os resultados apresentados em Nguyen et al. (2016).

Colhemos dados gerados por 50 execuções de cada versão de CE analisada. Esses dados referem-se ao melhor indivíduo de cada rodada. Execuções diferentes foram realizadas para *benchmarks* diferentes, sendo possível assim obter mais informações sobre o comportamento de cada método. De cada conjunto de 50 elementos, determinamos sua mediana, que adotamos, para estes casos, como medida de tendência central de qualidade. A escolha dessa estatística justifica-se por não haver garantias quanto à normalidade da amostra e também pelo fato de a mediana ser menos suscetível a *outliers* (SILVA; DIGNUM; VANNESCHI, 2012).

Os resultados referentes à qualidade foram também submetidos a testes de significância estatística. Realizamos comparação em pares, avaliando a diferença entre a utilização do MRX e cada uma das demais técnicas. Em termos estatísticos, estamos testando amostras pareadas, já que os grupos de comparação são gerados por dois algoritmos que foram aplicados a um mesmo grupo de problemas.

Ainda pelo fato de não haver garantias quanto à normalidade da amostra, foi utilizado o teste dos postos sinalizados de Wilcoxon para validação dos resultados, um método não paramétrico bastante utilizado em pesquisas que envolvem comparação entre pares para PG⁶ e apontado por Derrac et al. (2011) como adequado a essa categoria de experimentos. Por padrão, adotamos significância $\alpha = 0.01$.

O teste de Wilcoxon é útil para informar se duas distribuições são estatisticamente diferentes, dentro do nível de significância adotado. Para decidir se uma versão de PG é melhor que outra, adotamos o mesmo procedimento utilizado em Sotto e Melo (2016): sempre que o teste indicar que dois métodos são estatisticamente distintos, será considerado melhor aquele que fornecer o menor valor de mediana.

A dispersão dos dados analisados na amostra (sejam eles erro no treinamento, erro no teste ou quantidade de nós) é descrita por meio de gráficos de caixa (*boxplots*), como o da Figura 14. O intervalo correspondente à “caixa” contém 50% dos dados, sendo delimitado por três linhas: a inferior, que indica a posição do primeiro quartil, a linha interior, que indica a posição do segundo quartil (a mediana) e a superior, correspondente ao terceiro quartil. O ponto que repousa no interior da caixa indica a posição da média aritmética, enquanto as barras horizontais das partes inferior e superior correspondem às posições de mínimo e máximo, respectivamente. A Figura 14 possui também três “cruzes” na parte superior da barra de valor máximo, que é a forma de representar valores discrepantes da amostra, os *outliers*.

⁶ Como exemplo de pesquisas recentes que utilizaram Wilcoxon em trabalhos com PG, podemos citar Dou e Rockett (2018), Mei e Zhang (2018), Liskowski, Bladek e Krawiec (2018), Griffiths e Ekárt (2017), Chen, Zhang e Xue (2017), Castelli et al. (2017), Melo e Banzhaf (2017), dentre outros.

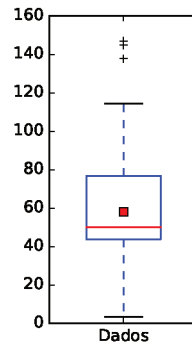


Figura 14 – Gráfico de caixa (*boxplot*) para visualização da dispersão dos dados.

4.1.3.2 Procedimentos para avaliação do comportamento do *bloat*

Realizamos análises sobre a tendência de crescimento do tamanho médio dos indivíduos da população na execução da GP-MRX e comparamos com informações fornecidas pelas execuções da GP e das PGs com operadores semânticos. Apresentamos os resultados através de gráficos, que trazem os valores do tamanho médio da população (a média de 50 execuções) ao longo das gerações. São fornecidos também os gráficos com média do erro dos melhores indivíduos ao longo dessas execuções.

Optamos por não acrescentar qualquer método de controle de *bloat* nas implementações. Esta decisão justifica-se pelo fato de querermos investigar a tendência característica de crescimento no número de nós em de cada versão executada. Porém, conforme será oportunamente relatado, houve casos em que foi necessário impor limites à profundidade das árvores geradas, caso contrário o consumo de memória física impediria a conclusão dos experimentos.

Como cada função é executada 50 vezes, sempre com população de 500 indivíduos, os valores de tamanho médio apresentados correspondem à média de 25000 árvores, geração a geração. Já o erro médio é a média dos erros dos melhores indivíduos, geração a geração, nas 50 execuções. Todos os resultados são apresentados graficamente.

4.2 Acréscimo de subárvores aleatórias ao modelo

Na Seção 3.2 apresentamos o operador MRX em sua forma básica, a qual foi modificada logo adiante, na Seção 3.3, de forma de aumentar a diversidade dos modelos criados por meio do acréscimo de subárvores aleatórias. Antes de comparar a GP-MRX com outras abordagens de Computação Evolucionária, apresentaremos nesta seção os resultados de experimentos realizados para avaliar o quão benéfico é este procedimento.

4.2.1 Configuração dos experimentos

Os dados utilizados nos experimentos provêm de duas implementações da GP-MRX: uma utilizando o operador com o acréscimo das subárvores aleatórias, que foi denominado MRX-Rand, e outra com um operador sem esse procedimento, denominado MRX-Simples. Com dados de erro (SAE) colhidos em 50 execuções de cada versão, aplicados aos *benchmarks* Koza, Nguyen, Keijzer e Vladislavleva, avaliamos a qualidade do melhor indivíduo obtido ao final da evolução e a reação ao efeito *bloat* de cada versão.

Aos resultados obtidos, aplicamos o teste de Wilcoxon, com significância de 1%, e relatamos o valor-p encontrado. Para explicitar a relação estatística do erro fornecido pela versão que utiliza o operador MRX-Simples, quando comparada à versão que utiliza o MRX-Rand, acrescentamos às tabelas informações no formato de sinais, onde “=” indica que os valores fornecidos pela versão com MRX-Rand e com MRX-Simples são estatisticamente iguais, “<” indica que valor fornecido pela versão com MRX-Simples é estatisticamente menor e “>” indica que o valor fornecido pela versão com MRX-Simples é estatisticamente maior. Os experimentos apresentados foram configurados com os seguintes parâmetros:

- Tamanho da população: 500;
- Taxa de recombinação: 90% para a versão com o operador MRX-Simples e 100% para a versão com MRX-Rand;
- Taxa de mutação: 5% para a versão com MRX-Simples e 0% para a versão com MRX-Rand;
- Quantidade de gerações: 51;
- Torneio: 5;
- Profundidade inicial: 6;
- Funções e constantes: adaptadas a cada *dataset*, de acordo com os dados da Tabela 9.

Gráficos de caixa (*boxplots*) são utilizados para analisar a dispersão gerada por cada versão. Suas legendas indicam MRX-R para a versão com subárvores aleatórias e MRX-S para a versão simples. O efeito *bloat* verificado em cada versão foi também analisado.

4.2.2 Treinamento

Os resultados obtidos pelo treinamento das duas versões do MRX, exibidos nas tabelas 10, 11, 12 e 13, proporcionam claras informações sobre a melhoria proporcionada pelo acréscimo de subárvores aleatórias. Há significativa melhoria (caracterizada pela queda

do erro) em todas as trinta e cinco funções analisadas. Esses resultados são amparados pelos testes de significância estatística, que apresentou valores p na ordem de 10^{-10} .

Tabela 10 – Treinamento usando funções do *benchmark* Koza: efeito do acréscimo de subárvores aleatórias.

Função	Estatística	MRX-Rand	MRX-Simples
Koza-1	Mediana	1.385e-08	1.451e-02
	Valor-P	–	7.557e-10 >
Koza-2	Mediana	3.018e-09	1.408e-01
	Valor-P	–	7.557e-10 >
Koza-3	Mediana	2.356e-08	1.985e-01
	Valor-P	–	7.557e-10 >

Tabela 11 – Treinamento usando funções do *benchmark* Nguenyn: efeito do acréscimo de subárvores aleatórias.

Função	Estatística	MRX-Rand	MRX-Simples
Nguyen-1	Mediana	8.425e-12	1.435e-03
	Valor-P	–	7.557e-10 >
Nguyen-3	Mediana	1.090e-06	3.212e-01
	Valor-P	–	7.557e-10 >
Nguyen-4	Mediana	5.138e-06	6.748e-02
	Valor-P	–	7.557e-10 >
Nguyen-5	Mediana	4.420e-07	3.211e+00
	Valor-P	–	7.557e-10 >
Nguyen-6	Mediana	3.870e-05	2.678e-02
	Valor-P	–	7.557e-10 >
Nguyen-7	Mediana	5.421e-08	5.315e-04
	Valor-P	–	7.557e-10 >
Nguyen-8	Mediana	1.127e-06	6.770e-03
	Valor-P	–	7.557e-10 >
Nguyen-9	Mediana	9.212e-06	1.468e-01
	Valor-P	–	7.557e-10 >
Nguyen-10	Mediana	4.064e-01	1.048e+01
	Valor-P	–	7.557e-10 >

Tabela 12 – Treinamento usando funções do *benchmark* Keijzer: efeito do acréscimo de subárvores aleatórias.

Função	Estatística	MRX-Rand	MRX-Simples
Keijzer-1	Mediana	5.880e-02	2.028e+00
	Valor-P	–	7.557e-10 >
Keijzer-2	Mediana	4.321e+00	7.867e+00
	Valor-P	–	7.557e-10 >
Keijzer-3	Mediana	1.356e+01	1.683e+01
	Valor-P	–	7.557e-10 >
Keijzer-4	Mediana	1.895e+01	4.104e+01
	Valor-P	–	7.557e-10 >
Keijzer-5	Mediana	3.315e+01	1.221e+02
	Valor-P	–	7.557e-10 >
Keijzer-6	Mediana	3.934e-06	1.959e-01
	Valor-P	–	7.557e-10 >
Keijzer-7	Mediana	2.520e-05	7.646e-01
	Valor-P	–	7.557e-10 >
Keijzer-8	Mediana	2.294e-11	2.321e-11
	Valor-P	–	8.031e-10 >
Keijzer-9	Mediana	1.075e-04	8.683e-01
	Valor-P	–	7.557e-10 >
Keijzer-10	Mediana	9.966e-01	2.053e+00
	Valor-P	–	7.557e-10 >
Keijzer-11	Mediana	6.566e+00	1.003e+01
	Valor-P	–	1.087e-09 >
Keijzer-12	Mediana	7.634e+00	5.456e+01
	Valor-P	–	7.557e-10 >
Keijzer-13	Mediana	2.000e+01	4.533e+01
	Valor-P	–	7.557e-10 >
Keijzer-14	Mediana	5.996e-01	2.485e+00
	Valor-P	–	7.557e-10 >
Keijzer-15	Mediana	1.547e+00	1.547e+01
	Valor-P	–	7.557e-10 >

Os gráficos de caixa apresentados nas figuras 15, 16, 17 e 18 indicam que o acréscimo de árvores aleatórias proporciona menor dispersão para o MRX. Além disso, nos casos em que trabalhou-se exclusivamente com subárvores vindas dos pais foram verificados mais *outliers* do que nos outros casos (18, contra 10).

Tabela 13 – Treinamento usando funções do *benchmark* Vladislavleva: efeito do acréscimo de subárvores aleatórias.

Função	Estatística	MRX-Rand	MRX-Simples
Vladislavleva-1	Mediana	3.722e+00	6.243e+00
	Valor-P	–	7.557e-10 >
Vladislavleva-2	Mediana	3.404e+00	2.038e+01
	Valor-P	–	7.557e-10 >
Vladislavleva-3	Mediana	3.139e+02	4.029e+02
	Valor-P	–	7.557e-10 >
Vladislavleva-4	Mediana	5.705e+01	1.053e+02
	Valor-P	–	7.557e-10 >
Vladislavleva-5	Mediana	1.716e+01	4.162e+01
	Valor-P	–	7.557e-10 >
Vladislavleva-6	Mediana	2.116e+01	7.413e+01
	Valor-P	–	7.557e-10 >
Vladislavleva-7	Mediana	2.891e+02	7.567e+02
	Valor-P	–	7.557e-10 >
Vladislavleva-8	Mediana	1.124e+01	2.173e+01
	Valor-P	–	7.557e-10 >

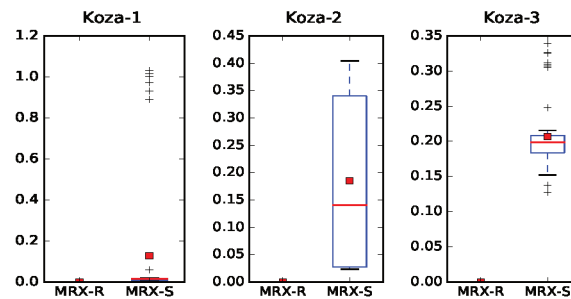


Figura 15 – Dispersão dos erros das respostas fornecidas no treinamento das funções do *benchmark* Koza: efeito do acréscimo de subárvores aleatórias.

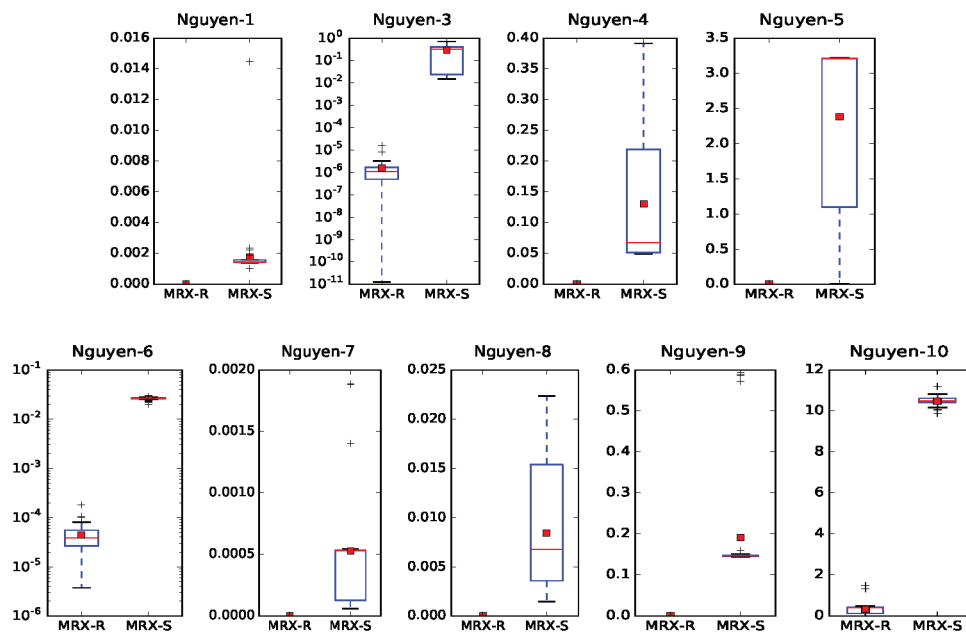


Figura 16 – Dispersão dos erros das respostas fornecidas no treinamento das funções do *benchmark* Nguyen: efeito do acréscimo de subárvores aleatórias.

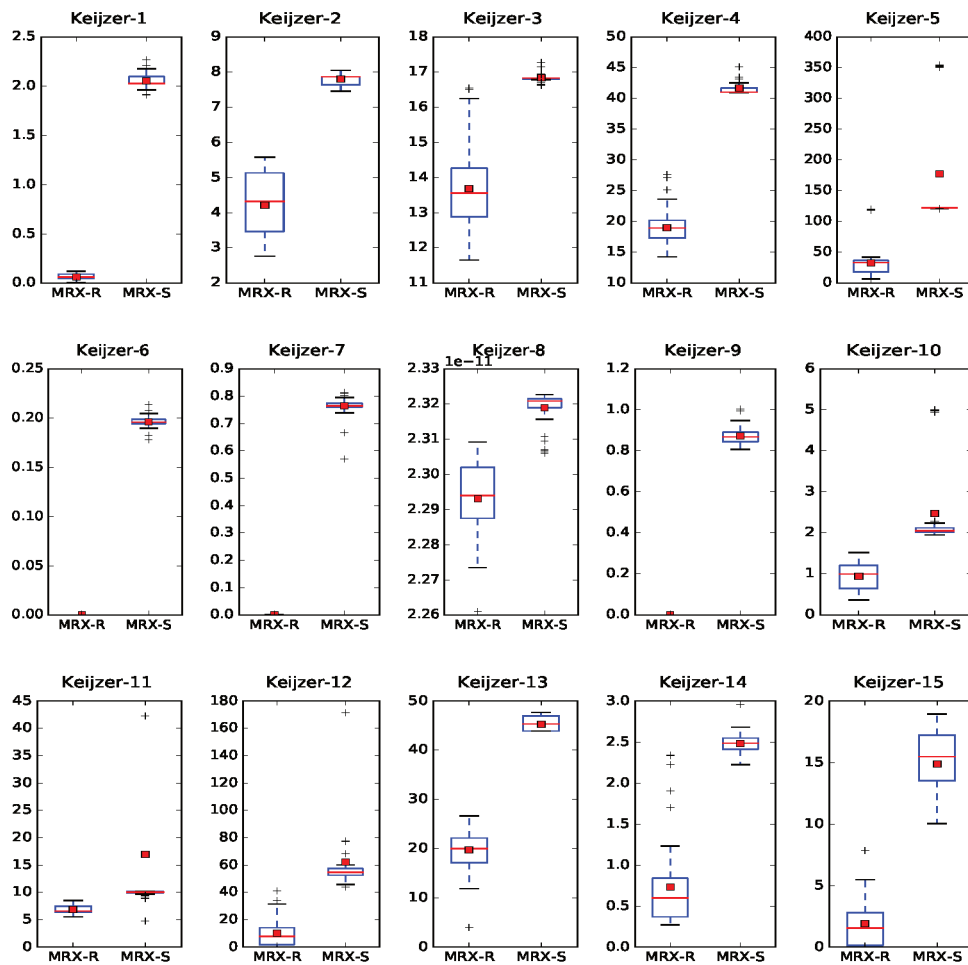


Figura 17 – Dispersão dos erros das respostas fornecidas no treinamento das funções do *benchmark* Keijzer: efeito do acréscimo de subárvores aleatórias.

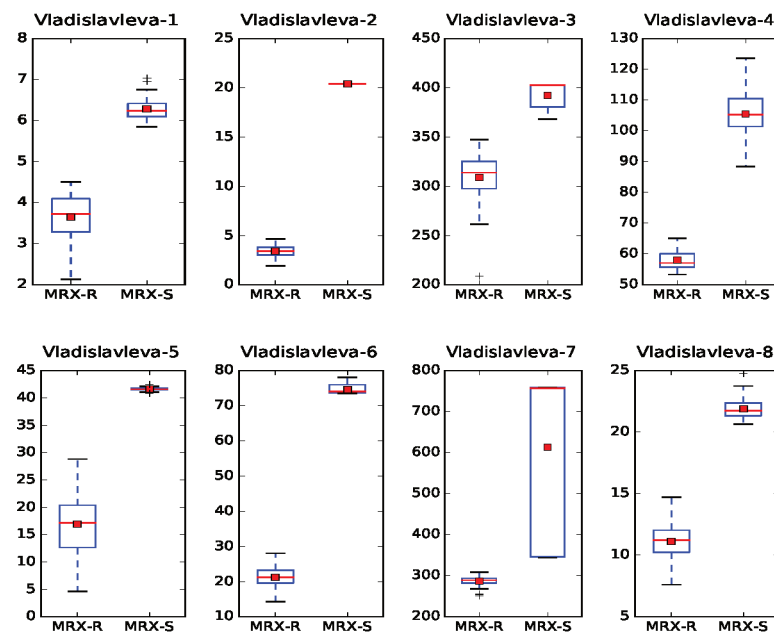


Figura 18 – Dispersão dos erros das respostas fornecidas no treinamento das funções do *benchmark* Vladislavleva: efeito do acréscimo de subárvores aleatórias.

4.2.3 Teste das melhores respostas

As tabelas 14, 15, 16 e 17 apresentam os resultados dos testes dos melhores indivíduos encontrados ao final da etapa de treinamento. A capacidade de generalização mostrou-se favorável à versão que não utiliza subárvores aleatórias em apenas cinco das trinta e cinco funções. Entretanto, dessas cinco apenas três mostram-se estatisticamente superiores.

Tabela 14 – Resultado dos testes do *benchmark* Koza: efeito do acréscimo de subárvores aleatórias.

Função	Estatística	MRX-R	MRX-S
Koza-1	Mediana	4.000e-08	3.268e-02
	Valor-P	–	7.557e-10 >
Koza-2	Mediana	1.811e-08	1.924e-01
	Valor-P	–	7.557e-10 >
Koza-3	Mediana	3.522e-08	3.879e-01
	Valor-P	–	7.557e-10 >

Tabela 15 – Resultado dos testes do *benchmark* Nguenyn: efeito do acréscimo de subárvores aleatórias.

Função	Estatística	MRX-R	MRX-S
Nguyen-1	Mediana	1.810e-11	4.208e-03
	Valor-P	–	7.557e-10 >
Nguyen-3	Mediana	3.305e-05	9.465e-01
	Valor-P	–	7.557e-10 >
Nguyen-4	Mediana	1.676e-05	1.302e-01
	Valor-P	–	7.557e-10 >
Nguyen-5	Mediana	9.936e-07	3.255e+00
	Valor-P	–	7.557e-10 >
Nguyen-6	Mediana	7.469e-04	2.742e-02
	Valor-P	–	7.557e-10 >
Nguyen-7	Mediana	1.019e-05	2.475e-03
	Valor-P	–	7.557e-10 >
Nguyen-8	Mediana	4.540e-05	1.779e-02
	Valor-P	–	7.557e-10 >
Nguyen-9	Mediana	1.475e-05	1.908e-01
	Valor-P	–	7.557e-10 >
Nguyen-10	Mediana	4.921e-01	1.189e+01
	Valor-P	–	7.557e-10 >

Tabela 16 – Resultados dos testes do *benchmark* Keijzer: efeito do acréscimo de subárvores aleatórias.

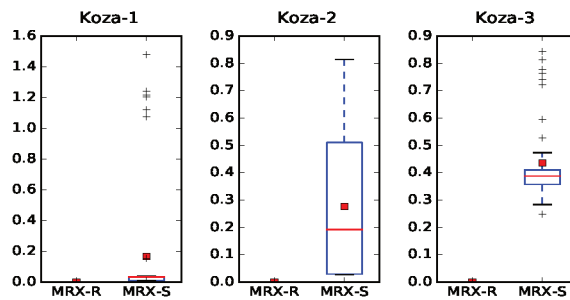
Função	Estatística	MRX-R	MRX-S
Keijzer-1	Mediana	6.831e+01	1.997e+02
	Valor-P	–	3.616e-01 =
Keijzer-2	Mediana	4.851e+02	7.793e+02
	Valor-P	–	1.499e-08 >
Keijzer-3	Mediana	1.487e+03	1.725e+03
	Valor-P	–	2.027e-05 >
Keijzer-4	Mediana	1.898e+01	4.103e+01
	Valor-P	–	7.557e-10 >
Keijzer-5	Mediana	3.547e+02	1.291e+03
	Valor-P	–	8.031e-10 >
Keijzer-6	Mediana	2.094e-02	2.516e+01
	Valor-P	–	7.557e-10 >
Keijzer-7	Mediana	5.144e-04	7.057e+00
	Valor-P	–	7.557e-10 >
Keijzer-8	Mediana	2.538e-10	2.511e-10
	Valor-P	–	6.024e-09 <
Keijzer-9	Mediana	3.085e-03	8.103e+00
	Valor-P	–	7.557e-10 >
Keijzer-10	Mediana	1.369e+02	2.144e+02
	Valor-P	–	8.031e-10 >
Keijzer-11	Mediana	2.721e+05	2.390e+05
	Valor-P	–	2.078e-01 =
Keijzer-12	Mediana	5.673e+05	1.703e+06
	Valor-P	–	1.294e-04 >
Keijzer-13	Mediana	1.071e+06	9.238e+05
	Valor-P	–	2.177e-03 <
Keijzer-14	Mediana	5.648e+04	1.470e+05
	Valor-P	–	5.176e-02 =
Keijzer-15	Mediana	1.040e+05	7.992e+05
	Valor-P	–	7.582e-09 >

Informações sobre a distribuição dos resultados dos testes complementam as infor-

Tabela 17 – Resultado dos testes do *benchmark* Vladislavleva: efeito do acréscimo de subárvores aleatórias.

Função	Estatística	MRX-R	MRX-S
Vladislavleva-1	Mediana	3.920e+02	6.689e+02
	Valor-P		4.517e-02 =
Vladislavleva-2	Mediana	1.339e+01	4.126e+01
	Valor-P		2.474e-07 >
Vladislavleva-3	Mediana	2.540e+03	3.309e+03
	Valor-P		2.257e-01 =
Vladislavleva-4	Mediana	3.983e+02	8.593e+02
	Valor-P		9.587e-02 =
Vladislavleva-5	Mediana	2.535e+02	5.648e+02
	Valor-P		2.298e-06 >
Vladislavleva-6	Mediana	5.177e+07	2.465e+05
	Valor-P		2.229e-09 <
Vladislavleva-7	Mediana	1.060e+03	2.786e+03
	Valor-P		4.840e-01 =
Vladislavleva-8	Mediana	1.119e+03	9.904e+02
	Valor-P		7.334e-02 =

mações apresentadas nas tabelas. Para as funções Koza (Figura 19) e Nguyen (Figura 20), tanto a dispersão quanto a quantidade de *outliers* foi menor na versão que empregou o MRX-Rand. Situação contrária é observada para as funções Keijzer (Figura 21) e Vladislavleva (Figura 22). Embora seja verificada a presença de *outliers* quando há o acréscimo de árvores aleatórias, observa-se que o comportamento ditado pela medida de tendência central ainda aponta, para a maioria dos casos, benefícios para a abordagem.

Figura 19 – Dispersão dos erros das respostas fornecidas nos testes das funções do *benchmark* Koza: efeito do acréscimo de subárvores aleatórias.

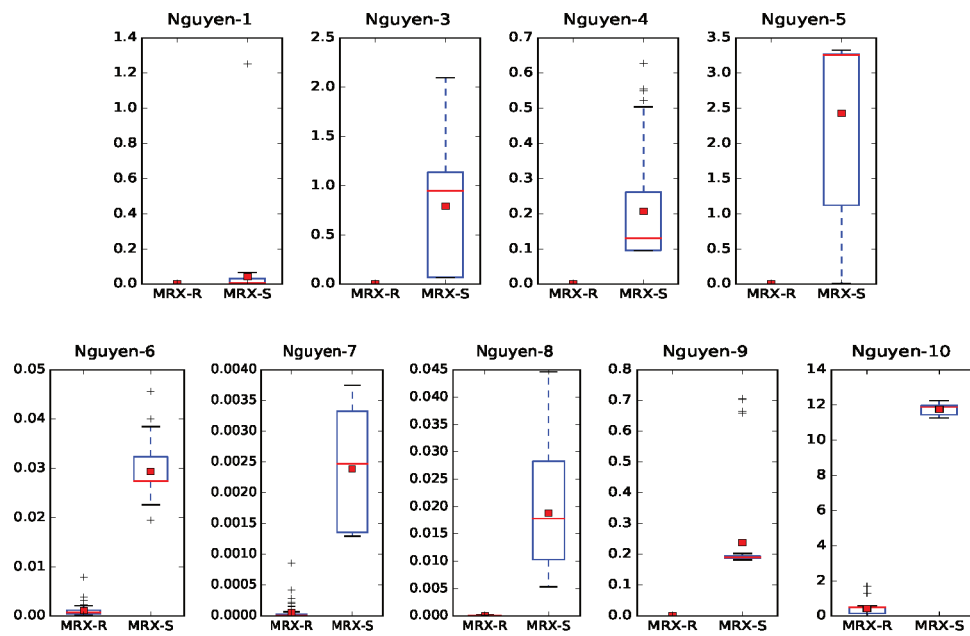


Figura 20 – Dispersão dos erros das respostas fornecidas nos testes das funções do *benchmark* Nguyen: efeito do acréscimo de subárvores aleatórias.

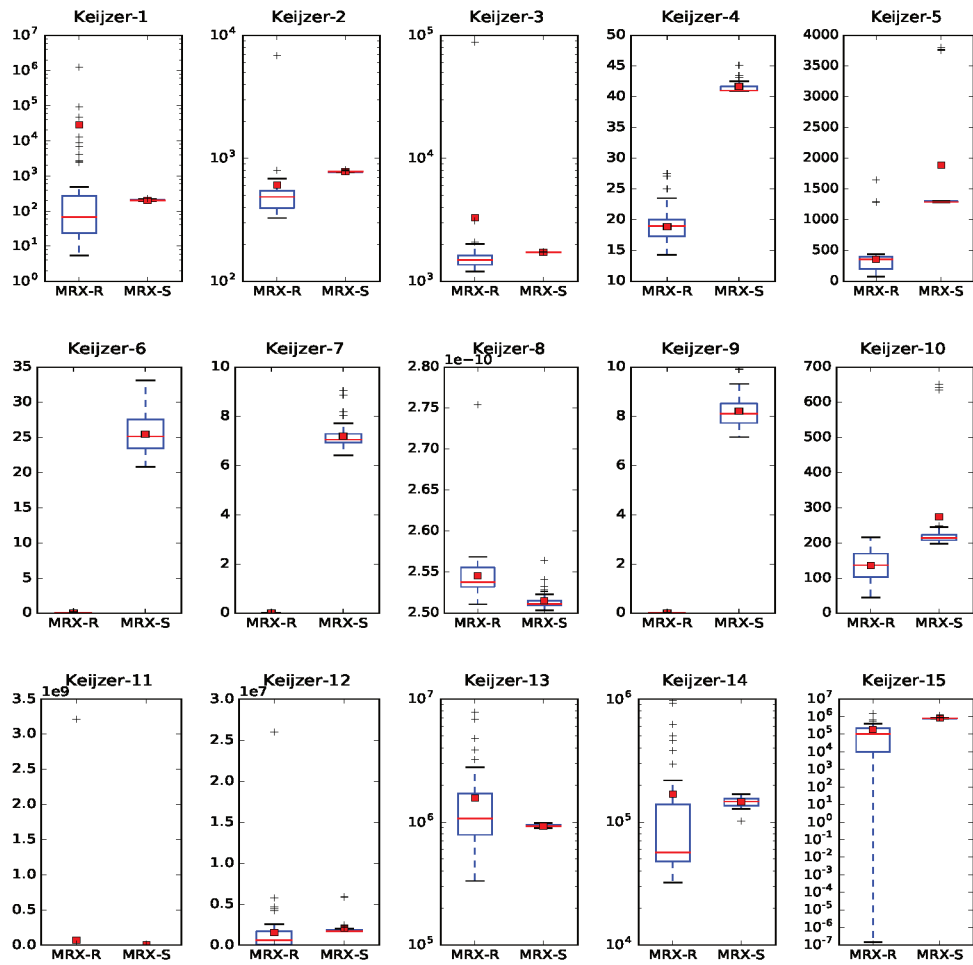


Figura 21 – Dispersão dos erros das respostas fornecidas nos testes das funções do *benchmark* Keijzer: efeito do acréscimo de subárvores aleatórias.

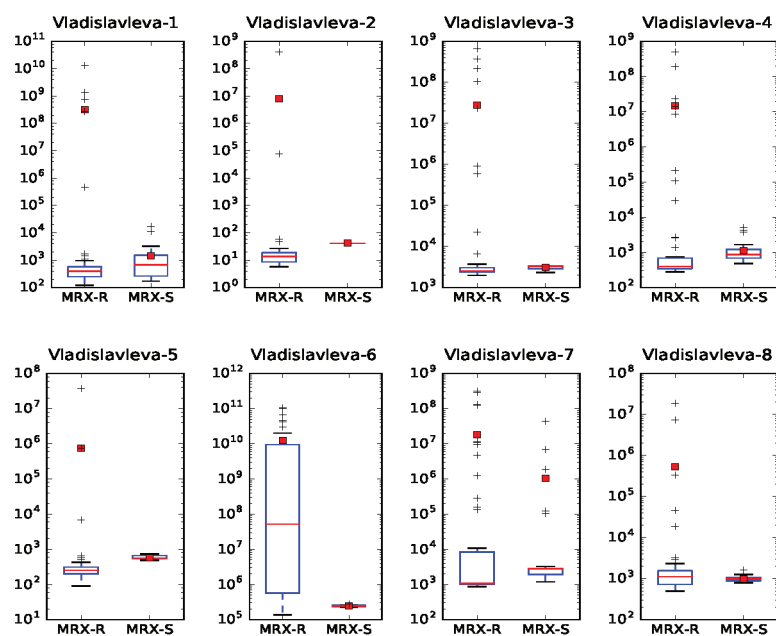


Figura 22 – Dispersão dos erros das respostas fornecidas nos testes das funções do *benchmark* Vladislavleva: efeito do acréscimo de subárvores aleatórias.

4.2.4 Tamanho das melhores respostas

É natural esperar que o operador MRX-Rand gere filhos maiores, dado que explicitamente aumenta o tamanho dos indivíduos criados. Isso pode ser percebido nas tabelas 18, 19, 20 e 21, onde podemos observar que as diferenças são estatisticamente comprovadas para todas as funções.

Entretanto, quando se observa as formulações de cada função (tabelas 5, 6, 7 e 8 da Seção 4.1.2) vemos que alguns resultados de tamanho para o MRX-Simples, embora numericamente menores, não são plausíveis de mapear uma resposta aceitável. Por exemplo, as funções Keijzer-1, Keijzer-2 e Keijzer-3 (Tabela 20), que correspondem à função $0.3x \sin(2\pi x)$ mapeada em diferentes domínios, não pode ser descrita por árvores com apenas 3 nós: pelo menos três nós já seriam utilizados para representar a multiplicação. Análise semelhante pode ser estendida a outras funções. De fato, mesmo com o operador MRX-Rand fornecendo respostas maiores que o MRX-Simples, a efetiva qualidade das respostas proporcionada pela inserção das árvores aleatórias compensa, como já pôde ser visto no treinamento e no teste, o tamanho das respostas obtidas.

Por fim, os gráficos descritos nas figuras 23, 24, 25 e 26 complementam as informações apresentadas nas tabelas.

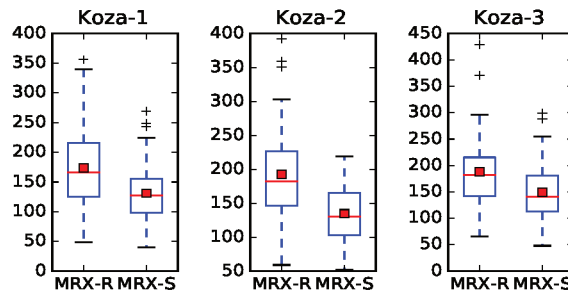


Figura 23 – Dispersão dos tamanhos das respostas fornecidas no treinamento das funções do *benchmark* Koza: efeito do acréscimo de subárvores aleatórias.

Tabela 18 – Quantidade de nós observados nas funções Koza: efeito do acréscimo de subárvores aleatórias.

Função	Estatística	MRX-R	MRX-S
Koza-1	Mediana	168	129
	Valor-P	–	7.5e-10 <
Koza-2	Mediana	183	131
	Valor-P	–	7.5e-10 <
Koza-3	Mediana	183	141
	Valor-P	–	7.4e-10 <

Tabela 19 – Quantidade de nós observados nas funções Nguyen: efeito do acréscimo de subárvores aleatórias.

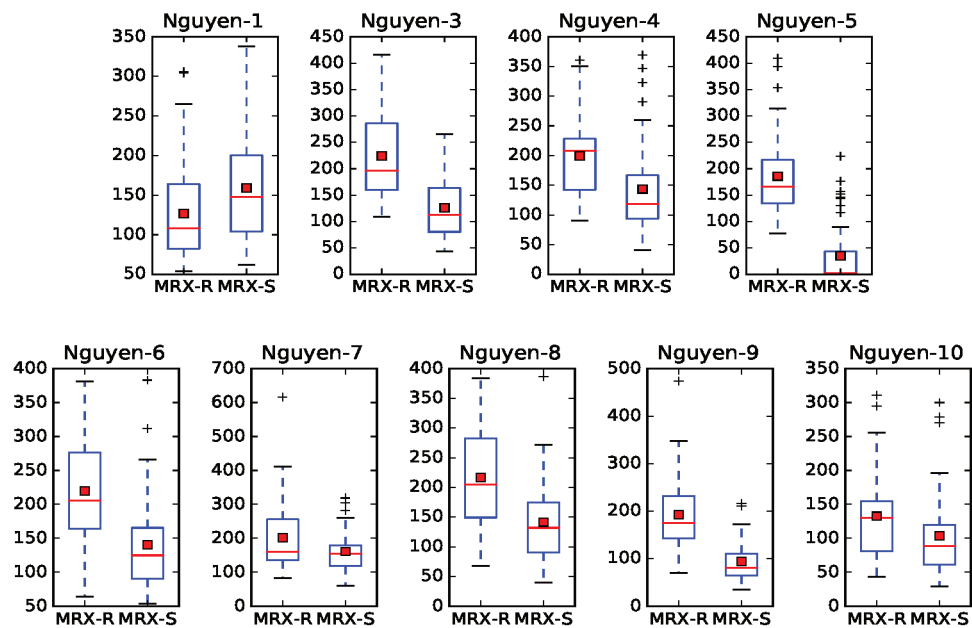
Função	Estatística	MRX-R	MRX-S
Nguyen-1	Mediana	109	149
	Valor-P	–	7.5e-10 >
Nguyen-3	Mediana	198	118
	Valor-P	–	7.5e-10 <
Nguyen-4	Mediana	213	121
	Valor-P	–	1.1e-09 <
Nguyen-5	Mediana	167	2
	Valor-P	–	7.6e-10 <
Nguyen-6	Mediana	206	126
	Valor-P	–	8.0e-10 <
Nguyen-7	Mediana	162	154
	Valor-P	–	7.5e-10 <
Nguyen-8	Mediana	206	132
	Valor-P	–	8.0e-10 <
Nguyen-9	Mediana	176	83
	Valor-P	–	7.5e-10 <
Nguyen-10	Mediana	131	92
	Valor-P	–	1.0e-09 <

Tabela 20 – Quantidade de nós observados nas funções Keijzer: efeito do acréscimo de subárvores aleatórias.

Função	Estatística	MRX-R	MRX-S
Keijzer-1	Mediana	203	3
	Valor-P	–	7.6e-10 <
Keijzer-2	Mediana	124	3
	Valor-P	–	7.6e-10 <
Keijzer-3	Mediana	130	2
	Valor-P	–	7.6e-10 <
Keijzer-4	Mediana	146	2
	Valor-P	–	7.5e-10 <
Keijzer-5	Mediana	90	65
	Valor-P	–	7.4e-10 <
Keijzer-6	Mediana	222	90
	Valor-P	–	7.5e-10 <
Keijzer-7	Mediana	222	86
	Valor-P	–	7.5e-10 <
Keijzer-8	Mediana	77	73
	Valor-P	–	3.7e-01 =
Keijzer-9	Mediana	196	92
	Valor-P	–	7.5e-10 <
Keijzer-10	Mediana	153	104
	Valor-P	–	7.5e-10 <
Keijzer-11	Mediana	154	50
	Valor-P	–	7.5e-10 <
Keijzer-12	Mediana	165	78
	Valor-P	–	7.5e-10 <
Keijzer-13	Mediana	134	3
	Valor-P	–	7.5e-10 <
Keijzer-14	Mediana	116	60
	Valor-P	–	7.5e-10 <
Keijzer-15	Mediana	147	107
	Valor-P	–	7.5e-10 <

Tabela 21 – Quantidade de nós observados nas funções Vladislavleva: efeito do acréscimo de subárvores aleatórias.

Função	Estatística	MRX-R	MRX-S
Vladislavleva-1	Mediana	186	130
	Valor-P	–	8.0e-10 <
Vladislavleva-2	Mediana	160	4
	Valor-P	–	7.5e-10 <
Vladislavleva-3	Mediana	132	2
	Valor-P	–	7.6e-10 <
Vladislavleva-4	Mediana	207	140
	Valor-P	–	7.4e-10 <
Vladislavleva-5	Mediana	230	75
	Valor-P	–	7.5e-10 <
Vladislavleva-6	Mediana	174	2
	Valor-P	–	7.5e-10 <
Vladislavleva-7	Mediana	162	2
	Valor-P	–	7.5e-10 <
Vladislavleva-8	Mediana	257	162
	Valor-P	–	7.5e-10 <

Figura 24 – Dispersão dos tamanhos das respostas fornecidas no treinamento das funções do *benchmark* Nguyen: efeito do acréscimo de subárvores aleatórias.

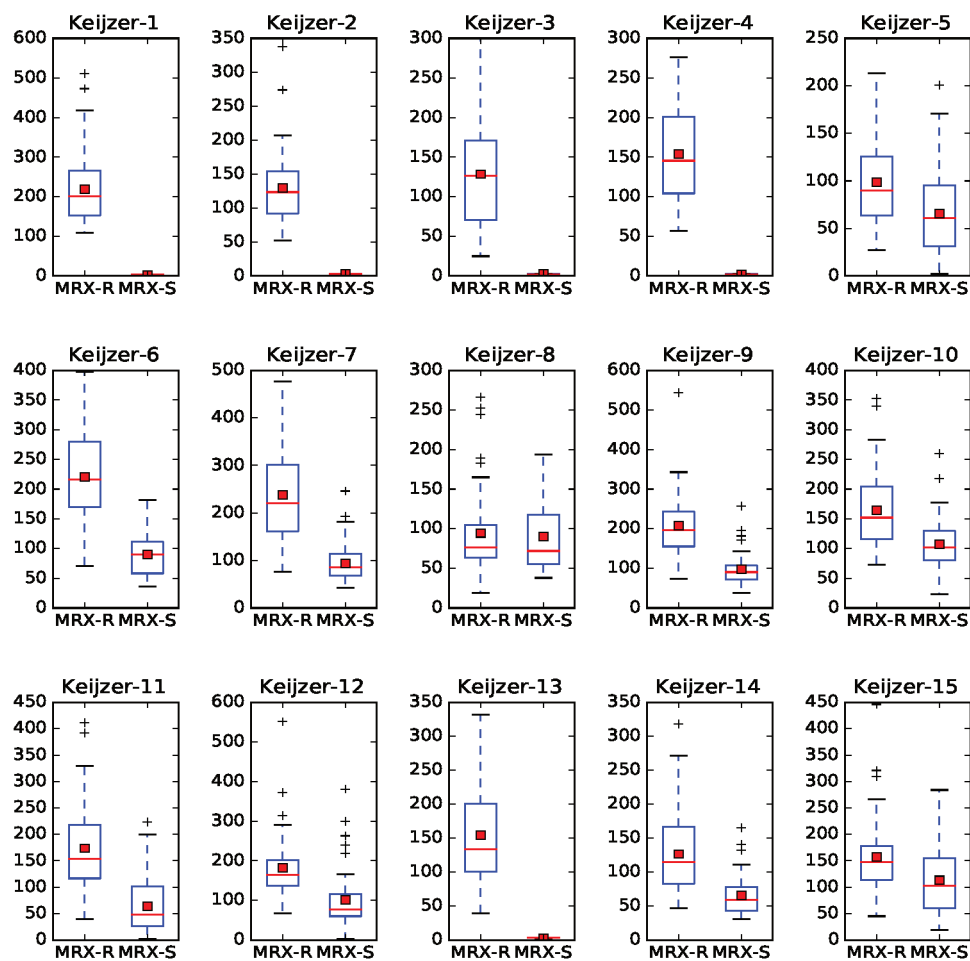


Figura 25 – Dispersão dos tamanho das respostas fornecidas no treinamento das funções do *benchmark* Keijzer: efeito do acréscimo de subárvores aleatórias.

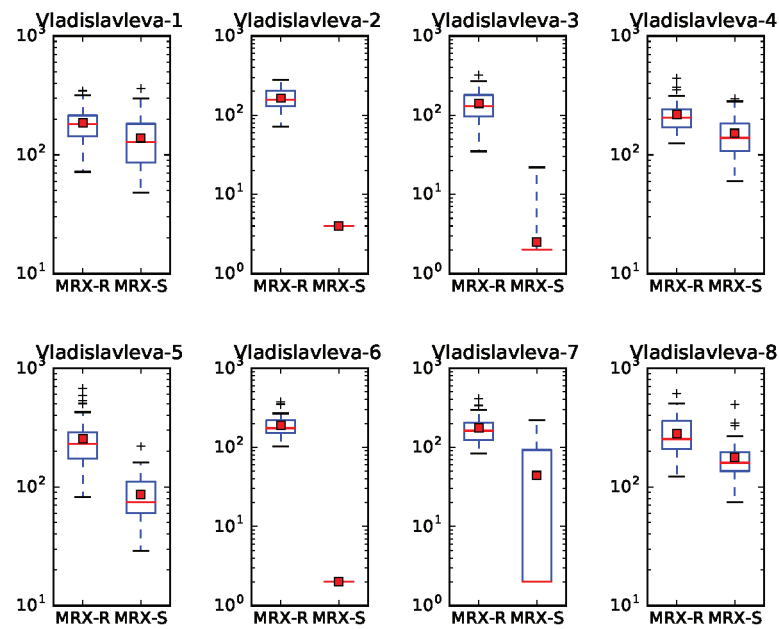
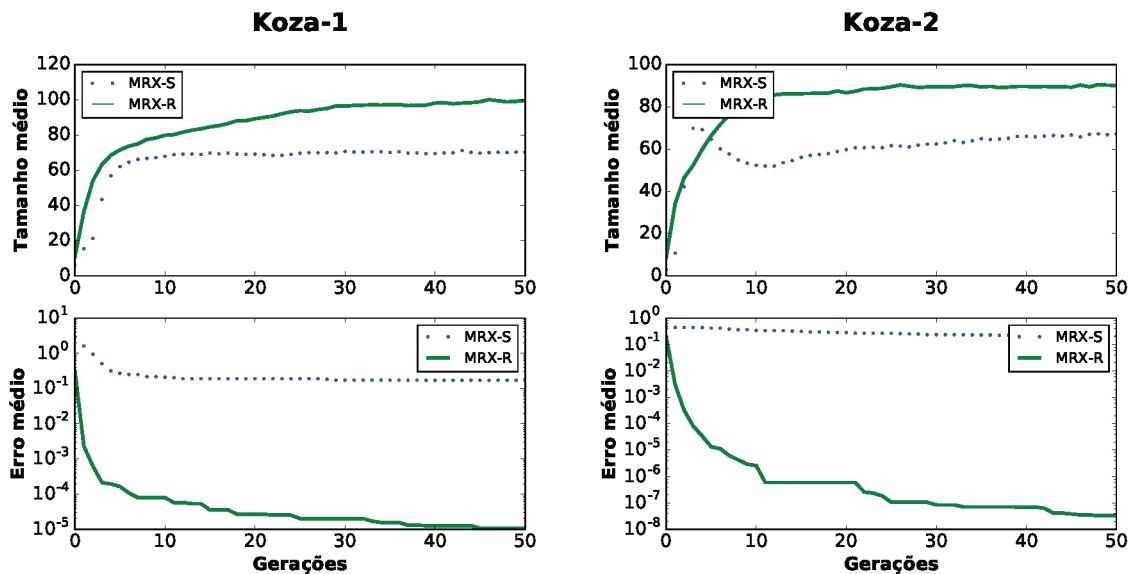


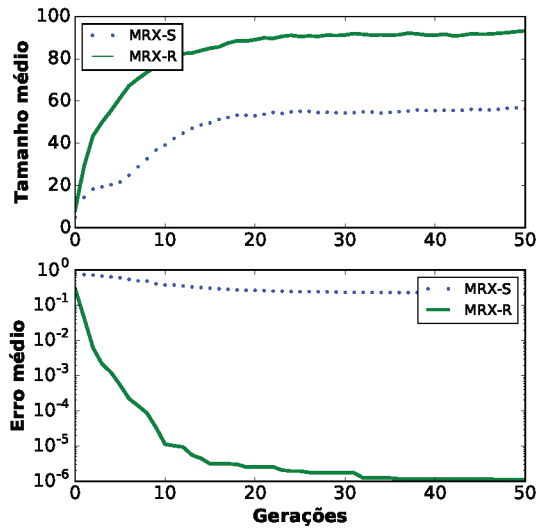
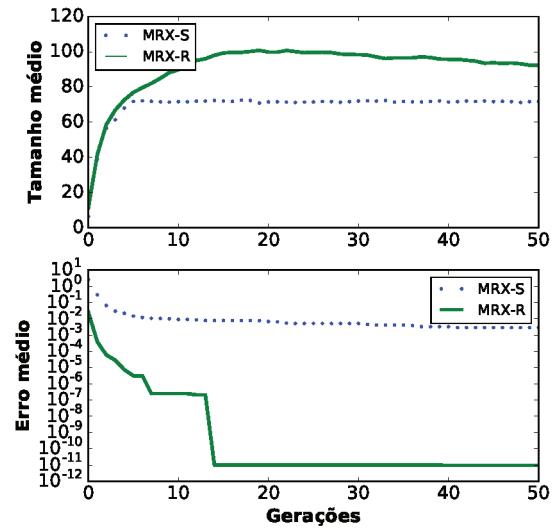
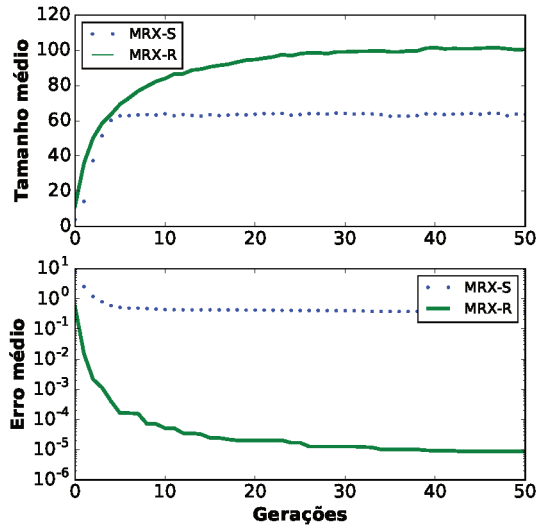
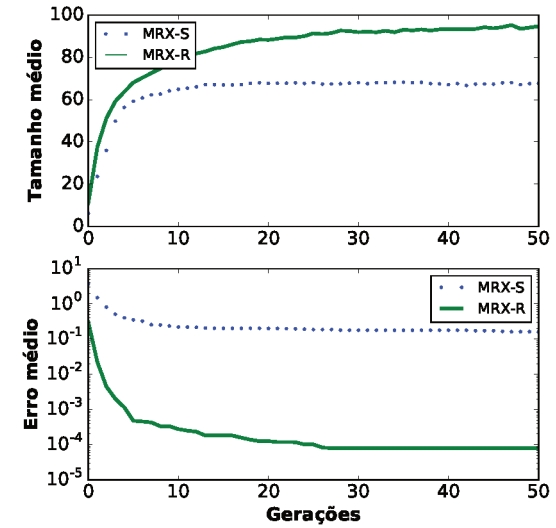
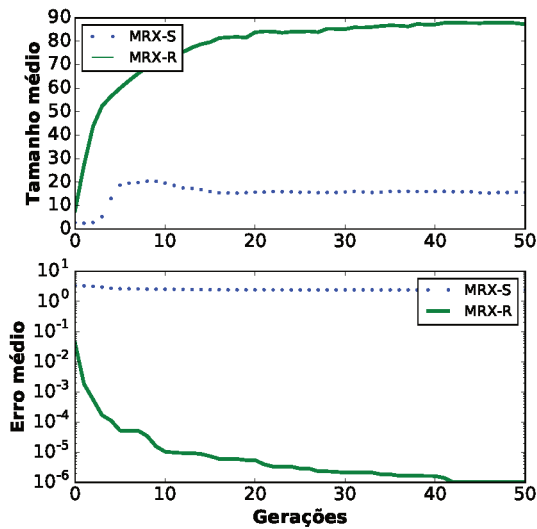
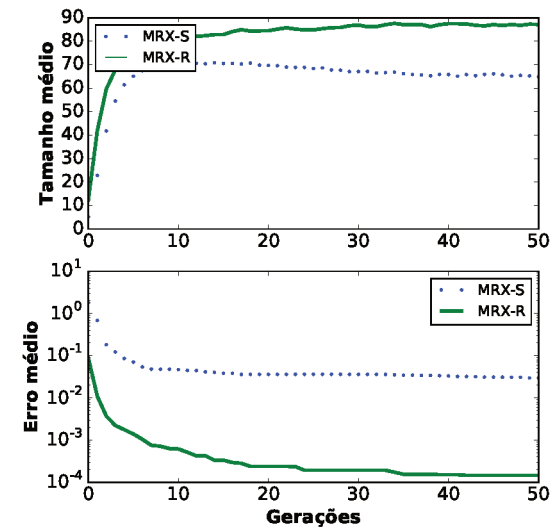
Figura 26 – Dispersão dos tamanhos das respostas fornecidas no treinamento das funções do *benchmark* Vladislavleva: efeito do acréscimo de subárvores aleatórias.

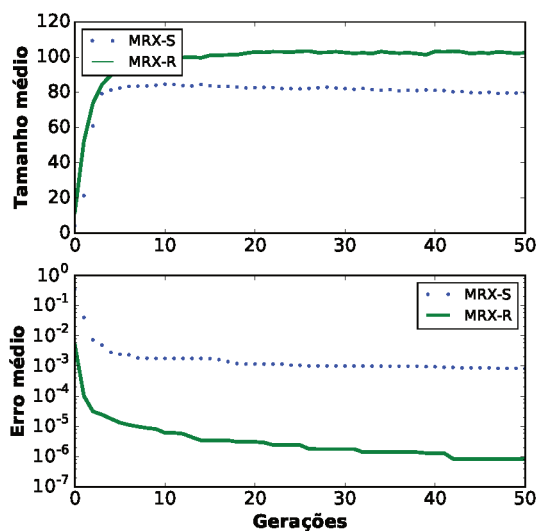
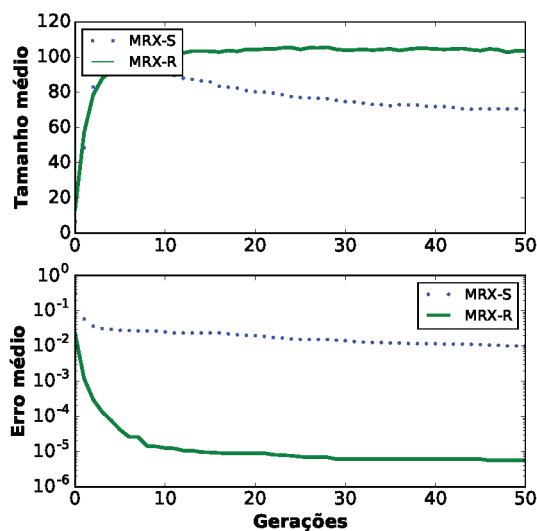
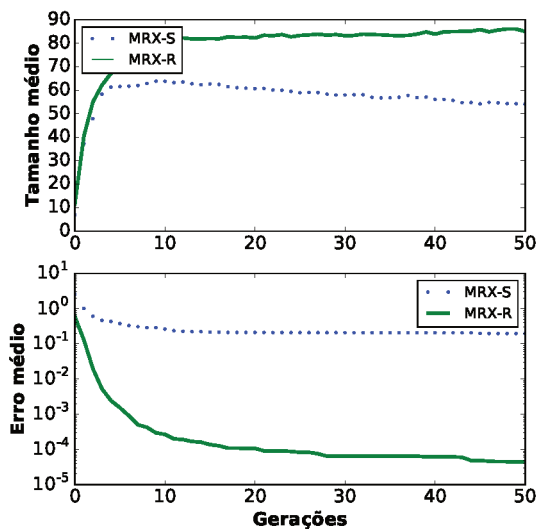
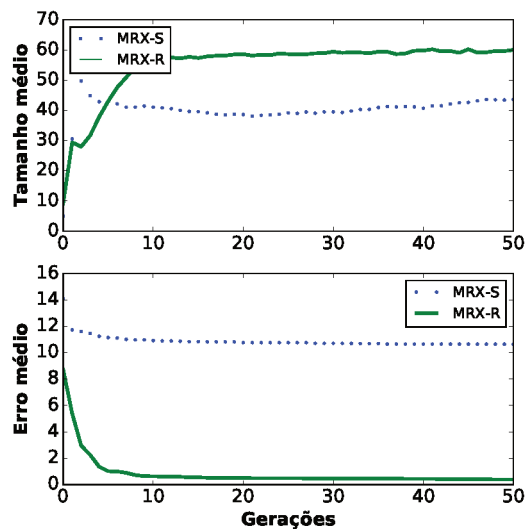
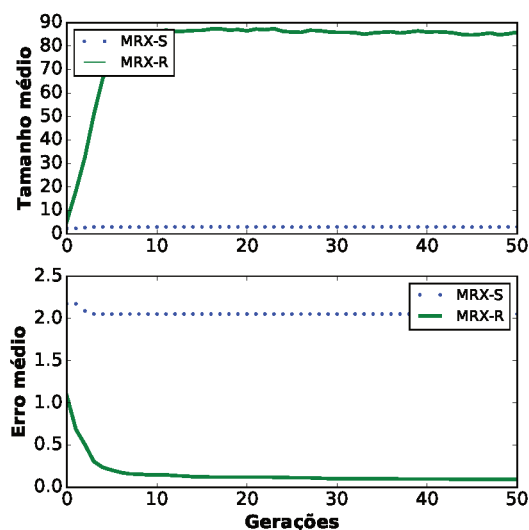
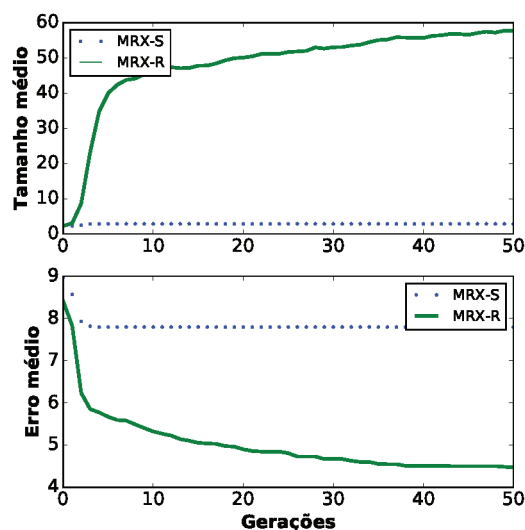
4.2.5 Efeito *bloat*

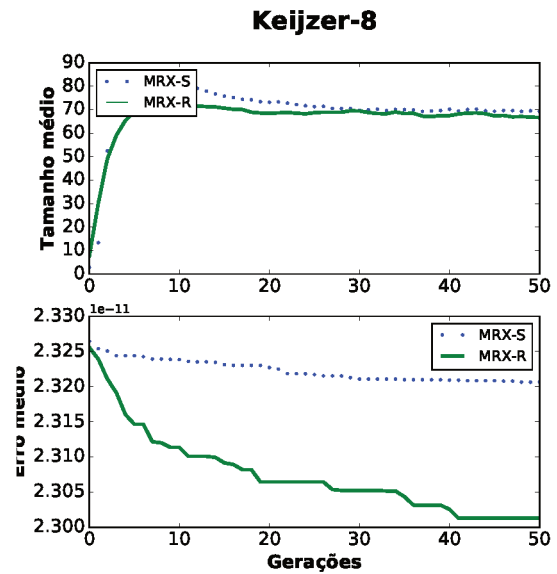
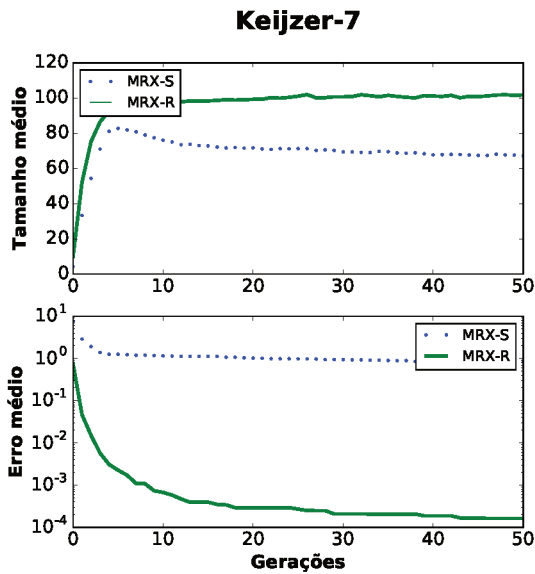
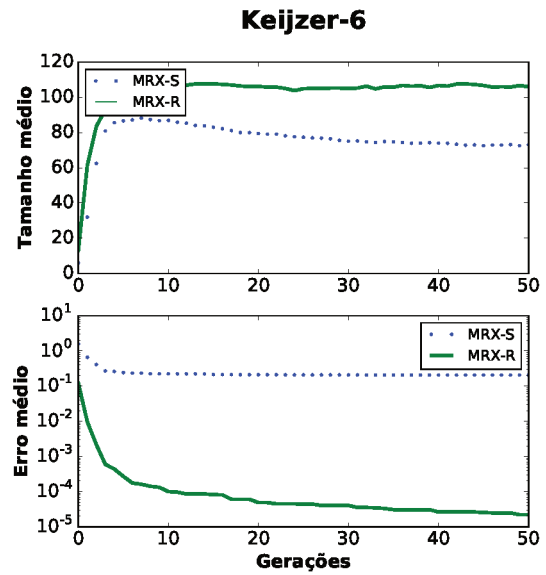
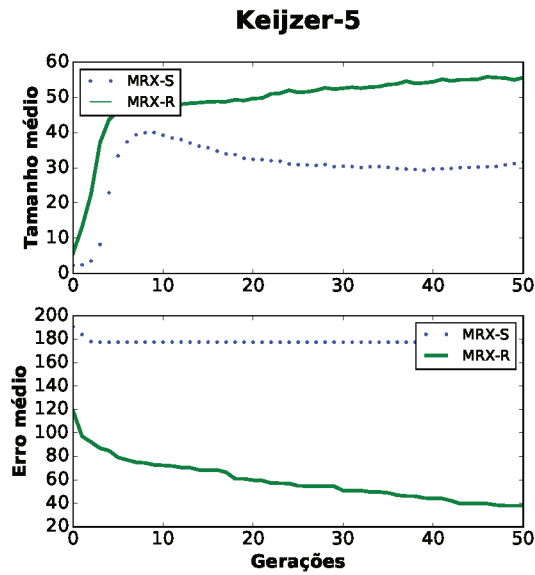
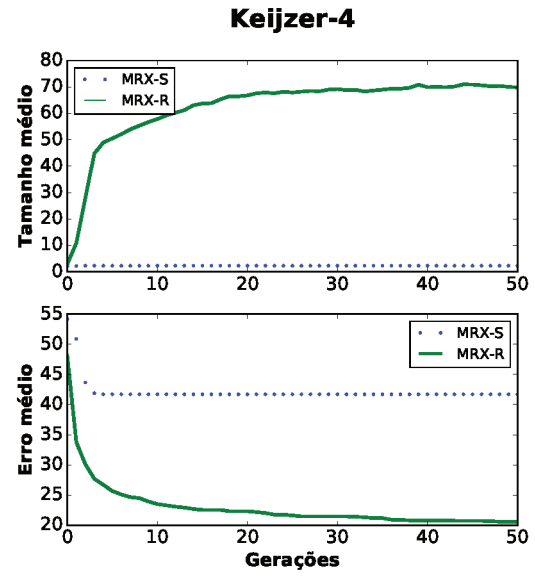
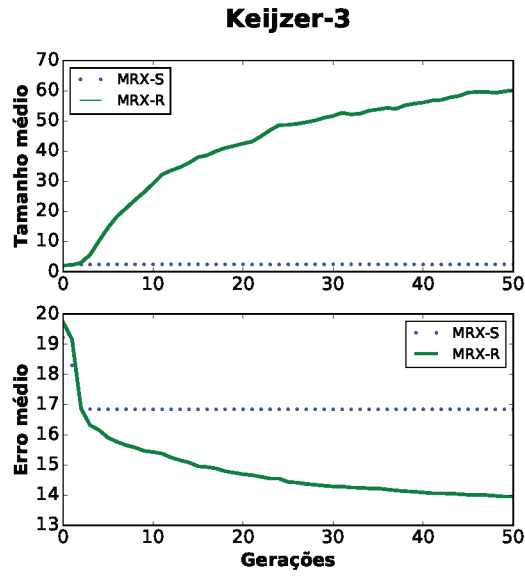
Os gráficos que compõem a Figura 27 apresentam o comportamento geral das funções que são evoluídas com a versão do MRX simples (MRX-S) e com a versão que insere subárvores aleatórias (MRX-R). De maneira geral, observa-se que o número médio de nós é maior quando do uso do MRX-S, efeito que já é esperado. Entretanto, não há *bloat* em nenhuma das versões: as árvores produzidas pela versão MRX-S são maiores, mas se mantém constante, sem a tendência de crescimento exponencial própria do *bloat*.

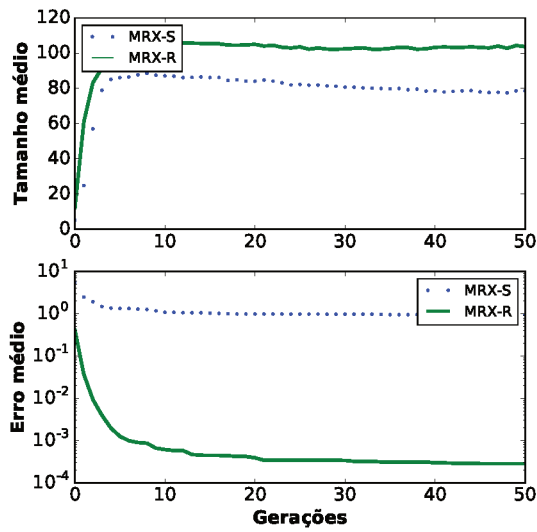
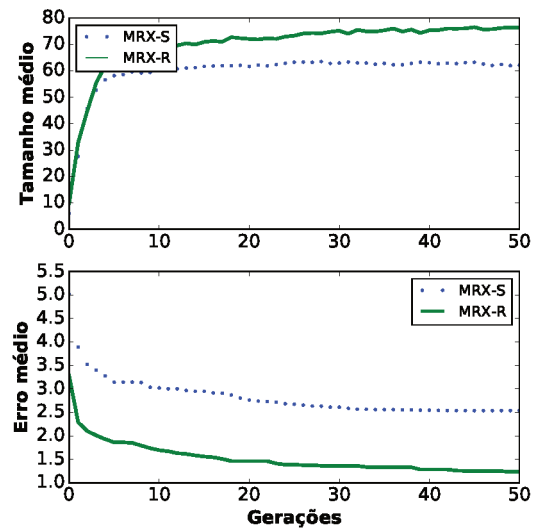
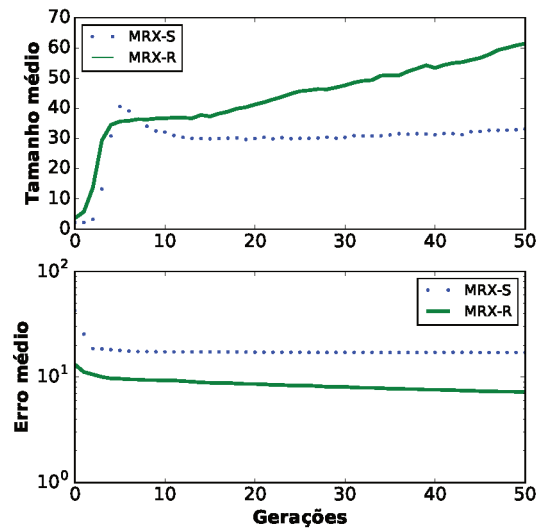
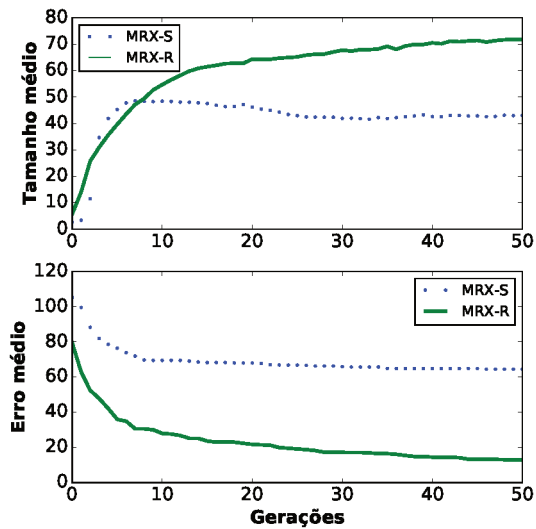
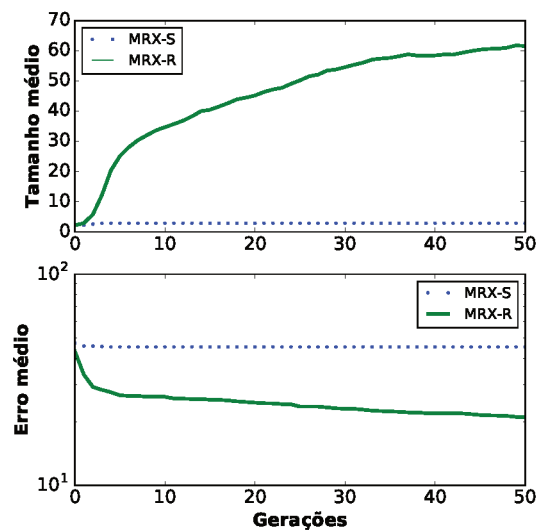
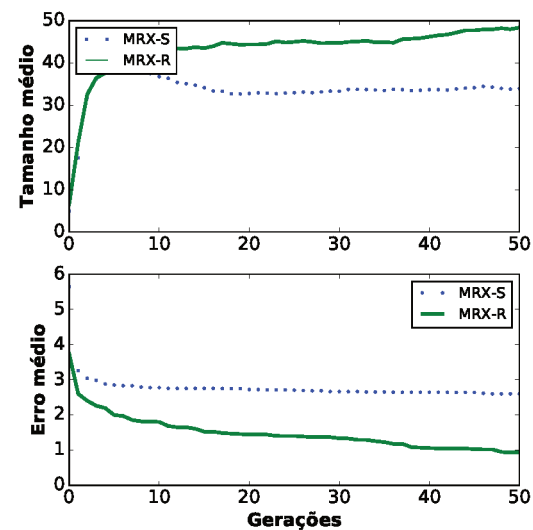
A análise conjunta dos gráficos de tamanho médio e erro médio para cada função proporciona outra interessante conclusão. Em casos onde o tamanho médio da população mantém-se desproporcionalmente menor para o MRX-S, o erro do menor indivíduo se mantém praticamente constante desde as primeiras gerações. Parece haver um indivíduo que supera os demais já no início das gerações e que este domina a população até o fim da evolução. Já o comportamento proporcionado pelo MRX-R é completamente distinto, com o erro médio decaindo sempre ao longo das gerações. Dessa forma, a diversidade parece ser mantida em níveis mais altos quando subárvores aleatórias são inseridas.

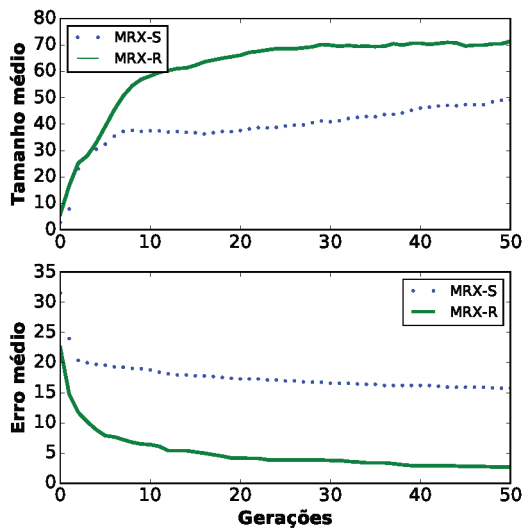
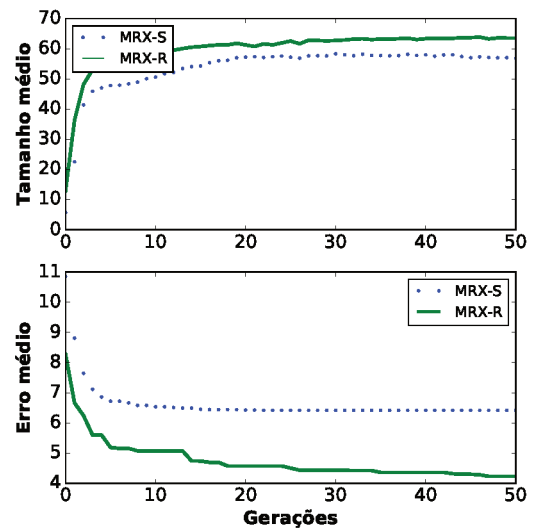
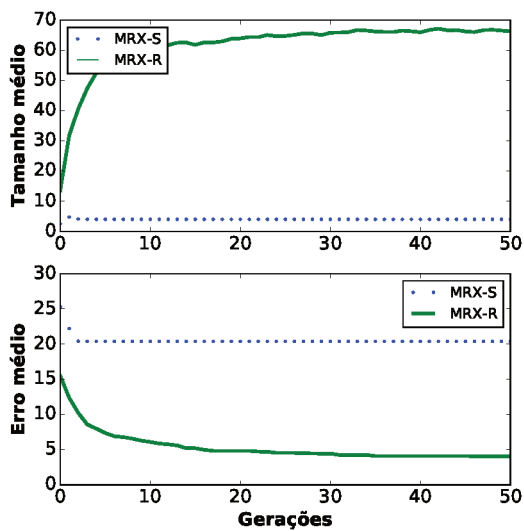
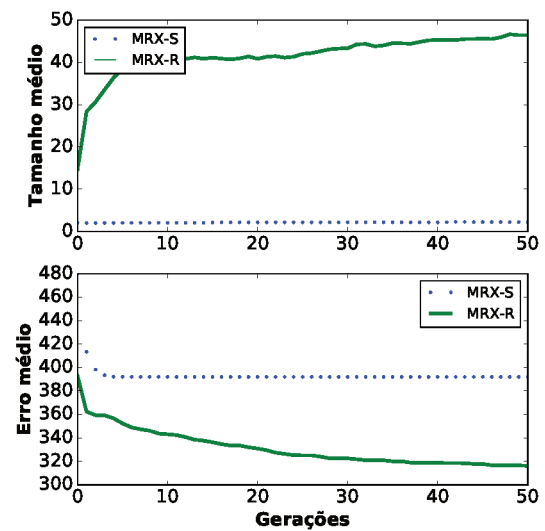
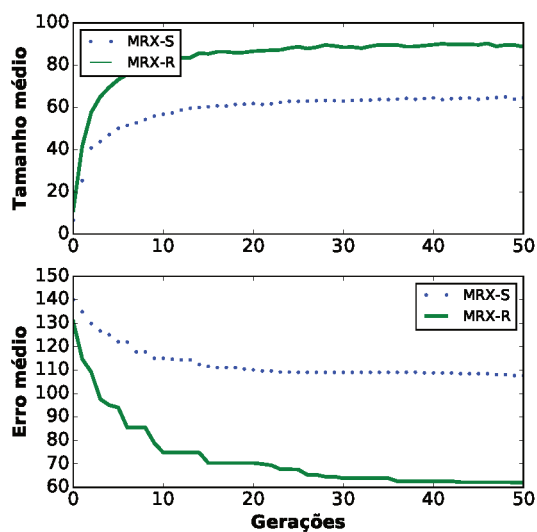
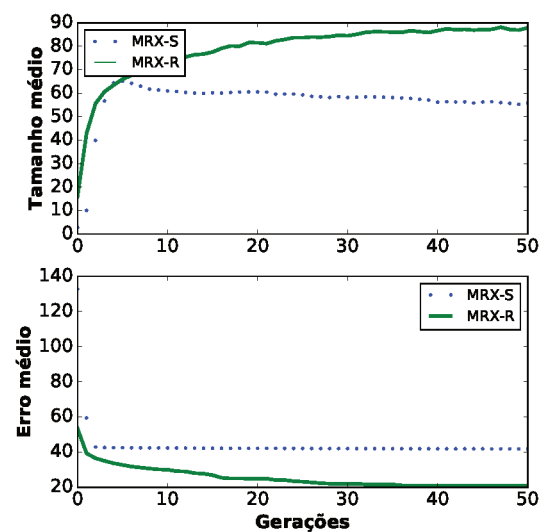


Koza-3**Nguyen-1****Nguyen-3****Nguyen-4****Nguyen-5****Nguyen-6**

Nguyen-7**Nguyen-8****Nguyen-9****Nguyen-10****Keijzer-1****Keijzer-2**



Keijzer-9**Keijzer-10****Keijzer-11****Keijzer-12****Keijzer-13****Keijzer-14**

Keijzer-15**Vladislavleva-1****Vladislavleva-2****Vladislavleva-3****Vladislavleva-4****Vladislavleva-5**

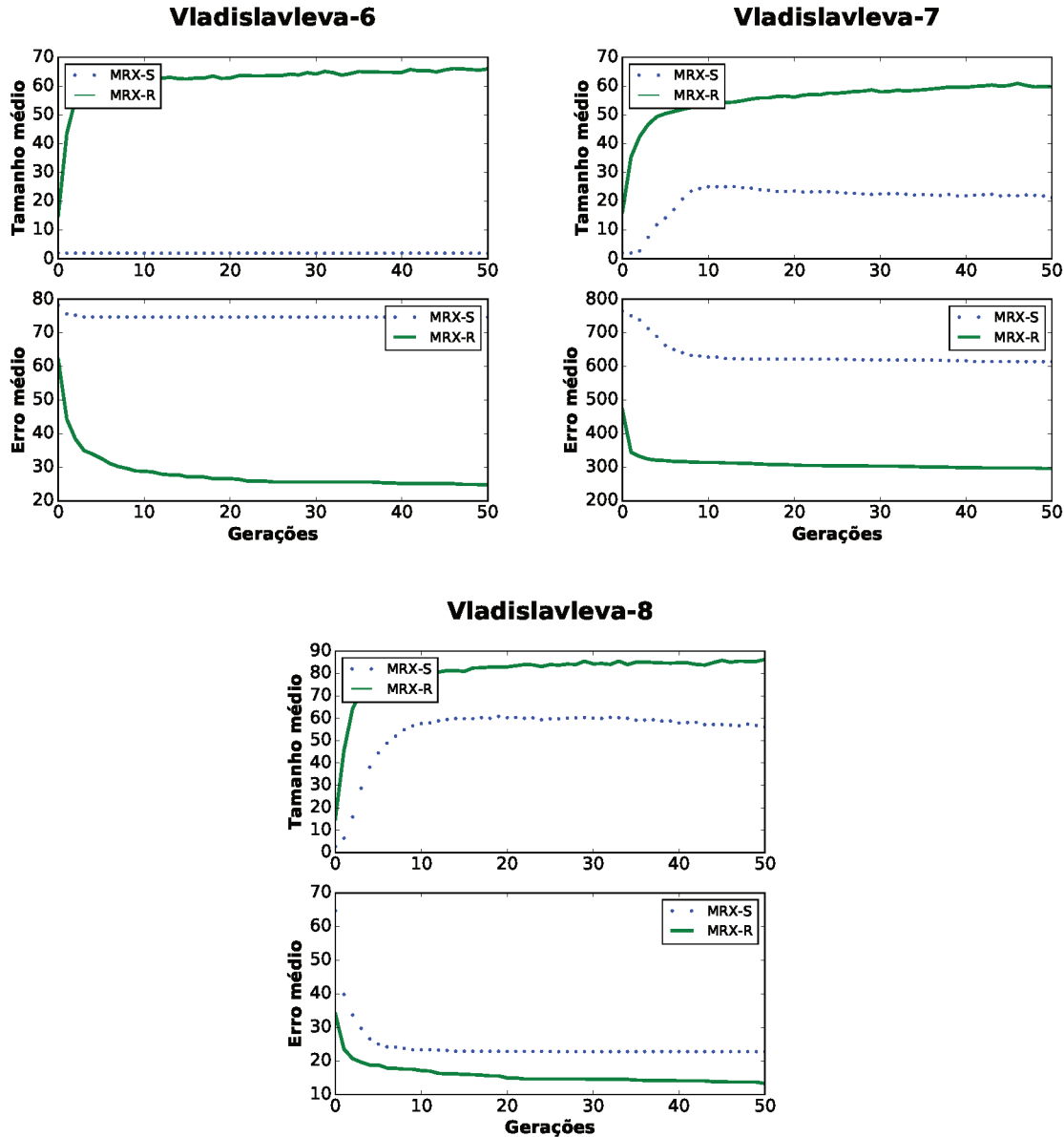


Figura 27 – O *bloat* em funções dos *benchmarks* Koza, Nguyen, Keijzer e Vladislavleva: efeito do acréscimo de subárvores aleatórias.

4.3 Comparação com operador sintático

Esta seção apresenta os resultados obtidos pelo experimento que compara a GP-MRX com a PG canônica, que denominamos GP e que implementa a recombinação sintática de subárvore. Um método de PG é considerado de alta qualidade se conseguir gerar indivíduos capazes de “aprender” certo padrão com alto índice de acerto, generalizar o padrão, fornecendo saídas corretas quando submetido a entradas que não foram vistas na fase de treinamento, e ainda apresentar baixa quantidade de elementos (nós) em sua estrutura.

A discussão realizada na Seção 3.4 levantou a hipótese de que a aplicação do operador MRX tende a diminuir o crescimento do tamanho médio da população, considerando alguns

fundamentos teóricos relacionados ao efeito *bloat*. Por essa razão, analisamos também o comportamento global da população no que se refere à quantidade média de nós de seus indivíduos.

4.3.1 Configuração dos experimentos

Conduzimos o experimento utilizando duas versões de PG: a canônica, que denominamos GP, e outra que implementa o MRX, denominada GP-MRX – ambas apresentadas na Seção 4.1.1. Cada versão de PG foi executada 50 vezes, utilizando funções dos *benchmarks* Koza, Nguyen, Keijzer e Vladislavleva. Para cada rodada, armazenamos dados referentes ao melhor indivíduo: a soma dos erros absolutos (SAE) ao final do treinamento, a soma dos erros absolutos do melhor indivíduo quando submetido ao conjunto de teste e a quantidade de nós dos melhores indivíduos. Armazenamos também os valores médios necessários à análise do *bloat*.

Aos resultados obtidos, aplicamos o teste de Wilcoxon, com significância de 1%, e relatamos o valor *p* encontrado. Para explicitar a relação estatística entre os erros verificados GP e na GP-MRX, acrescentamos às tabelas informações no formato de sinais, onde “=” indica que os valores fornecidos por ambas as versões são estatisticamente iguais, “<” indica que valor fornecido pela GP é estatisticamente menor e “>” indica que o valor fornecido pela GP é estatisticamente maior.

As execuções da GP e da GP-MRX foram realizadas adotando os seguintes parâmetros:

- Tamanho da população: 500;
- Taxa de recombinação: 90% para a GP e 100% para a GP-MRX;
- Taxa de mutação: 5% para a GP e 0% para a GP-MRX;
- Quantidade de gerações: 51;
- Torneio: 5;
- Profundidade inicial: 6;
- Funções e constantes: adaptadas a cada *dataset*, de acordo com os dados da Tabela 9.

A visualização da distribuição dos resultados foi feita com utilização de gráficos de caixa, onde, para cada função, foi plotado o *boxplot* para cada versão de PG. Tais informações servem de complemento aos resultados tabelados.

Para avaliar a tendência de crescimento médio da população apresentada pela GP e pela GP-MRX, realizamos 50 execuções de cada programa para cada uma das 35 funções

dos *benchmarks* utilizados. Para não influenciar os resultados, não aplicamos qualquer tipo de limitação no tamanho das árvores geradas ao longo das gerações.

4.3.2 Treinamento

Os dados organizados nas tabelas 22, 23, 24 e 25 referem-se aos resultados obtidos nos treinamentos das funções dos *benchmarks* Koza, Nguenyn, Keijzer e Vladislavleva, respectivamente, nas execuções da GP-MRX e da GP. São reportadas as comparações entre os valores médios (medianas) dos erros apresentados pelos melhores indivíduos encontrados ao longo dos ciclos de execução.

É notória a superioridade da GP-MRX, que foi estritamente melhor em 33 das 35 funções testadas. Os valores p , sempre muito abaixo de 0,01, proporcionam sustentação estatística aos resultados. A GP mostrou-se melhor no aprendizado do padrão mapeado pela função Nguyen-3 (Tabela 23), embora o teste estatístico tenha indicado as medianas entre as duas versões de PG não diferiu um do outro. Já a Nguyen-6 (também na Tabela 23) foi a única que foi de fato melhor mapeada pela GP.

A distribuição dos resultados foi organizada em gráficos de caixa e apresentados nas figuras 28, 29, 30 e 31, que contém informações relativas aos *benchmarks* Koza, Nguenyn, Keijzer e Vladislavleva, respectivamente. Os gráficos mostram que a GP-MRX tende a fornecer respostas com menor variação (tanto em termos de distância entre primeiro e terceiro quartil quanto em distância entre valores mínimo e máximo), enquanto a GP, além de ser mais dispersa, ainda conta com elevada quantidade de *outliers*.

Os *outliers* presentes na Nguyen-3 (Figura 29) geram desconfiança quanto ao resultado obtido pelo teste de hipótese, que declarou iguais as distribuições da GP e da GP-MRX para esta função. Para facilitar a visualização, colocamos seu gráfico em escala logarítmica. De fato, vemos que as duas implementações estão distribuídas de forma bem distinta, sendo que a GP-MRX forneceu resultados mais consistentes.

A função Nguyen-6 (Figura 29), a única que forneceu à GP resultados estatisticamente superiores à GP-MRX, foi também plotada em escala logarítmica. Embora não tenham sido identificados valores discrepantes na distribuição de respostas da GP, vemos que sua amplitude é muitas vezes superior ao observado nas respostas da GP-MRX que, mesmo com pior desempenho, proporcionou resultados mais concisos.

Tabela 22 – Treinamento usando funções do *benchmark* Koza: comparação entre o MRX e a recombinação de subárvores.

Função	Estatística	GP-MRX	GP
Koza-1	Mediana	1.385e-08	8.825e-02
	Valor-P	–	7.159e-09 >
Koza-2	Mediana	3.018e-09	1.597e-01
	Valor-P	–	8.031e-10 >
Koza-3	Mediana	2.356e-08	3.339e-01
	Valor-P	–	7.557e-10 >

Tabela 23 – Treinamento usando funções do *benchmark* Ngueyn: comparação entre o MRX e a recombinação de subárvores.

Função	Estatística	GP-MRX	GP
Nguyen-1	Mediana	8.425e-12	1.120e-11
	Valor-P	–	1.024e-09 >
Nguyen-3	Mediana	1.090e-06	1.677e-11
	Valor-P	–	3.173e-02 =
Nguyen-4	Mediana	5.138e-06	2.902e-02
	Valor-P	–	1.087e-09 >
Nguyen-5	Mediana	4.420e-07	1.654e-01
	Valor-P	–	7.557e-10 >
Nguyen-6	Mediana	3.870e-05	9.120e-12
	Valor-P	–	1.245e-04 <
Nguyen-7	Mediana	5.421e-08	1.536e-01
	Valor-P	–	7.557e-10 >
Nguyen-8	Mediana	1.127e-06	1.701e-01
	Valor-P	–	9.068e-10 >
Nguyen-9	Mediana	9.212e-06	3.671e+00
	Valor-P	–	3.451e-08 >
Nguyen-10	Mediana	4.064e-01	3.312e+00
	Valor-P	–	1.339e-08 >

Tabela 24 – Treinamento usando funções do *benchmark* Keijzer: comparação entre o MRX e a recombinação de subárvores.

Função	Estatística	GP-MRX	GP
Keijzer-1	Mediana	5.880e-02	1.099e+00
	Valor-P	–	7.557e-10 >
Keijzer-2	Mediana	4.321e+00	6.706e+00
	Valor-P	–	8.031e-10 >
Keijzer-3	Mediana	1.356e+01	1.680e+01
	Valor-P	–	3.175e-09 >
Keijzer-4	Mediana	1.895e+01	4.134e+01
	Valor-P	–	7.557e-10 >
Keijzer-5	Mediana	3.315e+01	5.679e+01
	Valor-P	–	4.033e-04 >
Keijzer-6	Mediana	3.934e-06	2.998e+00
	Valor-P	–	7.557e-10 >
Keijzer-7	Mediana	2.520e-05	1.078e+01
	Valor-P	–	7.557e-10 >
Keijzer-8	Mediana	2.294e-11	2.363e-11
	Valor-P	–	7.557e-10 >
Keijzer-9	Mediana	1.075e-04	9.044e+00
	Valor-P	–	7.557e-10 >
Keijzer-10	Mediana	9.966e-01	1.011e+01
	Valor-P	–	7.557e-10 >
Keijzer-11	Mediana	6.566e+00	9.598e+00
	Valor-P	–	2.218e-08 >
Keijzer-12	Mediana	7.634e+00	2.838e+01
	Valor-P	–	4.013e-09 >
Keijzer-13	Mediana	2.000e+01	3.230e+01
	Valor-P	–	1.876e-08 >
Keijzer-14	Mediana	5.996e-01	6.157e+00
	Valor-P	–	7.557e-10 >
Keijzer-15	Mediana	1.547e+00	1.288e+01
	Valor-P	–	7.557e-10 >

Tabela 25 – Treinamento usando funções do *benchmark* Vladislavleva: comparação entre o MRX e a recombinação de subárvores.

Função	Estatística	GP-MRX	GP
Vladislavleva-1	Mediana	4.135e+00	6.647e+00
	Valor-P	–	8.534e-10 >
Vladislavleva-2	Mediana	4.010e+00	1.916e+01
	Valor-P	–	7.557e-10 >
Vladislavleva-3	Mediana	3.261e+02	3.674e+02
	Valor-P	–	1.227e-09 >
Vladislavleva-4	Mediana	5.904e+01	1.360e+02
	Valor-P	–	7.557e-10 >
Vladislavleva-5	Mediana	2.162e+01	7.481e+01
	Valor-P	–	7.557e-10 >
Vladislavleva-6	Mediana	2.364e+01	6.256e+01
	Valor-P	–	7.557e-10 >
Vladislavleva-7	Mediana	3.005e+02	6.331e+02
	Valor-P	–	7.557e-10 >
Vladislavleva-8	Mediana	1.246e+01	4.639e+01
	Valor-P	–	7.557e-10 >

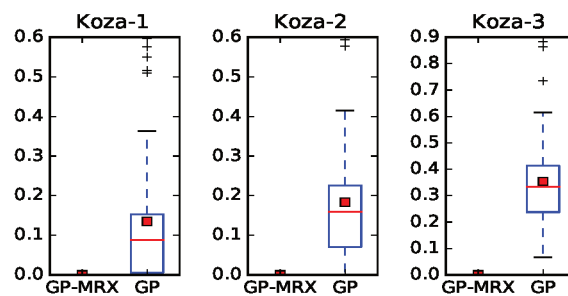


Figura 28 – Dispersão dos erros das respostas fornecidas no treinamento das funções do *benchmark* Koza: comparação entre o MRX e a recombinação de subárvores.

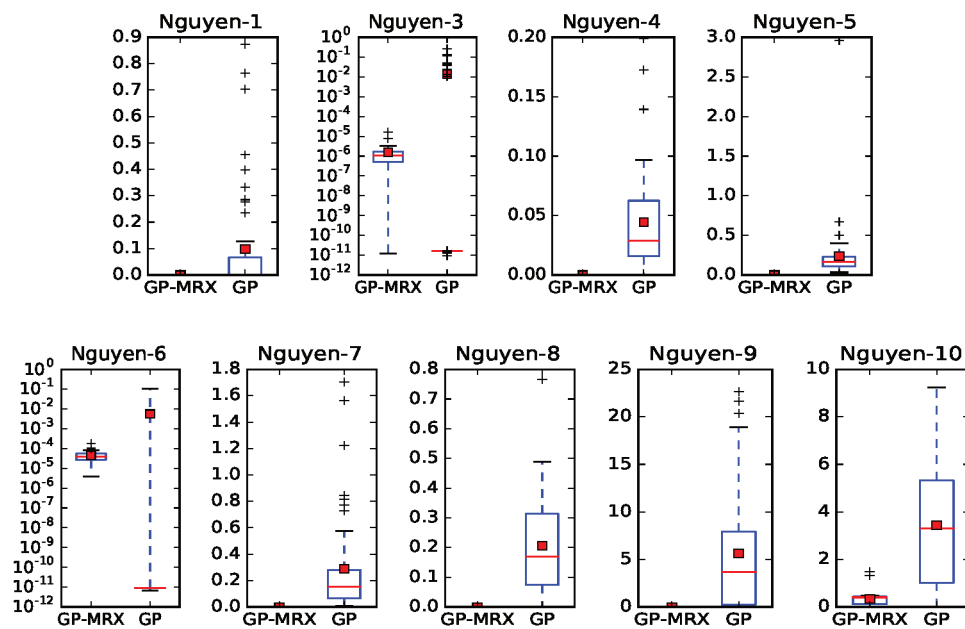


Figura 29 – Dispersão dos erros das respostas fornecidas no treinamento das funções do *benchmark* Nguyen: comparação entre o MRX e a recombinação de subárvores.

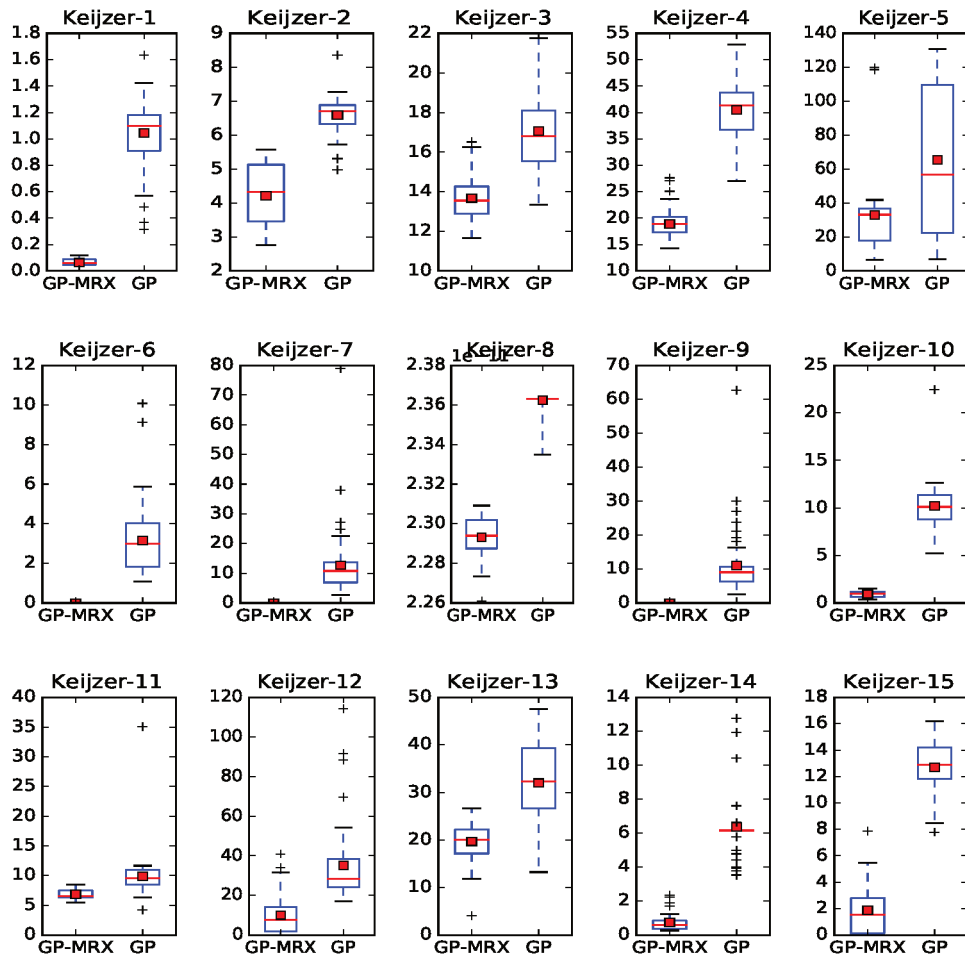


Figura 30 – Dispersão dos erros das respostas fornecidas no treinamento das funções do *benchmark* Keijzer: comparação entre o MRX e a recombinação de subárvores.

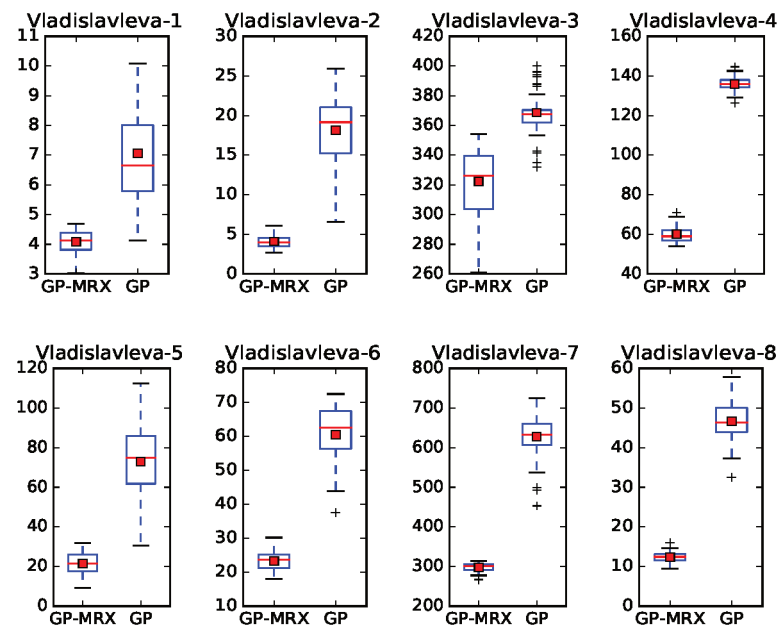


Figura 31 – Dispersão dos erros das respostas fornecidas no treinamento das funções do *benchmark* Vladislavleva: comparação entre o MRX e a recombinação de subárvores.

4.3.3 Teste das melhores respostas

Após encontrar os melhores indivíduos na fase de treinamento, esses foram submetidos os conjuntos de teste. Os resultados dos erros obtidos podem ser vistos na tabelas 26, 27, 28 e 29.

Os resultados dos testes foram coerentes com o que foi verificado no treinamento, com a diferença de que mais funções foram consideradas estatisticamente iguais pelo teste de hipóteses: Nguyen-1 e Nugyen-3 (Tabela 27), Keijzer-1, Keijzer-11, Keijzer-12, Keijzer-13 e Keijzer-14 (Tabela 28), Vladislavleva-3, Vladislavleva-4 e Vladislavleva-8 (Tabela 29). A GP mostrou-se melhor para generalizar a Nguyen-6 (Tabela 27), que já tinha obtido melhores resultados no treinamento, e também na Keijzer-8 (Tabela 28), Vladislavleva-1 e Vladislavleva-2 (Tabela 29).

A análise da distribuição dos erros traz interessantes informações adicionais os dados apresentados nas tabelas. Nas funções Nguyen (Figura 33), nota-se que embora os valores de erro máximo sejam maiores no teste do que no treinamento (o que é natural nesse tipo de experimento), os dados de teste e de treinamento distribuíram-se seguindo padrões semelhantes. O mesmo não pode ser dito para as funções Keijzer, cujos gráficos da Figura 34 sugerem que a GP-MRX é mais suscetível a *outliers*, que apareceram com maior frequência nos dois casos em que os métodos foram considerados estatisticamente iguais e também em outros, em que a GP-MRX foi superior – sendo que nestes últimos, a quantidade de pontos discrepantes foi menor. Curiosamente, embora a Keijzer-8 (também na Figura 34) pareça ser mais bem descrita pelo operador MRX, dada sua distribuição compacta em comparação, o resultado do teste de hipóteses apresentado na Tabela 28 dá a melhor colocação para a PG.

Para as funções Vladislavleva, cujos gráficos são apresentados na Figura 35, vemos que embora a GP-MRX tenha apresentado maior quantidade de *outliers* do que o observado na fase de treinamento (Figura 35), seu desempenho (capacidade de generalização) mostrou-se superior à GP.

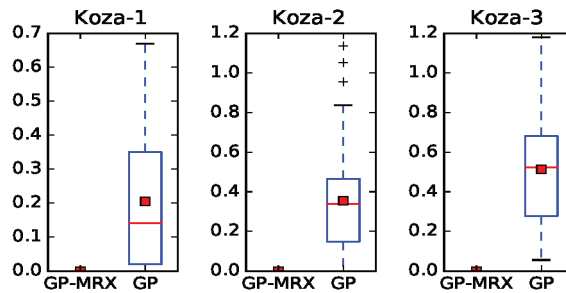


Figura 32 – Dispersão dos erros das respostas fornecidas nos testes das funções do *benchmark* Koza: comparação entre o MRX e a recombinação de subárvores.

Tabela 26 – Resultado dos testes do *benchmark* Koza: comparação entre o MRX e a recombinação de subárvores.

Função	Estatística	GP-MRX	GP
Koza-1	Mediana	4.000e-08	1.416e-01
	Valor-P	–	3.451e-08 >
Koza-2	Mediana	1.811e-08	3.391e-01
	Valor-P	–	8.031e-10 >
Koza-3	Mediana	3.522e-08	5.242e-01
	Valor-P	–	7.557e-10 >

Tabela 27 – Resultado dos testes do *benchmark* Ngueyn: comparação entre o MRX e a recombinação de subárvores.

Função	Estatística	GP-MRX	GP
Nguyen-1	Mediana	1.810e-11	8.243e-12
	Valor-P	–	8.056e-01 =
Nguyen-3	Mediana	3.305e-05	1.493e-11
	Valor-P	–	3.097e-02 =
Nguyen-4	Mediana	1.676e-05	9.361e-02
	Valor-P	–	1.155e-09 >
Nguyen-5	Mediana	9.936e-07	2.245e-01
	Valor-P	–	7.557e-10 >
Nguyen-6	Mediana	7.469e-04	8.977e-12
	Valor-P	–	1.245e-04 <
Nguyen-7	Mediana	1.019e-05	3.539e-01
	Valor-P	–	7.557e-10 >
Nguyen-8	Mediana	4.540e-05	1.530e-01
	Valor-P	–	9.068e-10 >
Nguyen-9	Mediana	1.475e-05	3.198e+00
	Valor-P	–	3.451e-08 >
Nguyen-10	Mediana	4.921e-01	4.279e+00
	Valor-P	–	1.876e-08 >

Tabela 28 – Resultados dos testes do *benchmark* Keijzer: comparação entre o MRX e a recombinação de subárvores.

Função	Estatística	GP-MRX	GP
Keijzer-1	Mediana	6.831e+01	1.301e+02
	Valor-P	–	8.431e-01 =
Keijzer-2	Mediana	4.851e+02	6.986e+02
	Valor-P	–	5.339e-08 >
Keijzer-3	Mediana	1.487e+03	1.739e+03
	Valor-P	–	4.338e-07 >
Keijzer-4	Mediana	1.898e+01	4.132e+01
	Valor-P	–	7.557e-10 >
Keijzer-5	Mediana	3.547e+02	6.099e+02
	Valor-P	–	5.198e-04 >
Keijzer-6	Mediana	2.094e-02	2.520e+01
	Valor-P	–	7.557e-10 >
Keijzer-7	Mediana	5.144e-04	1.034e+02
	Valor-P	–	7.557e-10 >
Keijzer-8	Mediana	2.538e-10	2.495e-10
	Valor-P	–	1.417e-08 <
Keijzer-9	Mediana	3.085e-03	8.817e+01
	Valor-P	–	7.557e-10 >
Keijzer-10	Mediana	1.369e+02	1.007e+03
	Valor-P	–	7.557e-10 >
Keijzer-11	Mediana	2.721e+05	2.371e+05
	Valor-P	–	9.024e-02 =
Keijzer-12	Mediana	5.673e+05	8.409e+05
	Valor-P	–	6.535e-01 =
Keijzer-13	Mediana	1.071e+06	9.291e+05
	Valor-P	–	5.176e-02 =
Keijzer-14	Mediana	5.648e+04	2.088e+05
	Valor-P	–	1.159e-02 =
Keijzer-15	Mediana	1.040e+05	4.662e+05
	Valor-P	–	5.307e-07 >

Tabela 29 – Resultado dos testes do *benchmark* Vladislavleva: comparação entre o MRX e a recombinação de subárvores.

Função	Estatística	GP-MRX	GP
Vladislavleva-1	Mediana	4.869e+02	1.679e+02
	Valor-P		1.851e-03 <
Vladislavleva-2	Mediana	1.466e+01	3.912e+01
	Valor-P		6.110e-06 >
Vladislavleva-3	Mediana	2.699e+03	2.731e+03
	Valor-P		3.876e-01 =
Vladislavleva-4	Mediana	3.829e+02	6.038e+02
	Valor-P		1.660e-01 =
Vladislavleva-5	Mediana	3.063e+02	1.023e+03
	Valor-P		7.557e-10 >
Vladislavleva-6	Mediana	1.542e+09	4.479e+05
	Valor-P		1.285e-03 <
Vladislavleva-7	Mediana	1.077e+03	2.439e+03
	Valor-P		2.638e-03 >
Vladislavleva-8	Mediana	1.106e+03	1.422e+03
	Valor-P		9.807e-01 =

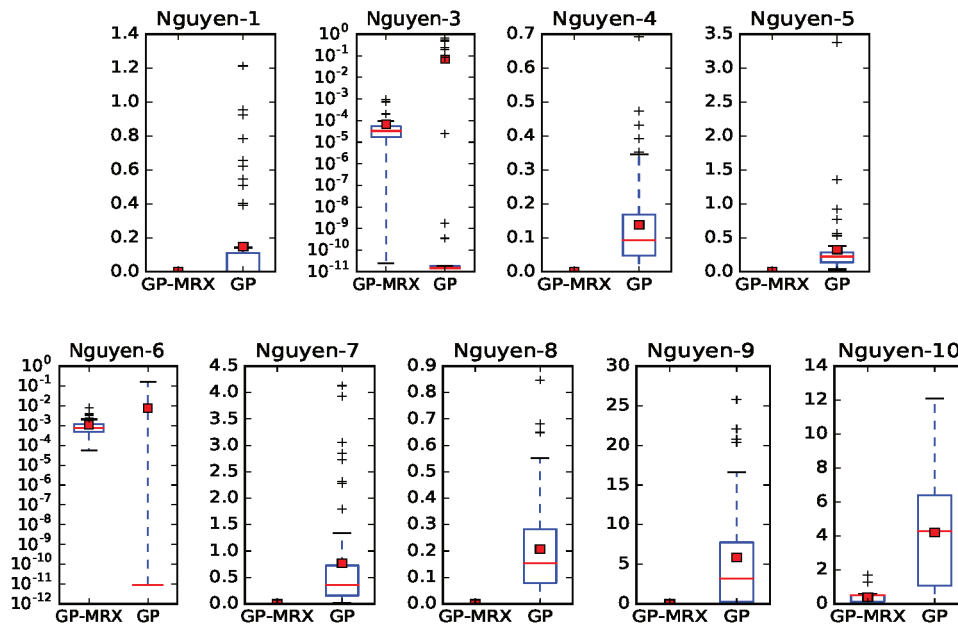


Figura 33 – Dispersão dos erros das respostas fornecidas nos testes das funções do *benchmark* Nguyen: comparação entre o MRX e a recombinação de subárvores.

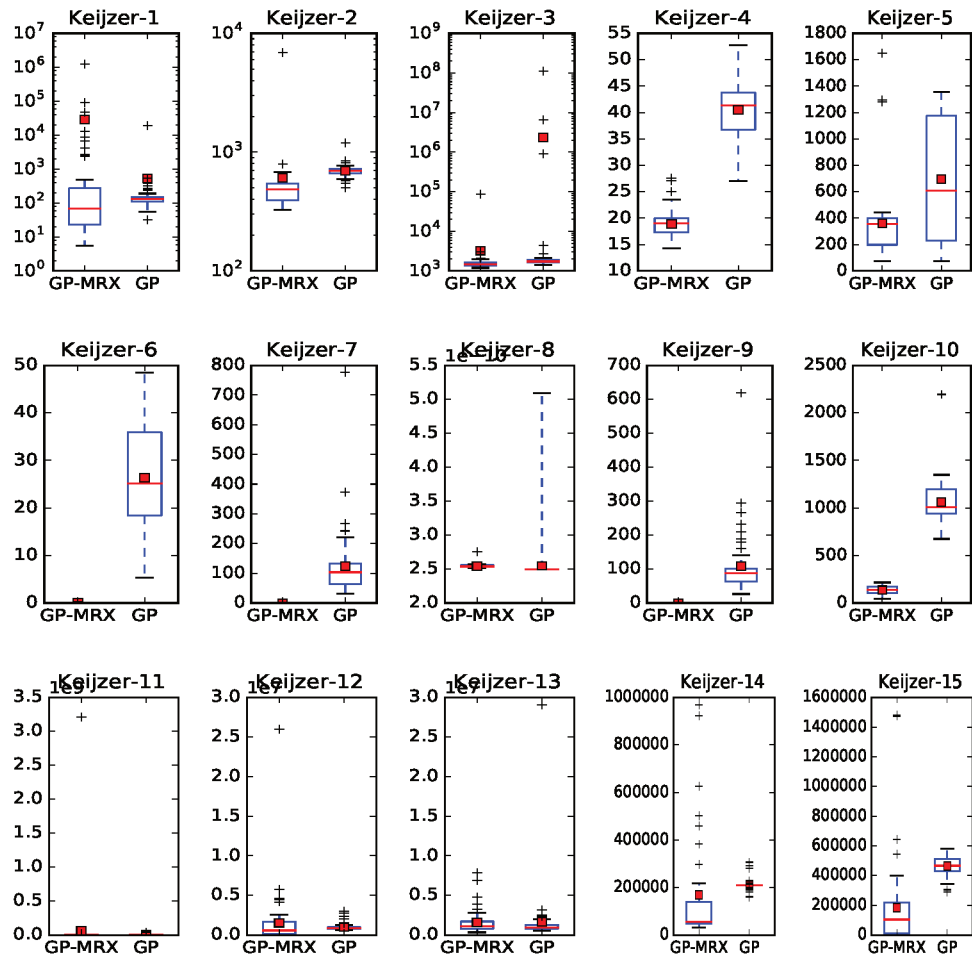


Figura 34 – Dispersão dos erros das respostas fornecidas nos testes das funções do *benchmark* Keijzer: comparação entre o MRX e a recombinação de subárvores.

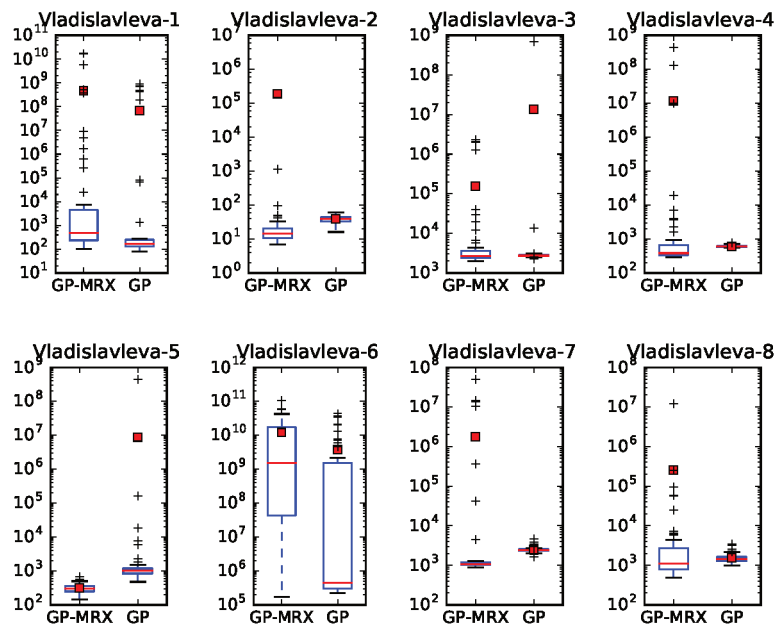


Figura 35 – Dispersão dos erros das respostas fornecidas nos testes das funções do *benchmark* Vladislavleva: comparação entre o MRX e a recombinação de subárvores.

4.3.4 Tamanho das melhores respostas

As tabelas 30, 31, 32 e 33 exibem as informações referentes aos tamanhos (quantidades de nós) observados nas melhores respostas fornecidas pelos experimentos para as funções dos grupos Koza, Nguenyn, Keijzer e Vladislavleva, respectivamente. A GP-MRX forneceu respostas mais compactas em 21 das 35 funções analisadas. A GP forneceu melhores resultados em 10 das 14 funções restantes, enquanto as demais não apresentaram diferenças estatísticas significativas.

O que se percebe de maneira geral é que estruturas sintaticamente mais simples, como as funções Keijzer-6, Keijzer-7, Keijzer-8, Keijzer-9 e Keijzer-10 (Tabela 32) são melhor descritas pela GP. São funções compactas e que não apresentam constantes em sua formação, podendo até mesmo terem sido geradas nas primeiras gerações da GP, antes da população sofrer os efeitos do *bloat*, ou ao acaso, em alguma geração mais adiantada. É importante atentar, porém, que mesmo a GP tendo gerado indivíduos mais compactos, a verificação dos erros no treinamento (Tabela 24) e no teste (Tabela 28) mostram que capacidade de convergência da GP-MRX ainda é maior.

A distribuição dos tamanhos dos indivíduos gerados indicam também a superioridade da GP-MRX. Como pode ser visto nos gráficos das figuras 36, 37, 38 e 39, a dispersão dos indivíduos gerados pelo operador MRX foi sempre menor e quase totalmente livre de *outliers*.

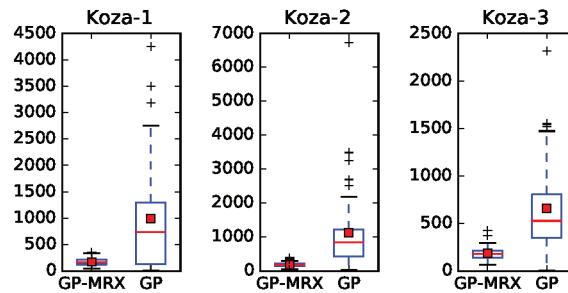


Figura 36 – Dispersão dos tamanhos das respostas fornecidas no treinamento das funções do *benchmark* Koza: comparação entre o MRX e a recombinação de subárvores.

Tabela 30 – Quantidade de nós observados nas funções Koza: comparação entre o MRX e a recombinação de subárvores.

Função	Estatística	GP-MRX	GP
Koza-1	Mediana	168	754
	Valor-P	–	2.3e-07 >
Koza-2	Mediana	183	855
	Valor-P	–	8.0e-10 >
Koza-3	Mediana	183	536
	Valor-P	–	1.7e-09 >

Tabela 31 – Quantidade de nós observados nas funções Nguyen: comparação entre o MRX e a recombinação de subárvores.

Função	Estatística	GP-MRX	GP
Nguyen-1	Mediana	109	11
	Valor-P	–	1.7e-01 =
Nguyen-3	Mediana	198	17
	Valor-P	–	4.9e-01 =
Nguyen-4	Mediana	213	1156
	Valor-P	–	1.0e-09 >
Nguyen-5	Mediana	167	1299
	Valor-P	–	8.5e-10 >
Nguyen-6	Mediana	206	9
	Valor-P	–	1.6e-03 <
Nguyen-7	Mediana	162	1044
	Valor-P	–	1.0e-09 >
Nguyen-8	Mediana	206	1133
	Valor-P	–	9.1e-10 >
Nguyen-9	Mediana	176	492
	Valor-P	–	6.8e-07 >
Nguyen-10	Mediana	131	657
	Valor-P	–	9.1e-10 >

Tabela 32 – Quantidade de nós observados nas funções Keijzer: comparação entre o MRX e a recombinação de subárvores.

Função	Estatística	GP-MRX	GP
Keijzer-1	Mediana	203	525
	Valor-P	–	2.2e-09 >
Keijzer-2	Mediana	124	502
	Valor-P	–	2.0e-09 >
Keijzer-3	Mediana	130	316
	Valor-P	–	3.6e-06 >
Keijzer-4	Mediana	146	393
	Valor-P	–	6.0e-06 >
Keijzer-5	Mediana	90	371
	Valor-P	–	1.4e-09 >
Keijzer-6	Mediana	222	81
	Valor-P	–	1.4e-03 <
Keijzer-7	Mediana	222	74
	Valor-P	–	6.3e-08 <
Keijzer-8	Mediana	77	2
	Valor-P	–	3.0e-08 <
Keijzer-9	Mediana	196	58
	Valor-P	–	7.1e-07 <
Keijzer-10	Mediana	153	80
	Valor-P	–	2.8e-04 <
Keijzer-11	Mediana	154	281
	Valor-P	–	2.0e-06 >
Keijzer-12	Mediana	165	305
	Valor-P	–	2.7e-05 >
Keijzer-13	Mediana	134	284
	Valor-P	–	5.2e-09 >
Keijzer-14	Mediana	116	3
	Valor-P	–	1.4e-08 <
Keijzer-15	Mediana	147	165
	Valor-P	–	8.0e-02 =

Tabela 33 – Quantidade de nós observados nas funções Vladislavleva: comparação entre o MRX e a recombinação de subárvores.

Função	Estatística	GP-MRX	GP
Vladislavleva-1	Mediana	190	353
	Valor-P	–	3.6e-09 >
Vladislavleva-2	Mediana	176	49
	Valor-P	–	4.5e-03 <
Vladislavleva-3	Mediana	118	86
	Valor-P	–	4.1e-01 =
Vladislavleva-4	Mediana	203	75
	Valor-P	–	3.2e-07 <
Vladislavleva-5	Mediana	217	113
	Valor-P	–	4.1e-03 <
Vladislavleva-6	Mediana	213	379
	Valor-P	–	2.7e-09 >
Vladislavleva-7	Mediana	142	373
	Valor-P	–	5.3e-07 >
Vladislavleva-8	Mediana	256	367
	Valor-P	–	2.1e-04 >

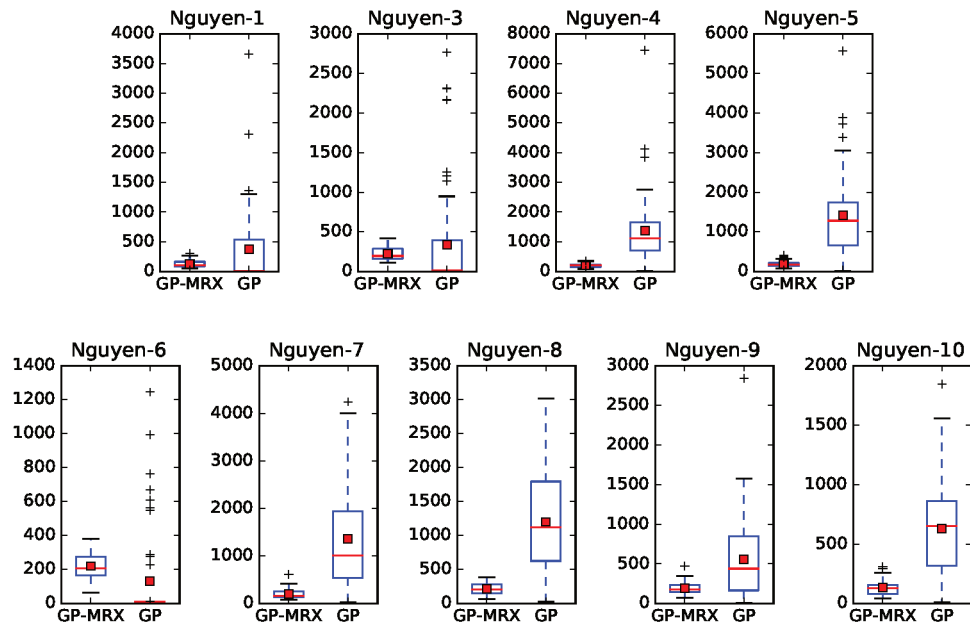


Figura 37 – Dispersão dos tamanhos das respostas fornecidas no treinamento das funções do *benchmark* Nguyen: comparação entre o MRX e a recombinação de subárvores.

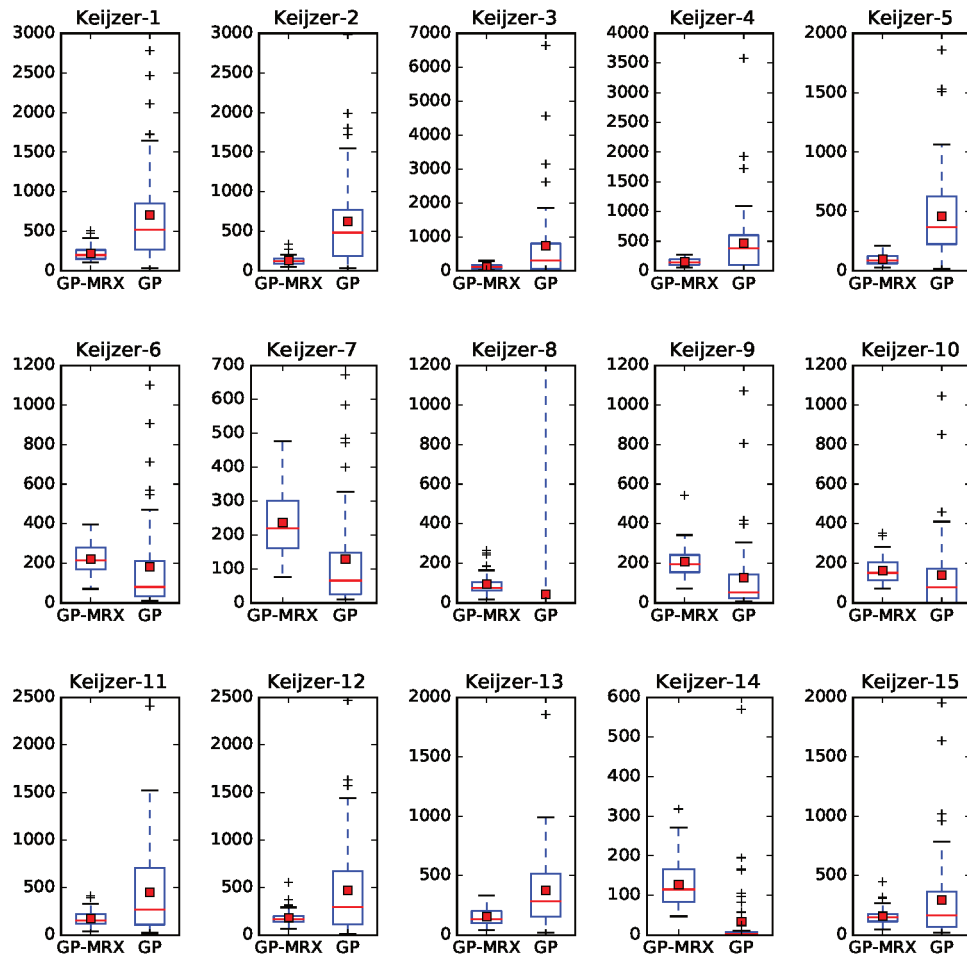


Figura 38 – Dispersão dos tamanho das respostas fornecidas no treinamento das funções do *benchmark* Keijzer: comparação entre o MRX e a recombinação de subárvores.

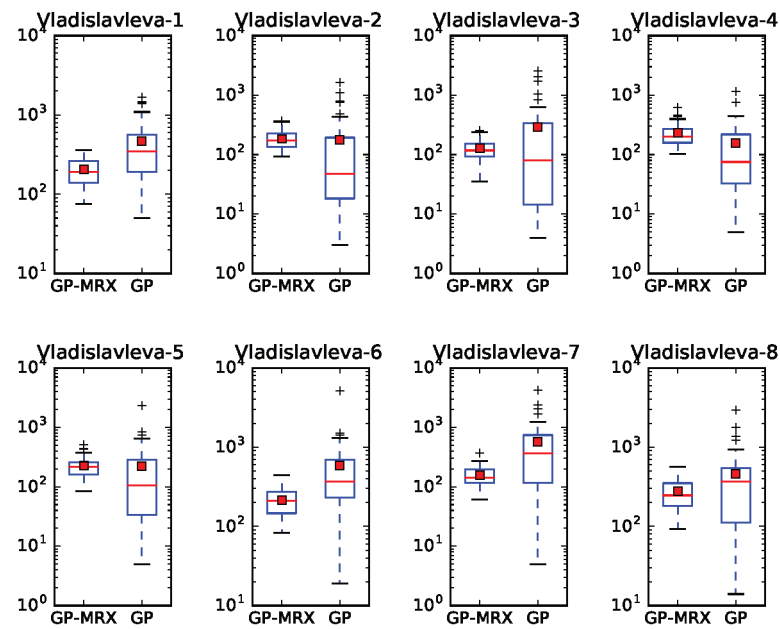


Figura 39 – Dispersão dos tamanhos das respostas fornecidas no treinamento das funções do *benchmark* Vladislavleva.

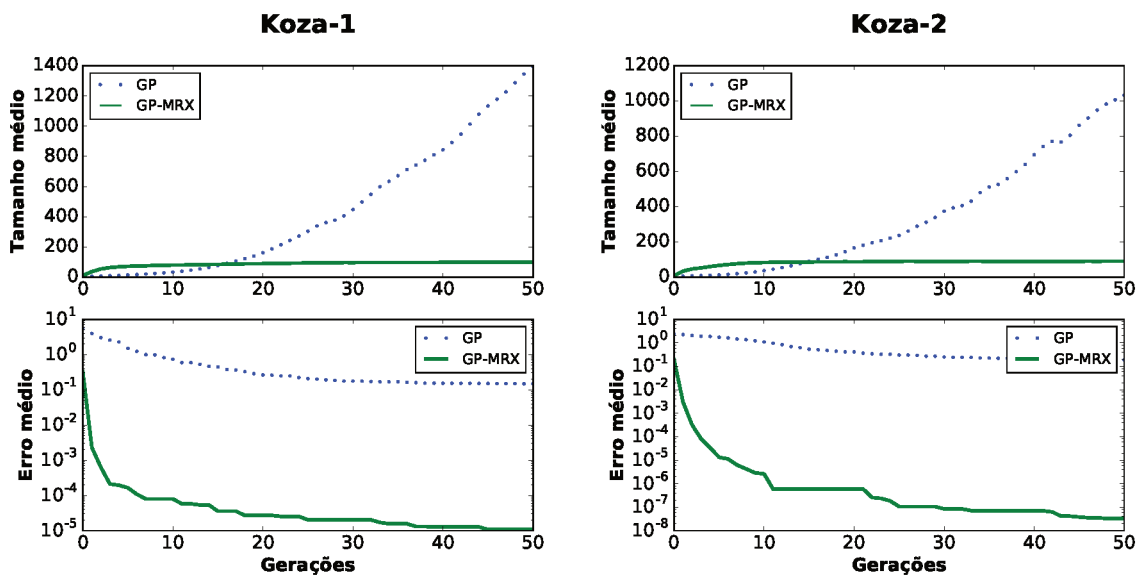
4.3.5 Efeito *bloat*

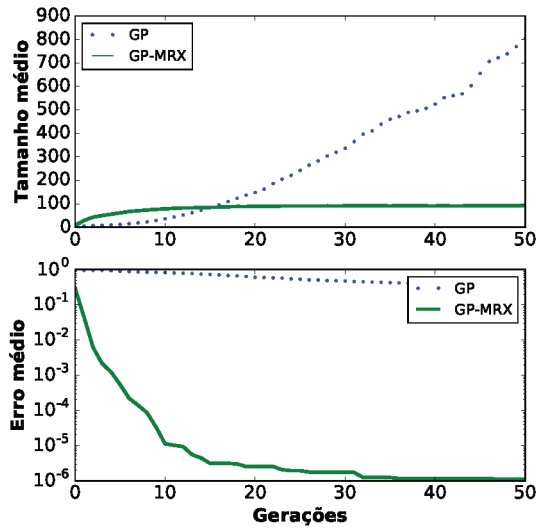
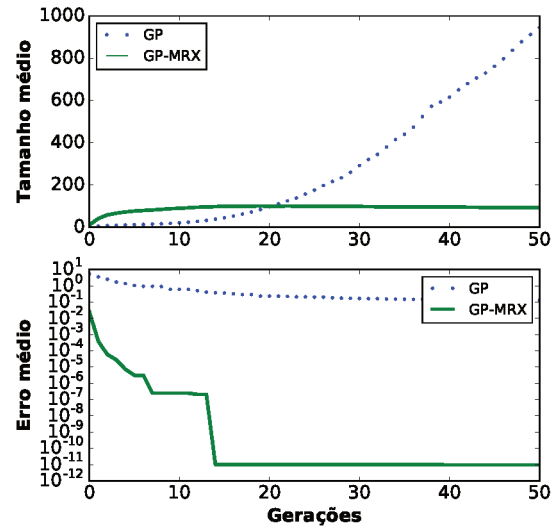
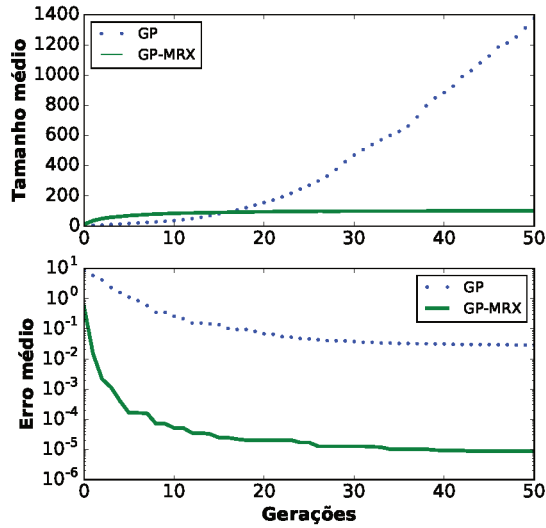
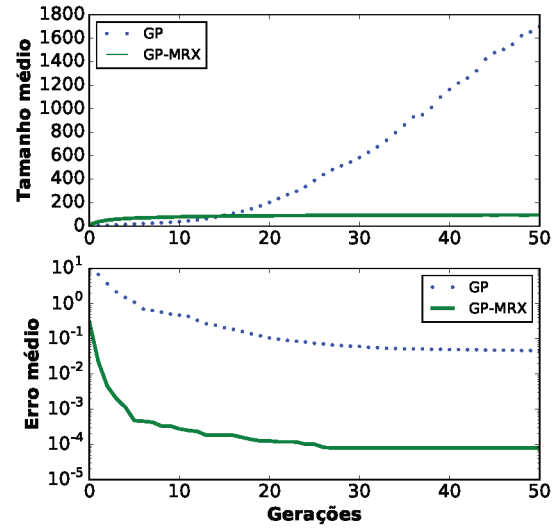
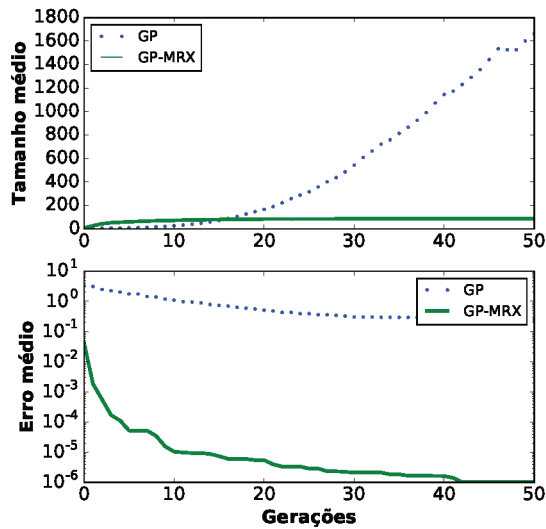
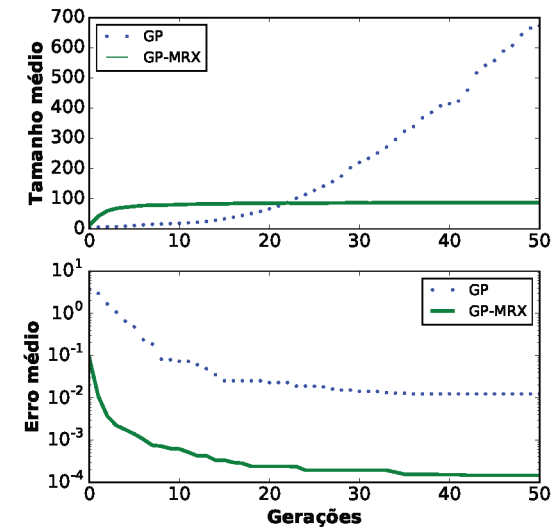
Esta seção apresenta a comparação da influência do *bloat* ao longo das gerações nas duas versões de PG. Os gráficos da Figura 40 indicam que a GP-MRX é melhor do que a GP em 100% dos casos analisados, tanto no que se refere à evolução do tamanho médio dos componentes da população quanto à evolução do erro médio. É importante observar que a maioria dos gráficos de erro foram colocados em escala logarítmica para facilitar a comparação entre os métodos.

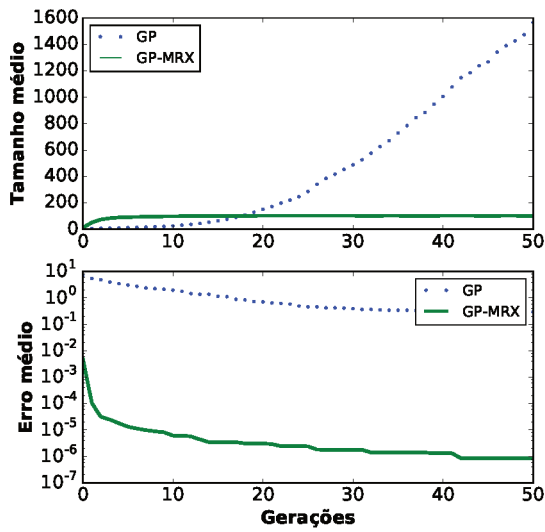
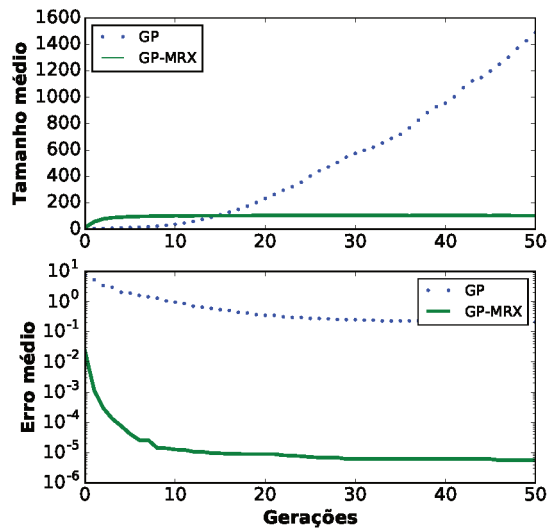
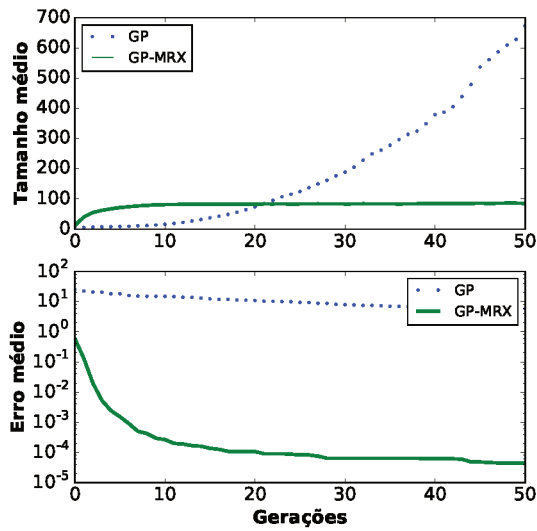
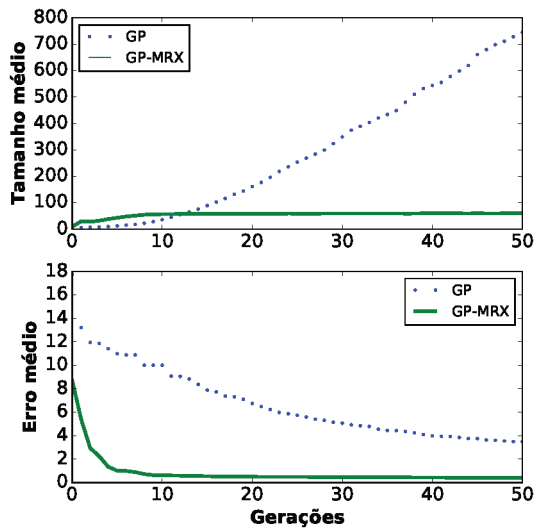
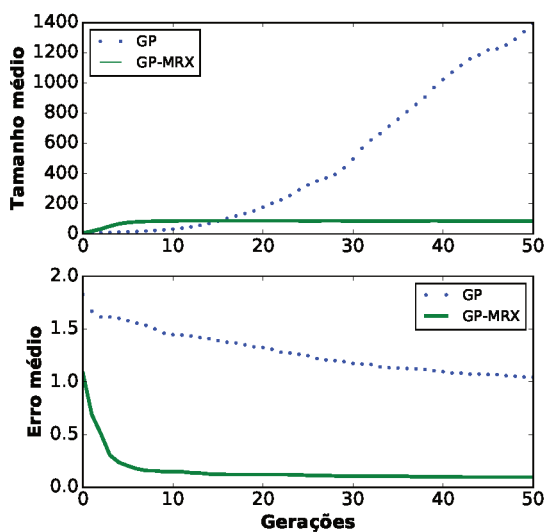
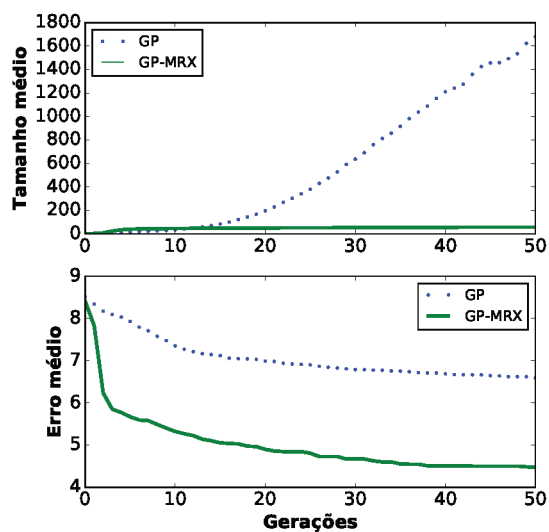
Como tendência geral, vemos que as primeiras gerações são sempre favoráveis à GP, enquanto a GP-MRX apresenta árvores maiores no início da evolução. Entretanto, a abordagem da recombinação MRX faz com que o tamanho médio logo se estabilize, com quase imperceptíveis variações, ao mesmo tempo em que a GP produz crescimento com características exponenciais.

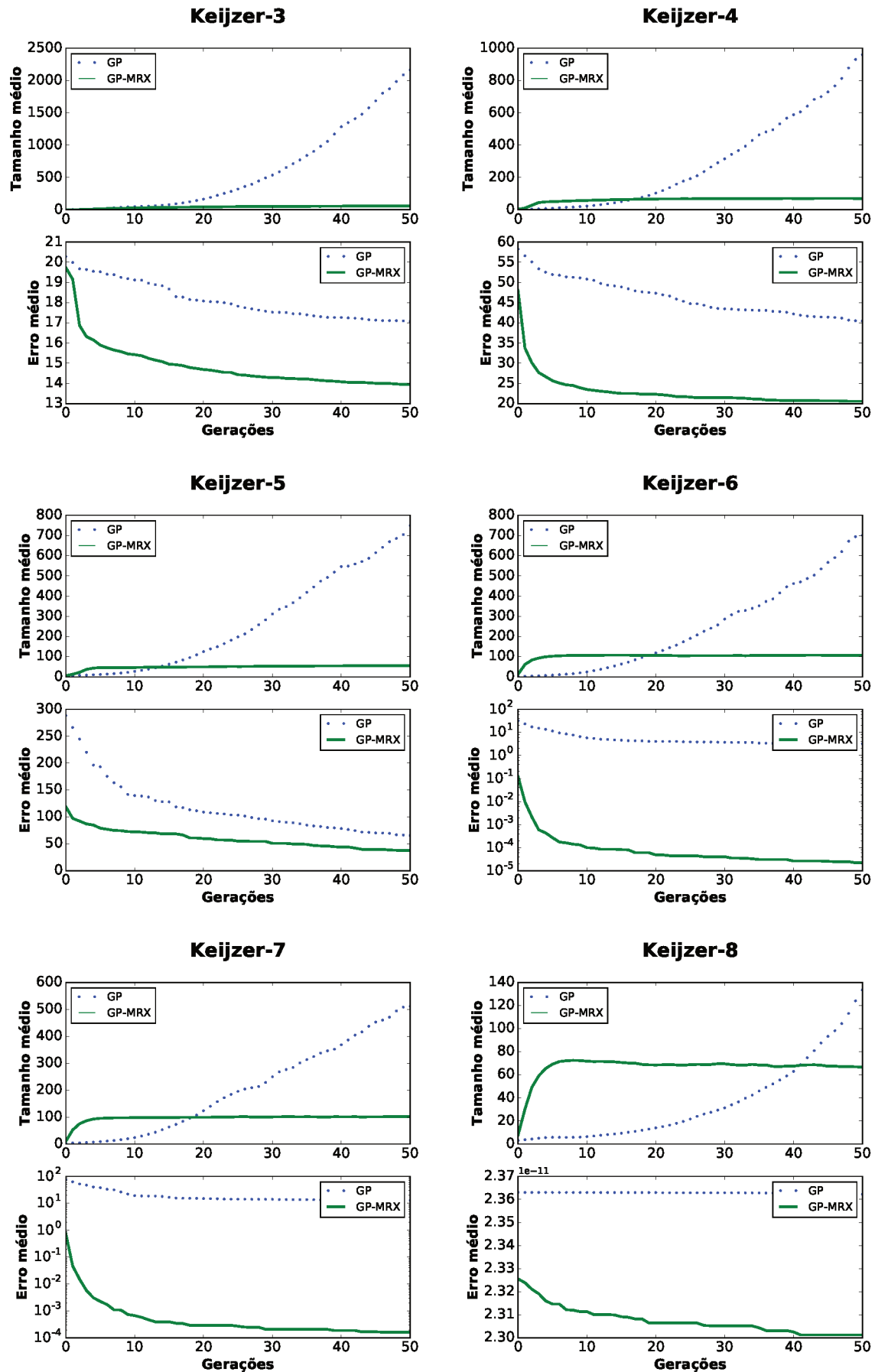
A relação entre o quanto a árvore cresce em média e a forma como o erro médio se comporta é de grande importância para entender as características básicas de cada método. O comportamento dos gráficos de tamanho médio da GP, que parecem se comportar como funções exponenciais, contrastam com o que é visto nos gráficos que na GP-MRX, que parecem mapear o comportamento de uma assíntota.

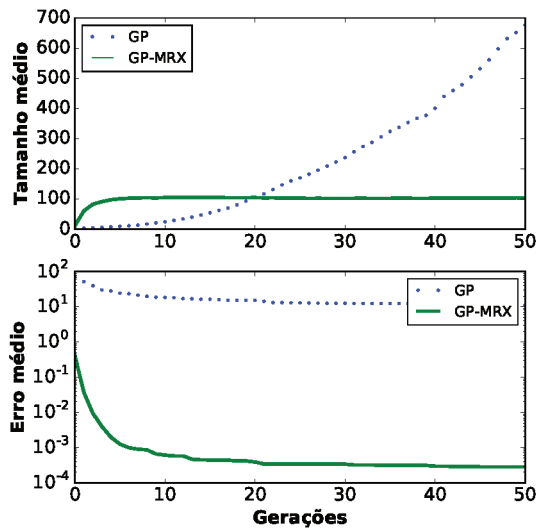
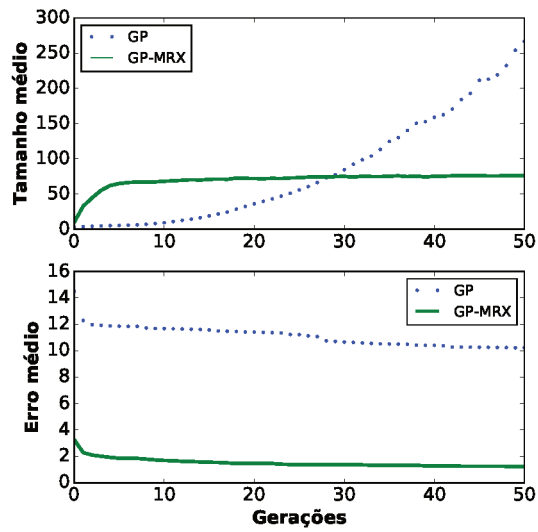
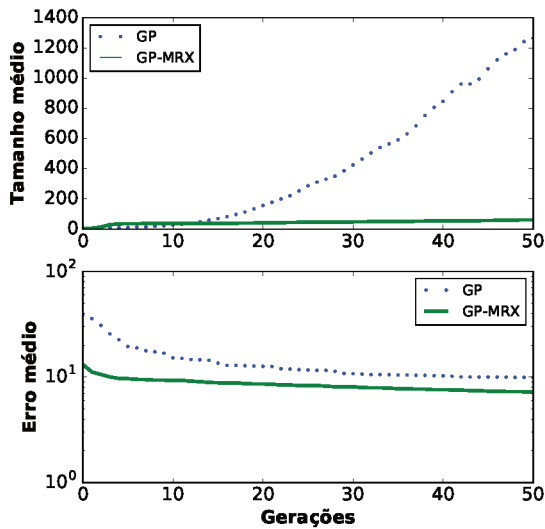
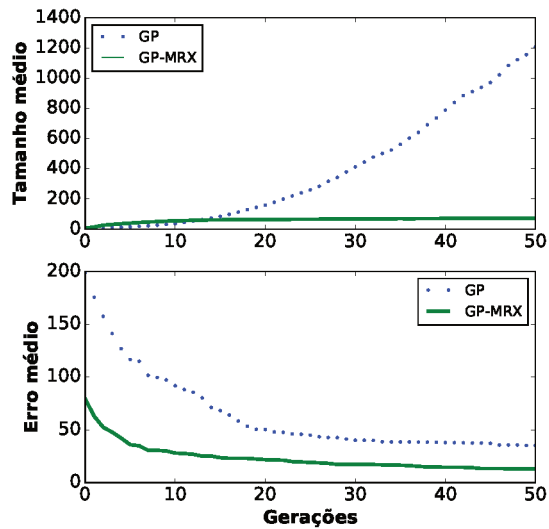
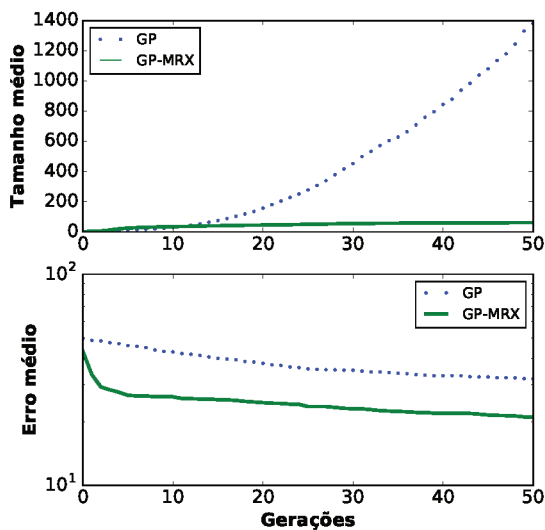
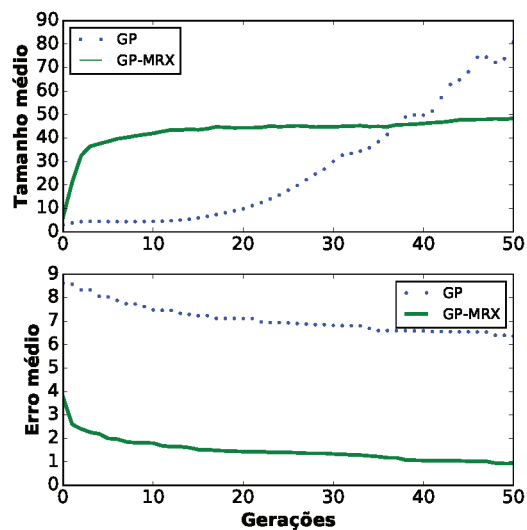
Já os gráficos de erro indicam que a convergência não ocorre prematuramente. A tendência de diminuição do erro continua por toda a evolução com a descoberta de melhores indivíduos. Acreditamos que esse comportamento seja resultado do acréscimo de diversidade causado pela inserção de subárvores aleatórias.

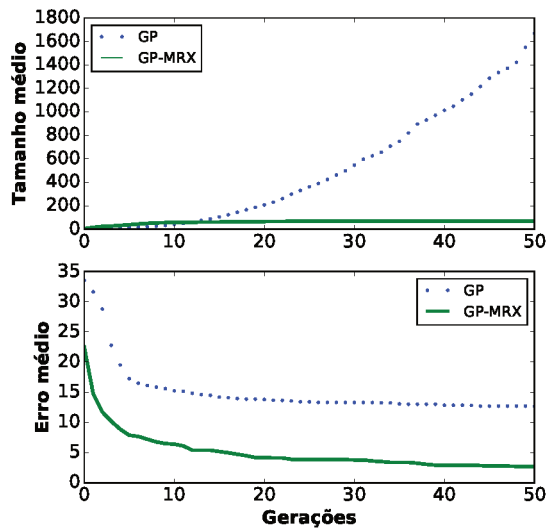
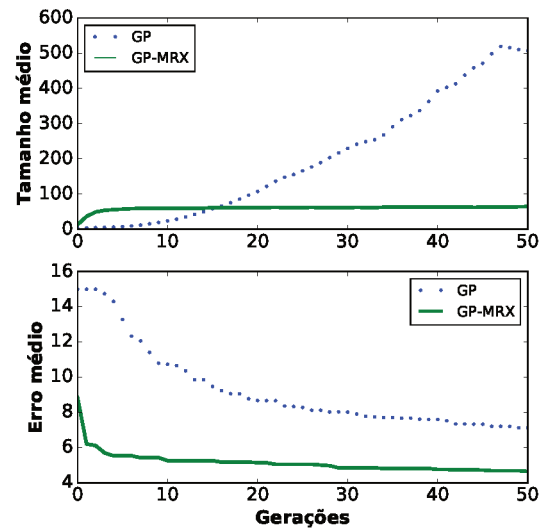
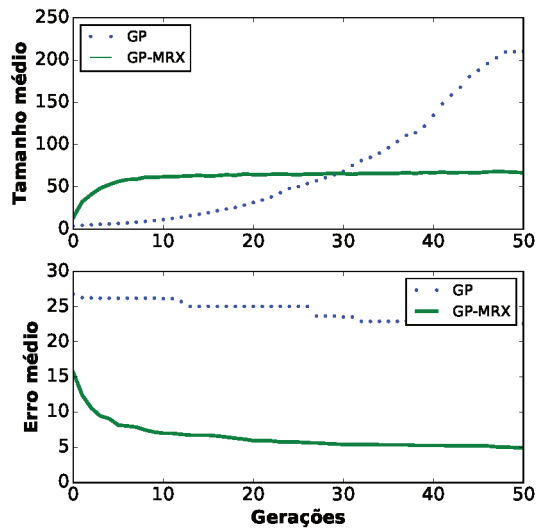
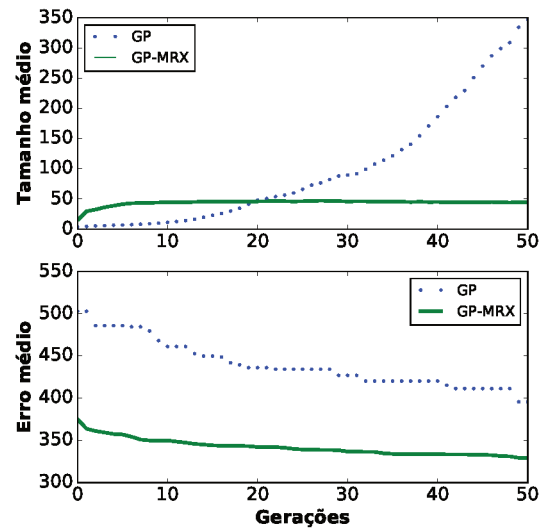
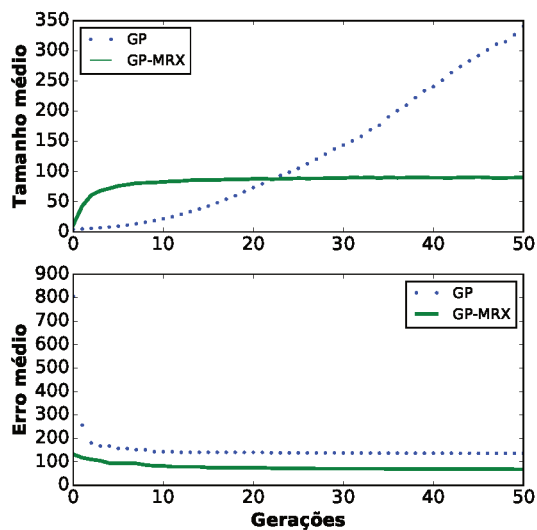
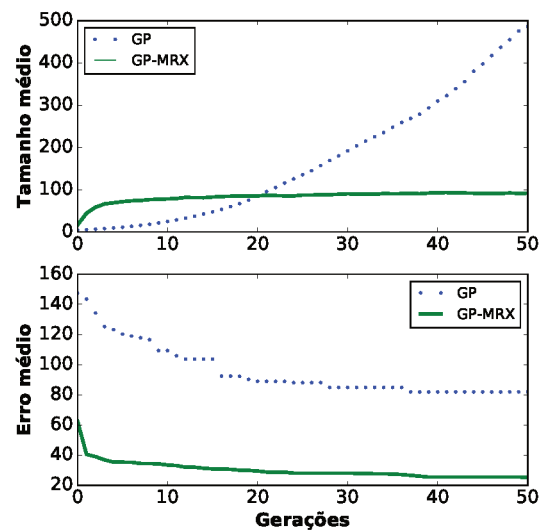


Koza-3**Nguyen-1****Nguyen-3****Nguyen-4****Nguyen-5****Nguyen-6**

Nguyen-7**Nguyen-8****Nguyen-9****Nguyen-10****Keijzer-1****Keijzer-2**



Keijzer-9**Keijzer-10****Keijzer-11****Keijzer-12****Keijzer-13****Keijzer-14**

Keijzer-15**Vladislavleva-1****Vladislavleva-2****Vladislavleva-3****Vladislavleva-4****Vladislavleva-5**

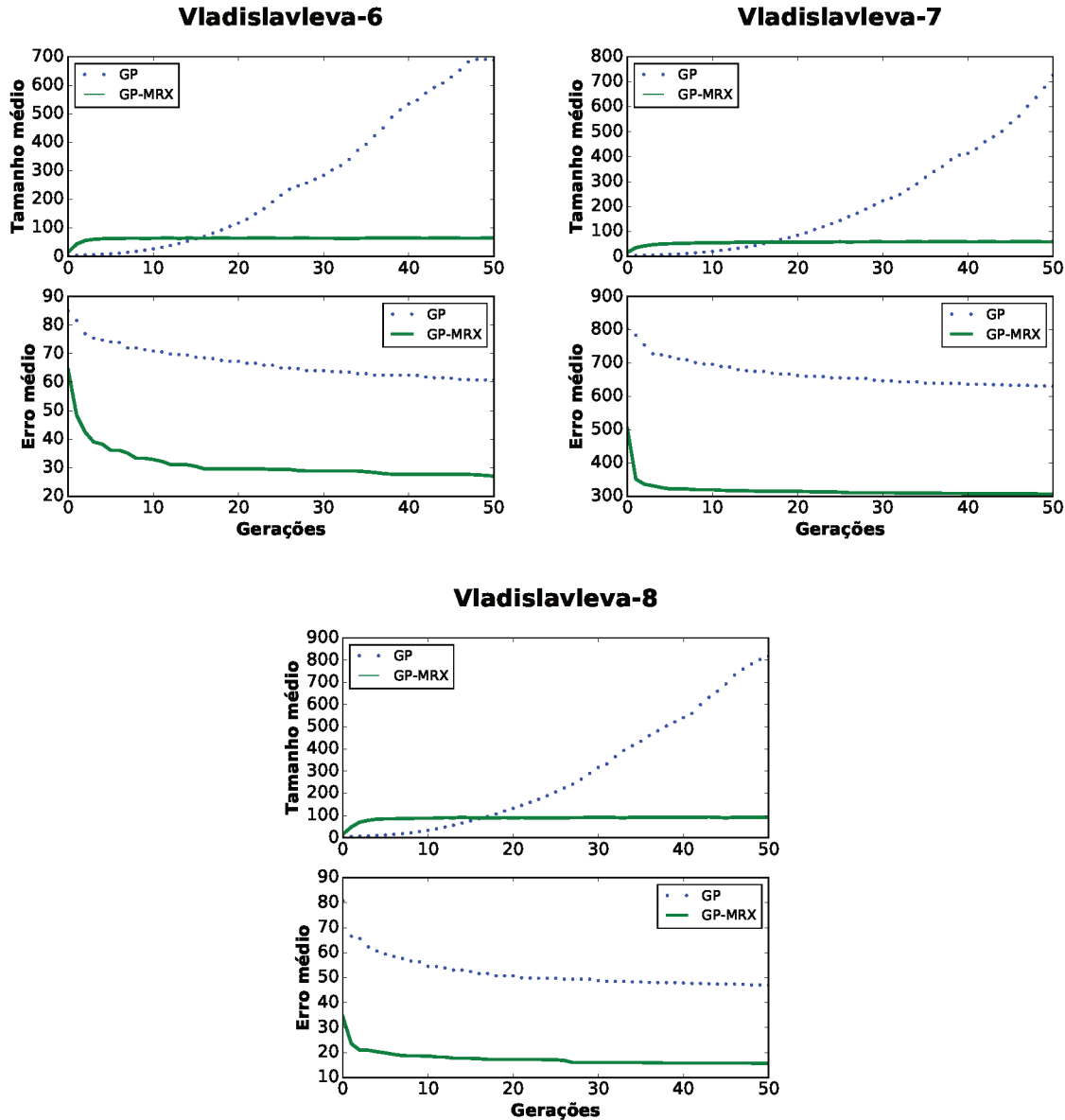


Figura 40 – Efeito *bloat* em funções dos *benchmarks* Koza, Nguyen, Keijzer e Vladislavleva.

4.4 Comparação com operadores semânticos

Esta seção apresenta resultados de experimentos realizados para comparar o operador MRX com três operadores semânticos propostos nos últimos anos. Seguimos aqui os mesmos procedimentos adotados na Seção 4.3, onde apresentamos uma análise guiada pela qualidade (resultados dos treinamentos, testes e quantidade de nós) e pela forma como os métodos reagem ao efeito *bloat*.

4.4.1 Configuração dos experimentos

Os experimentos realizados consistem na comparação dos resultados obtidos pela GP-MRX e por implementações que utilizam os operadores AGX (KRAWIEC; PAWLAK,

2013), RDO (PAWLAK; WIELOCH; KRAWIEC, 2015) e SSGX (NGUYEN et al., 2016). Para os operadores semânticos, utilizamos o código desenvolvido e disponibilizado⁷ por Nguyen et al. (2016). Assim como a GP-MRX, esses programas são escritos em linguagem Java e organizam os resultados em uma série de arquivos texto.

Optamos não utilizar o SGX porque, como discutido na Seção 2.2.5.2, este operador genético resulta em árvores excessivamente grandes, o que dificultaria a representação gráfica do efeito *bloat* e comparação com outros operadores. Ademais, os operadores semânticos selecionados para análise foram originalmente desenvolvidos com o principal objetivo de reduzir o crescimento de código observado no SGX (NGUYEN et al., 2016).

As aplicações foram executadas 50 vezes, sendo colhidas informações referentes à soma dos erros absolutos (SAE) tando ao final do treinamento quanto do teste do melhor indivíduo. Informações relativas ao tamanho (número de nós observados nas melhores respostas) também foram armazenadas.

Aos resultados obtidos, aplicamos o teste de Wilcoxon, com significância de 1%. Para explicitar a relação estatística dos erros resultantes das implementações que usam os operadores semânticos, acrescentamos às tabelas informações no formato de sinais, onde “=” indica que a distribuição de erros entres as versões são estatisticamente iguais, enquanto “<” e “>” indicam que o erro observado na operação semântica é estatisticamente menor e maior, respectivamente, do que o observado na MRX.

Os programas foram executados de acordo com os seguintes parâmetros:

- Tamanho da população: 500;
- Taxa de recombinação: 90% para as aplicações com AGX, SSGX e RDO; 100% para a aplicação com MRX;
- Taxa de mutação: 5% para as aplicações com AGX, SSGX e RDO; 0% para a aplicação com MRX;
- Quantidade de gerações: 250;
- Torneio: 5;
- Profundidade inicial: 6;
- Constantes: não utilizadas;
- Funções: +, −, ×, % (divisão protegida), sin, cos e exp.

A forma como foram conduzidos os experimentos desta seção diferencia-se, por algumas razões, dos procedimentos adotados na Seção 4.3. Optamos por executar os

⁷ Disponível em <<https://github.com/jmmcd/GP-SSGX>>.

algoritmos por 250 gerações (ao invés de 51) com o objetivo de obter mais informações sobre seus comportamentos e tendências. Além disso, não utilizamos as funções e constantes sugeridas na Tabela 9, uma vez que optamos por não modificar o código da implementação desenvolvida e disponibilizada por Nguyen et al. (2016).

Para melhor investigar as tendências de crescimento dos indivíduos das populações, não foram impostas técnicas explícitas de controle de *bloat* na GP-MRX. A mesma abordagem foi tentada nas aplicações semânticas, porém sem sucesso: já nas primeiras gerações, o excessivo consumo de memória demandado interrompia a execução, impedindo o término do experimento. Por essa razão, modificamos a profundidade máxima aceita⁸ na criação de novos indivíduos nas implementações semânticas, passando de 17 para 30, de forma a diminuir o quanto possível os obstáculos ao crescimento natural dos indivíduos.

Por fim, cabe salientar que realizamos experimentos apenas com os *benchmarks* Koza e Nguyen, já que as funções Keijzer e Vladislavleva exigem, como descrito na Tabela 9, configurações específicas de funções e constantes que a implementação fornecida por Nguyen et al. (2016) não suporta.

4.4.2 Treinamento

Como pode ser visto nas tabelas 34 e 35 o treinamento com o MRX mostrou-se significativamente superior às demais abordagens em nove das doze funções testadas. O único método de recombinação que apresenta resultados competitivos é o RDO, que embora tenha se mostrado superior em três funções, teve a distribuição de seus resultados considerados estatisticamente iguais aos resultados obtidos pelo MRX.

Tabela 34 – Treinamento usando funções do *benchmark* Koza.

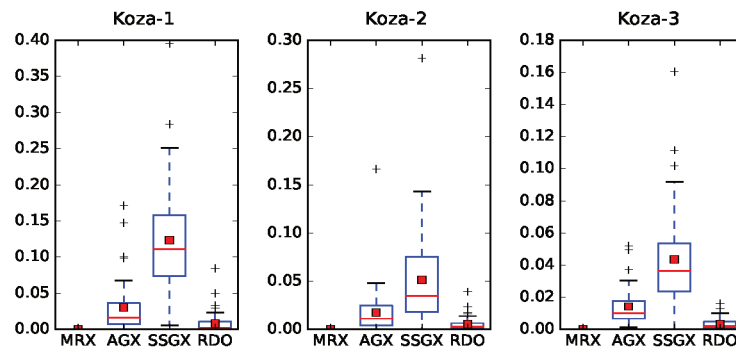
Função	Estatística	MRX	AGX	SSGX	RDO
Koza-1	Mediana	5.909e-12	1.959e-02	1.115e-01	1.798e-03
	Valor-P	–	1.087e-09	7.557e-10	1.865e-09
			>	>	>
Koza-2	Mediana	3.457e-12	1.109e-02	3.440e-02	2.810e-03
	Valor-P	–	7.557e-10	9.068e-10	7.557e-10
			>	>	>
Koza-3	Mediana	2.856e-12	1.091e-02	3.665e-02	2.122e-03
	Valor-P	–	7.557e-10	7.557e-10	7.557e-10
			>	>	>

Os dados de dispersão dos valores das respostas obtidos nos treinamentos, exibidos nos gráficos das figuras 41 e 42 indicam que o MRX produz resultados mais estáveis do que os outros três métodos. Valores máximos elevados e grande quantidade de *outliers* foram comuns nos treinamentos do AGX, SSGX e RDO, aparecendo todas as funções.

⁸ Este é um dos métodos mais básicos de controle de *bloat*: se um filho gerado exceder o valor de profundidade máxima permitido, ele será descartado e um de seus pais é selecionado para compor a próxima geração em seu lugar. Mais detalhes são descritos no Apêndice A.

Tabela 35 – Treinamento usando funções do *benchmark* Nguyen.

Função	Estatística	MRX	AGX	SSGX	RDO
Nguyen-1	Mediana	6.569e-12	3.852e-03	9.279e-02	1.108e-11
	Valor-P	–	1.024e-09 >	7.557e-10 >	6.110e-06 >
Nguyen-3	Mediana	1.951e-07	2.272e-02	1.474e-01	2.998e-03
	Valor-P	–	7.557e-10 >	8.031e-10 >	8.031e-10 >
Nguyen-4	Mediana	4.024e-07	1.771e-02	1.113e-01	2.203e-03
	Valor-P	–	7.557e-10 >	7.557e-10 >	7.557e-10 >
Nguyen-5	Mediana	2.442e-08	1.492e-02	5.336e-02	1.746e-03
	Valor-P	–	8.031e-10 >	7.557e-10 >	9.068e-10 >
Nguyen-6	Mediana	3.539e-06	1.203e-02	7.780e-02	5.129e-04
	Valor-P	–	4.013e-09 >	8.031e-10 >	6.692e-06 >
Nguyen-7	Mediana	3.334e-08	5.473e-03	1.101e-01	8.732e-04
	Valor-P	–	6.381e-09 >	8.534e-10 >	4.013e-09 >
Nguyen-8	Mediana	3.752e-07	1.082e-02	1.401e-01	6.180e-12
	Valor-P	–	1.324e-07 >	7.557e-10 >	5.272e-01 =
Nguyen-9	Mediana	2.606e-07	2.080e-01	4.761e-01	3.673e-11
	Valor-P	–	8.035e-05 >	2.509e-09 >	2.861e-01 =
Nguyen-10	Mediana	1.196e-02	4.565e-01	5.778e-01	4.587e-11
	Valor-P	–	3.851e-08 >	1.865e-09 >	1.689e-01 =

Figura 41 – Dispersão dos erros das respostas fornecidas no treinamento das funções do *benchmark* Koza.

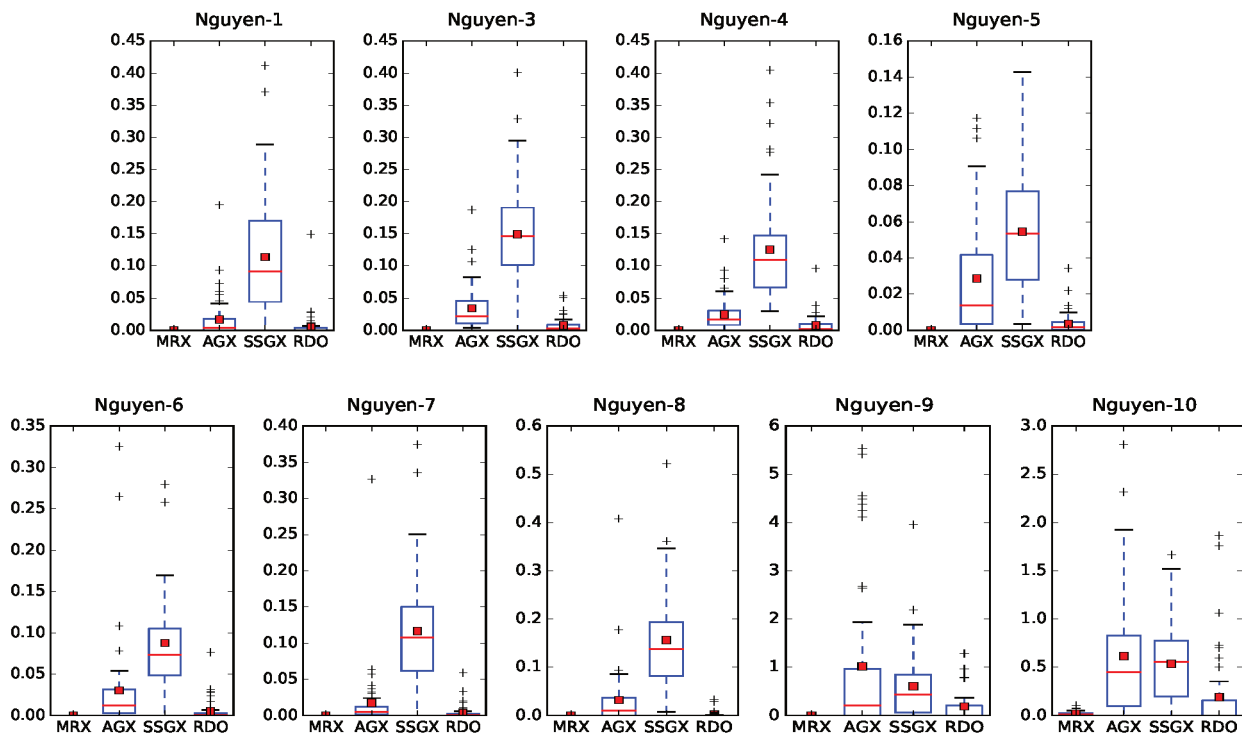


Figura 42 – Dispersão dos erros das respostas fornecidas no treinamento das funções do *benchmark* Nguyen.

4.4.3 Teste das melhores respostas

Os resultado obtidos pelos testes dos melhores indivíduos (tabelas 36 e 37) seguem a mesma tendência observada nos resultados obtidos no treinamento, com o MRX sendo estatisticamente superior em nove das doze funções e não sendo pior em nenhuma. A análise da dispersão dos resultados, porém, revelou importantes características de cada operador, como pode ser visto nos gráficos das figuras 43 e 44. De forma geral, os operadores semânticos apresentaram baixa capacidade de generalização e excessiva quantidade de *outliers*.

Tabela 36 – Resultados dos testes do *benchmark* Koza.

Função	Estatística	MRX	AGX	SSGX	RDO
Koza-1	Mediana	2.325e-11	1.161e-01	2.109e-01	2.859e-02
	Valor-P	–	1.087e-09	7.557e-10	2.365e-09
			>	>	>
Koza-2	Mediana	6.695e-12	4.329e-02	6.255e-02	1.439e-02
	Valor-P	–	7.557e-10	1.087e-09	7.557e-10
			>	>	>
Koza-3	Mediana	5.376e-12	2.319e-02	4.401e-02	5.752e-03
	Valor-P	–	7.557e-10	7.557e-10	7.557e-10
			>	>	>

Tabela 37 – Resultados dos testes do *benchmark* Nguyen.

Função	Estatística	MRX	AGX	SSGX	RDO
Nguyen-1	Mediana	1.958e-11	3.014e-02	1.313e-01	1.141e-04
	Valor-P	–	3.679e-05	1.087e-09	1.197e-04
			>	>	>
Nguyen-3	Mediana	1.805e-05	3.178e-01	4.260e-01	2.103e-01
	Valor-P	–	1.227e-09	9.634e-10	4.254e-09
			>	>	>
Nguyen-4	Mediana	5.388e-06	7.486e-02	1.573e-01	2.966e-02
	Valor-P	–	7.557e-10	7.557e-10	7.557e-10
			>	>	>
Nguyen-5	Mediana	8.265e-08	3.551e-02	7.253e-02	6.137e-03
	Valor-P	–	8.031e-10	7.557e-10	9.068e-10
			>	>	>
Nguyen-6	Mediana	3.230e-04	6.017e-02	1.040e-01	1.412e-02
	Valor-P	–	4.013e-09	8.031e-10	3.351e-06
			>	>	>
Nguyen-7	Mediana	1.266e-05	3.041e-02	1.301e-01	1.308e-02
	Valor-P	–	3.785e-09	8.534e-10	5.064e-09
			>	>	>
Nguyen-8	Mediana	4.877e-05	5.867e-02	2.475e-01	1.124e-11
	Valor-P	–	1.909e-07	7.557e-10	4.602e-01
			>	>	=
Nguyen-9	Mediana	3.533e-07	2.161e-01	4.409e-01	4.317e-11
	Valor-P	–	8.707e-05	2.661e-09	2.905e-01
			>	>	=
Nguyen-10	Mediana	1.611e-02	6.589e-01	5.696e-01	4.456e-11
	Valor-P	–	4.790e-08	2.100e-09	2.184e-01
			>	>	=

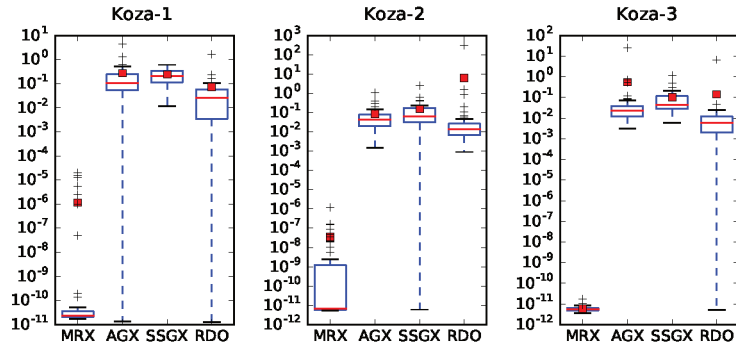


Figura 43 – Dispersão dos erros das respostas fornecidas no teste das funções do *benchmark* Koza.

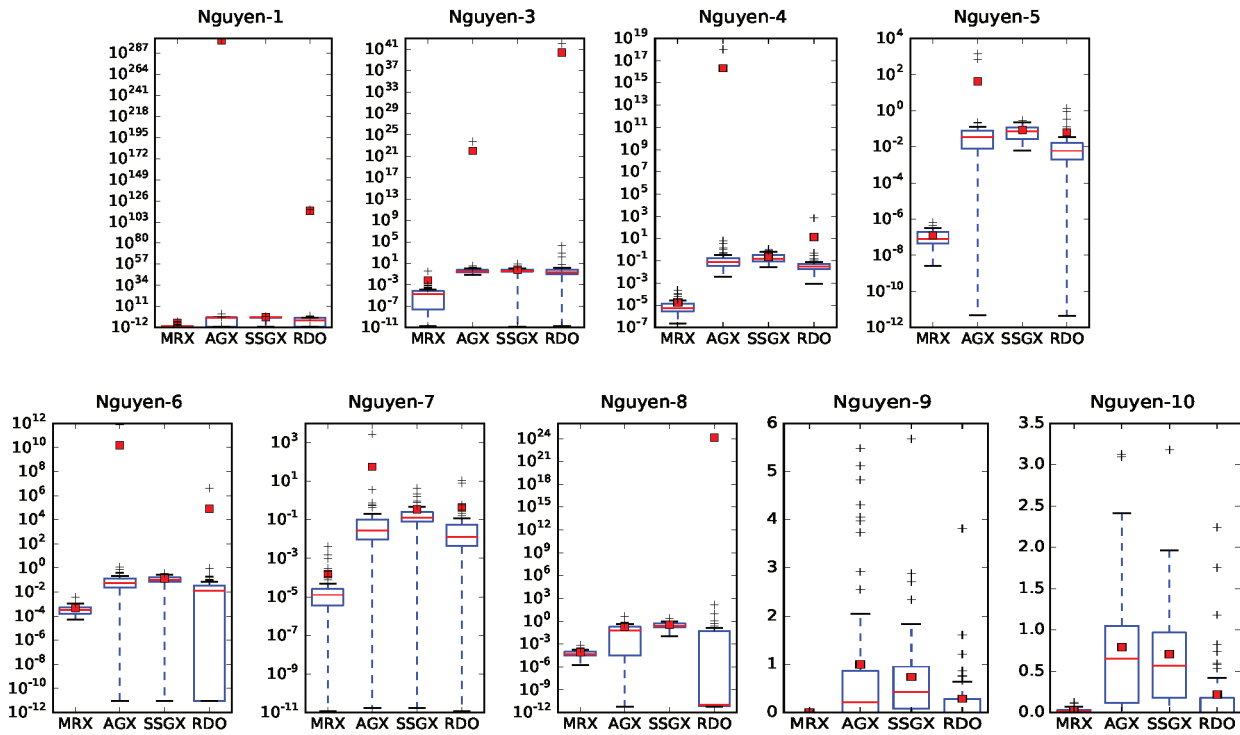


Figura 44 – Dispersão dos erros das respostas fornecidas no teste das funções do *benchmark* Nguyen.

4.4.4 Tamanho das melhores respostas

Como tendência geral, observa-se nas tabelas 38 e 39 que o operador SSGX produziu resultados mais compactos na maioria das funções testadas, quase sempre seguido do MRX. Embora o SSGX tenha apresentado respostas mais compactas do que os outros operadores, seu desempenho no treinamento (tabelas 34 e 35) e também no teste (tabelas 36 e 37) mostraram-se bastante inferiores quando comparados aos demais operadores, principalmente em comparação com o MRX.

Tabela 38 – Quantidade de nós observados nas funções Koza.

Função	Estatística	MRX	AGX	SSGX	RDO
Koza-1	Mediana	212	433	113	358
	Valor-P	–	9.05e-10	3.19e-06	9.06e-10
			>	<	>
Koza-2	Mediana	189	331	108	357
	Valor-P	–	7.54e-10	7.02e-03	9.06e-10
			>	<	>
Koza-3	Mediana	192	551	108	339
	Valor-P	–	7.55e-10	3.77e-01	1.15e-09
			>	=	>

Tabela 39 – Quantidade de nós observados nas funções Nguyen.

Função	Estatística	MRX	AGX	SSGX	RDO
Nguyen-1	Mediana	175	389	106	314
	Valor-P	–	6.47e-07	3.96e-04	3.76e-06
			>	<	>
Nguyen-3	Mediana	214	436	106	388
	Valor-P	–	7.55e-10	2.19e-06	1.22e-09
			>	<	>
Nguyen-4	Mediana	270	437	106	422
	Valor-P	–	7.55e-10	7.55e-10	8.53e-10
			>	<	>
Nguyen-5	Mediana	199	381	104	354
	Valor-P	–	3.27e-09	7.54e-10	2.29e-09
			>	<	>
Nguyen-6	Mediana	202	480	103	387
	Valor-P	–	2.43e-09	2.19e-06	3.09e-08
			>	<	>
Nguyen-7	Mediana	213	431	104	373
	Valor-P	–	1.15e-09	4.01e-07	1.09e-09
			>	<	>
Nguyen-8	Mediana	236	410	107	363
	Valor-P	–	9.89e-08	1.86e-05	2.41e-06
			>	<	>
Nguyen-9	Mediana	185	31	105	45
	Valor-P	–	7.49e-07	2.05e-09	1.78e-02
			<	<	=
Nguyen-10	Mediana	151	118	103	40
	Valor-P	–	4.85e-06	4.56e-07	1.24e-04
			<	<	<

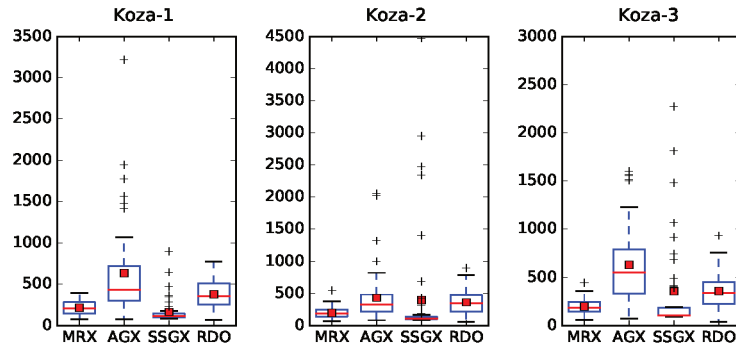


Figura 45 – Dispersão dos tamanhos (quantidade de nós) das respostas fornecidas no treinamento das funções do *benchmark* Koza.

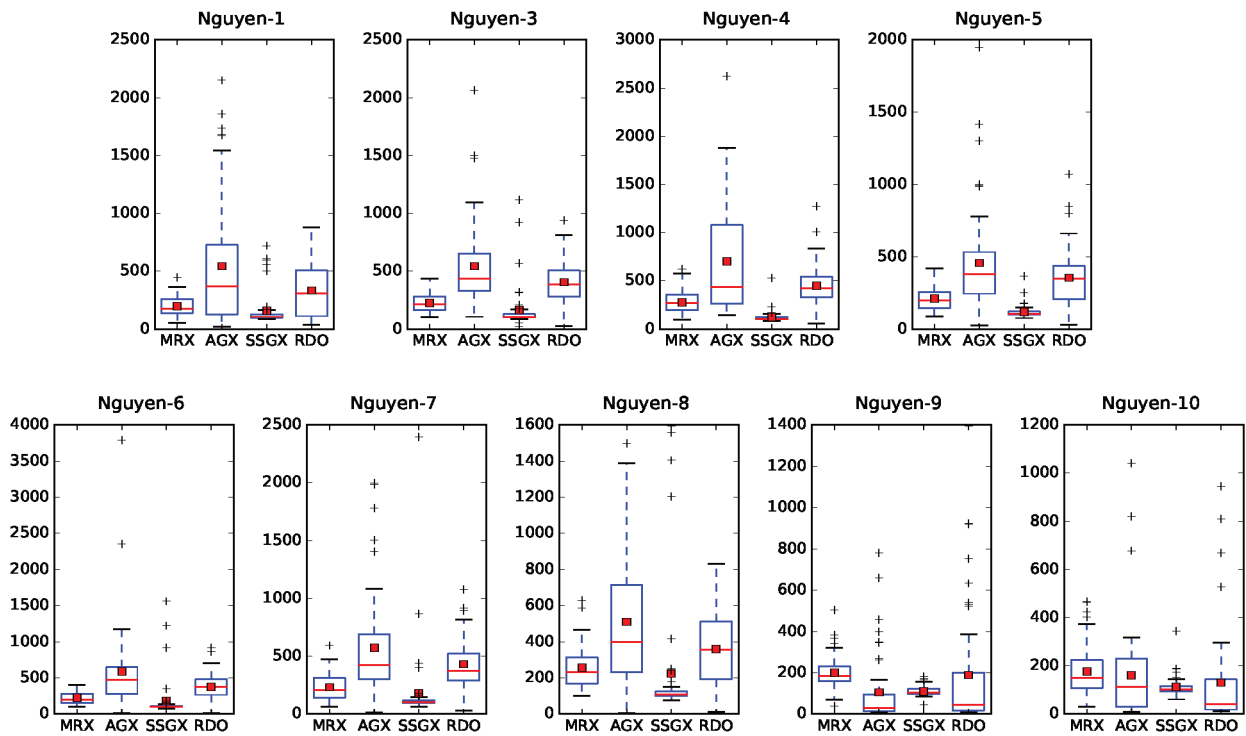
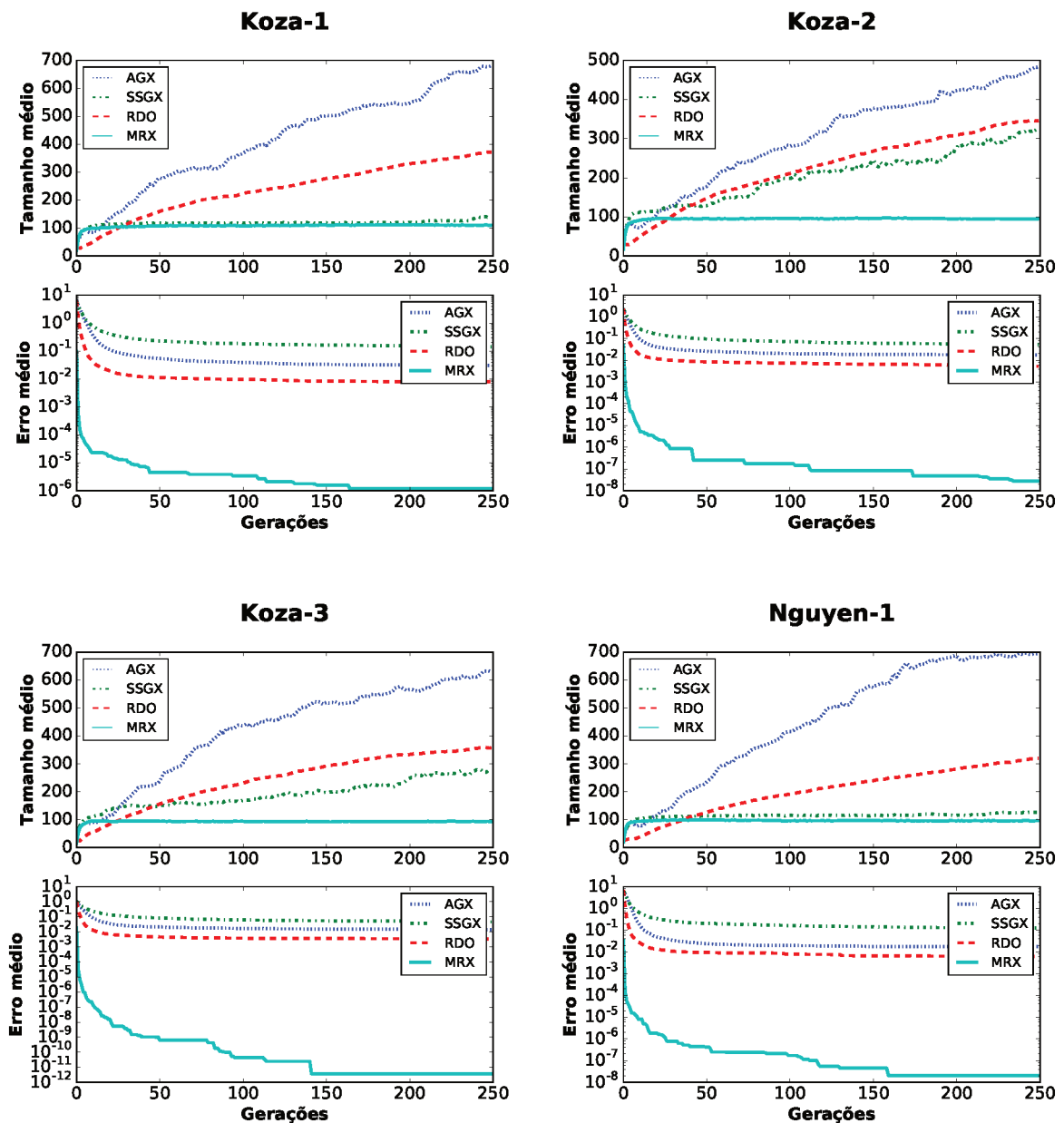


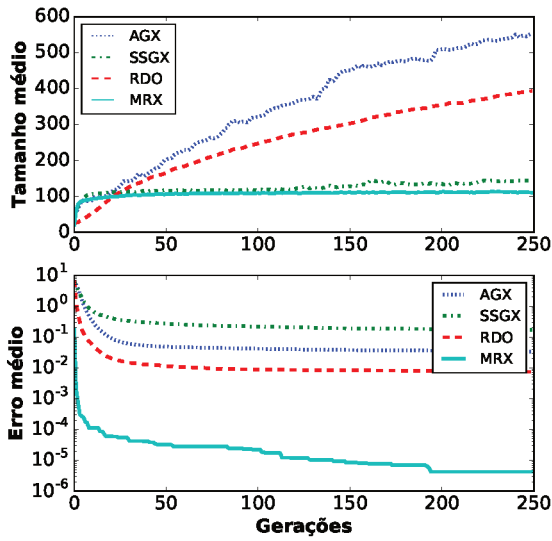
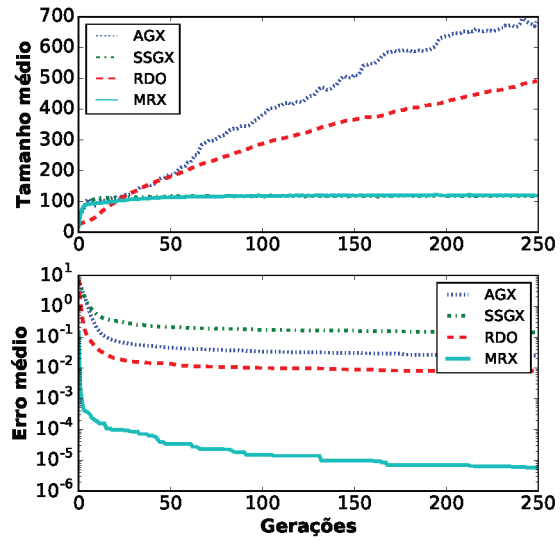
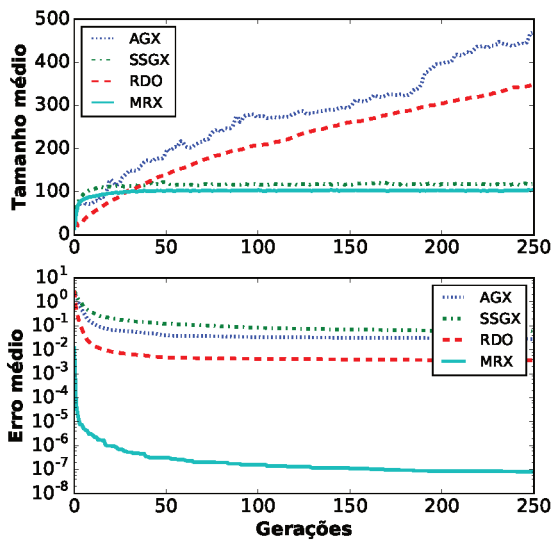
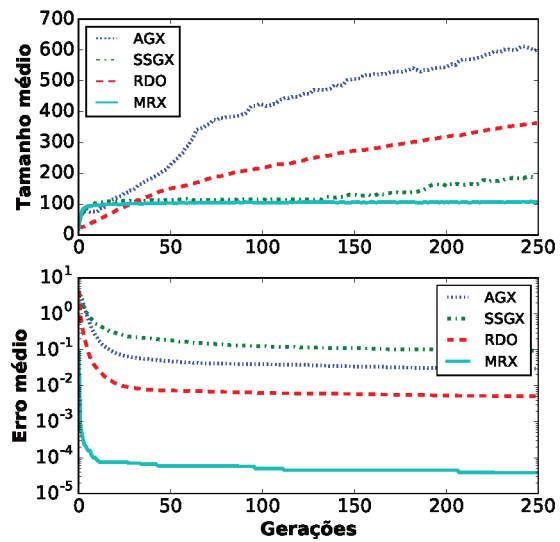
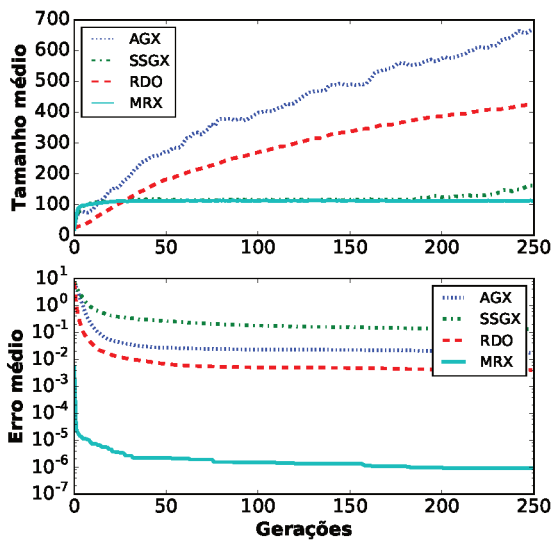
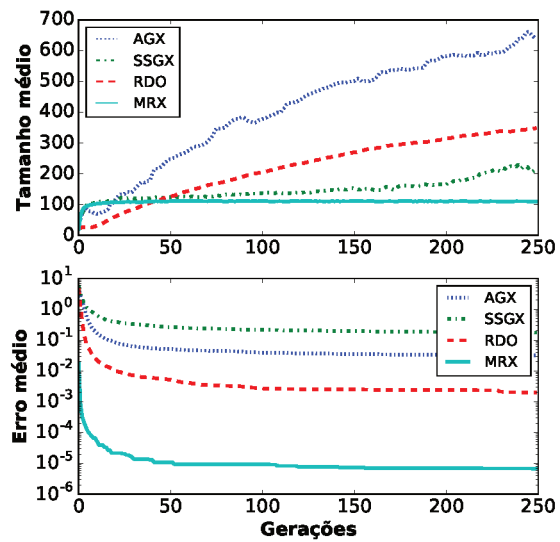
Figura 46 – Dispersão dos tamanhos (quantidade de nós) das respostas fornecidas no treinamento das funções do *benchmark* Nguyen.

4.4.5 O efeito *bloat*

A Figura 47 reúne uma série de gráficos que apresentam informações sobre o *bloat* nos operadores MRX, AGX, SSGX, e RDO. Comparamos os tamanhos médios das populações, geração a geração, e acrescentamos também a evolução do erro médio dos melhores indivíduos em todas as 50 execuções realizadas.

Podemos observar que único operador que, em média, se compara ao MRX em questão de *bloat* é o SSGX. Entretanto, a evolução do erro indica a superioridade do MRX, que sempre produz respostas melhores do que todos os demais operadores. Dentre as versões semânticas, o RDO é o operador que apresenta melhor desempenho no decréscimo do erro. De maneira geral, os três operadores semânticos seguem padrões bem semelhantes entre si, mas muito distantes (para pior) do MRX.



Nguyen-3**Nguyen-4****Nguyen-5****Nguyen-6****Nguyen-7****Nguyen-8**

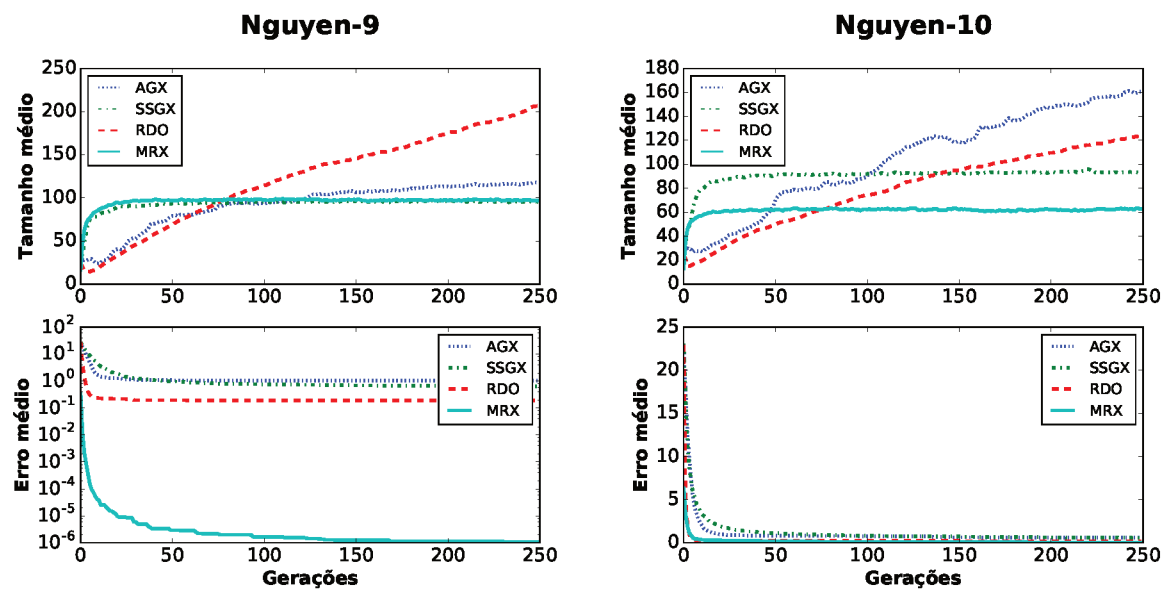


Figura 47 – Efeito *bloat* em funções dos *benchmarks* Koza e Nguyen, comparando o MRX e os operadores semânticos.

4.5 Comparação com a MRGP

Essa seção apresenta os resultados dos experimentos realizados para comparar o desempenho a GP-MRX com outra técnica de CE que evolui populações com base em conceitos de Regressão Linear Múltipla: a MRGP, discutida na Seção 2.4.

4.5.1 Configuração dos experimentos

Os dados utilizados para as análises apresentadas nesta seção são resultados de 50 execuções da GP-MRX e 50 execuções da MRGP, cuja implementação está disponível no site do Projeto FlexGP⁹. O experimento utilizou os *benchmark* Koza e Nguyen.

Ajustamos os parâmetros utilizados de forma que ambas as implementações executassem sob as mesmas condições. Isso significou não acrescentar constantes na GP-MRX, já que a MRGP, por padrão, não as utiliza. Como utilizamos apenas as funções dos *benchmarks* Koza e Nguyen, este fator não impactou os resultados, uma vez que, como discutido na Seção 4.1.2, o uso de constantes para estas duas funções é opcional. A não adoção de constantes foi também a razão de não realizar experimentos com os *benchmarks* Keijzer e Vladislavleva. Assim, utilizamos os seguintes parâmetros:

- Tamanho da população: 500;
- Taxa de recombinação: 90% para a MRGP e 100% para a GP-MRX;
- Taxa de mutação: 5% para a MRGP e 0% para a GP-MRX;
- Quantidade de gerações: 51 para a GP-MRX; até 51 gerações ou 15 minutos de execução para o MRGP;
- Torneio: 5;
- Profundidade inicial: 6;
- Funções: adaptadas a cada *dataset*, de acordo com os dados da Tabela 9;
- Constante: não foram utilizadas.

O número de gerações da MRGP pode ser inferior ao valor pré estabelecido, já que é especificada, no momento da execução, a quantidade máxima de minutos que o treinamento deve ocorrer.

A aplicação disponibilizada pelo autor não fornece informações diretas sobre o erro ao final do treinamento. Ao invés disso, é gravado em disco arquivos texto quem contém informações referentes ao erro quadrático médio apresentado no teste do melhor indivíduo.

⁹ <<http://flexgp.github.io/gp-learners/mrgp.html>>.

Por essa razão, não apresentamos aqui o valor obtido no treinamento. Ressalta-se, porém, que o padrão observado nos experimentos anteriores indicam que a atividade de teste é mais complexa do que a de treinamento, já requer do algoritmo a capacidade de generalização. Dessa forma, acreditamos não haver prejuízos na omissão dos dados de treinamento.

Colhemos o valor do erro quadrático médio MSE do melhor indivíduo de cada execução. As medianas desses conjuntos de valores são organizadas em tabelas, que reúnem também informações referentes testes de significância estatística. Aplicamos o teste não paramétrico de Wilcoxon, com significância de 1%, e reportamos o valor p obtido. Para explicitar a relação estatística do erro fornecido pela MRGP quando comparada com a GP-MRX, acrescentamos às tabelas informações no formato de sinais, onde “=” indica que os erros da GP-MRX e da MRGP são estatisticamente iguais, “<” indica que erro da MRGP é estatisticamente maior e “>” indica que o erro da MRGP é estatisticamente maior.

4.5.2 Teste das melhores respostas

As tabelas 40 e 41 exibem os resultados as comparações entre as duas técnicas. Os melhores indivíduos encontrados em cada execução foram submetidos aos conjuntos de teste. É possível constatar que a GP-MRX mostrou-se mais eficaz que a MRGP para todas as funções do *benchmark* Koza e Nguyen.

Tabela 40 – Comparação entre os testes do *benchmark* Koza na GP-MRX e na MRGP.

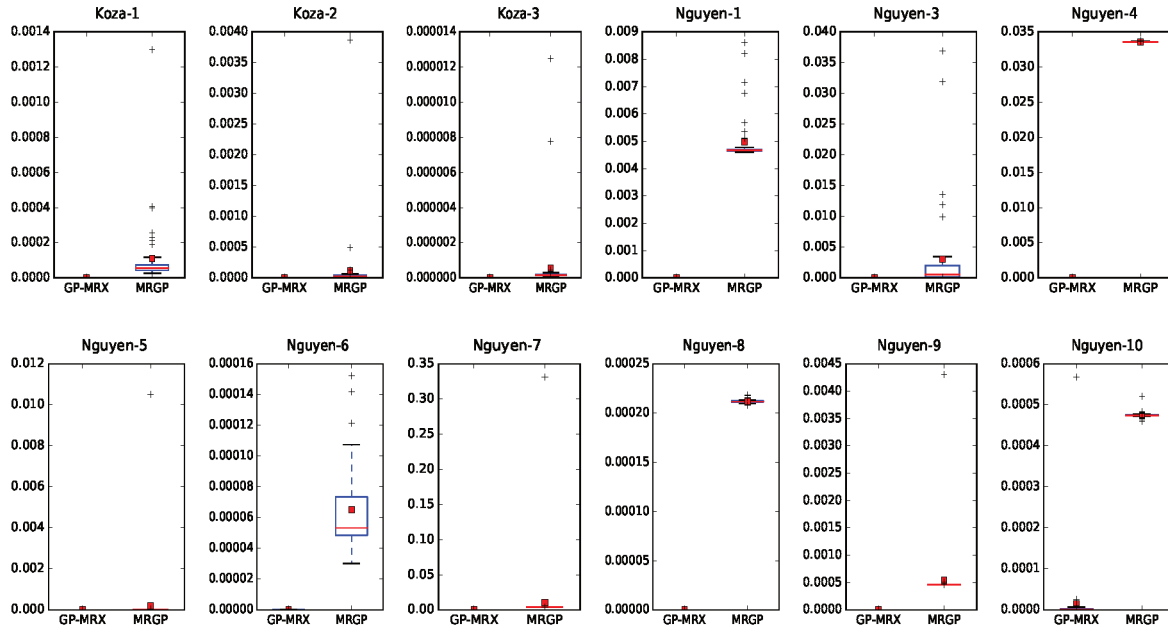
Função	Estatística	GP-MRX	MRGP
Koza-1	Mediana	2.692e-24	5.683e-05
	Valor-p	–	7.557e-10
			>
Koza-2	Mediana	1.708e-17	2.208e-05
	Valor-p	–	7.557e-10
			>
Koza-3	Mediana	9.544e-26	1.528e-07
	Valor-p	–	7.554e-10
			>

Já os *boxplots* exibidos na Figura 48 trazem informações referentes à dispersão dos valores dos testes. Notamos que a GP-MRX apresenta comportamento com melhor qualidade do que a MRGP, que além de apresentar maior amplitude, sofre também com a ocorrência de *outliers* em muitas de suas funções (com exceção apenas da Koza-2 e da Nguyen-4).

É interessante observar que mesmo tendo apresentado desempenho inferior à GP-MRX, a MRGP destaca-se por obter respostas de melhor qualidade às demais técnicas comparadas neste capítulo. Acreditamos que este comportamento seja decorrente do uso da técnica de regressão, que no caso da MRGP “calibra” cada sub-expressão presente no modelo.

Tabela 41 – Comparação entre os testes do *benchmark* Koza na GP-MRX e na MRGP.

Função	Estatística	GP-MRX	MRGP
Nguyen-1	Mediana	1.351e-24	4.668e-03
	Valor-p	–	1.110e-09 >
Nguyen-3	Mediana	6.893e-11	6.852e-04
	Valor-p	–	7.557e-10 >
Nguyen-4	Mediana	9.056e-12	3.358e-02
	Valor-p	–	7.557e-10 >
Nguyen-5	Mediana	5.128e-17	1.424e-06
	Valor-p	–	7.557e-10 >
Nguyen-6	Mediana	9.779e-09	5.349e-05
	Valor-p	–	7.557e-10 >
Nguyen-7	Mediana	1.433e-11	3.764e-03
	Valor-p	–	7.557e-10 >
Nguyen-8	Mediana	1.142e-10	2.115e-04
	Valor-p	–	7.557e-10 >
Nguyen-9	Mediana	7.959e-16	4.576e-04
	Valor-p	–	2.397e-09 >
Nguyen-10	Mediana	8.522e-07	4.742e-04
	Valor-p	–	1.110e-09 >

Figura 48 – Distribuição dos erros das respostas fornecidas pelo testes das funções dos *benchmarks* Koza e Nguyen.

4.5.3 Tamanho das melhores respostas

A MRGP realiza o controle de *bloat* por meio de técnicas multiobjetivo, conforme discutido na Seção 2.4.2.1. Por essa razão, ela tende a obter respostas mais compactas

do que a GP-MRX. Essa tendência é comprovada pelas informações que constam nas tabelas 42 e 43. A GP-MRX só obteve resultados competitivos nas funções Nguyen-9 e Nguyen-10, que não diferiram estatisticamente da MRGP.

Tabela 42 – Comparação entre os testes do *benchmark* Koza na GP-MRX e na MRGP.

Função	Estatística	GP-MRX	MRGP
Koza-1	Mediana	177	54
	Valor-P	–	7.552e-10 <
Koza-2	Mediana	210	54
	Valor-P	–	8.022e-10 <
Koza-3	Mediana	183	48
	Valor-P	–	7.555e-10 <

Tabela 43 – Comparação entre os testes do *benchmark* Koza na GP-MRX e na MRGP.

Função	Estatística	GP-MRX	MRGP
Nguyen-1	Mediana	142	51
	Valor-P	–	7.547e-10 <
Nguyen-3	Mediana	160	54
	Valor-P	–	7.550e-10 <
Nguyen-4	Mediana	194	51
	Valor-P	–	7.552e-10 <
Nguyen-5	Mediana	182	54
	Valor-P	–	7.552e-10 <
Nguyen-6	Mediana	203	54
	Valor-P	–	7.552e-10 <
Nguyen-7	Mediana	212	54
	Valor-P	–	7.554e-10 <
Nguyen-8	Mediana	210	48
	Valor-P	–	7.548e-10 <
Nguyen-9	Mediana	178	150
	Valor-P	–	1.383e-02 =
Nguyen-10	Mediana	150	171
	Valor-P	–	8.772e-01 =

A dispersão dos tamanhos apresentados pelos indivíduos, cuja informação está presentes nos gráficos da Figura 49, reforçam as evidências quanto à melhor adequação da MRGP em relação ao tamanho.

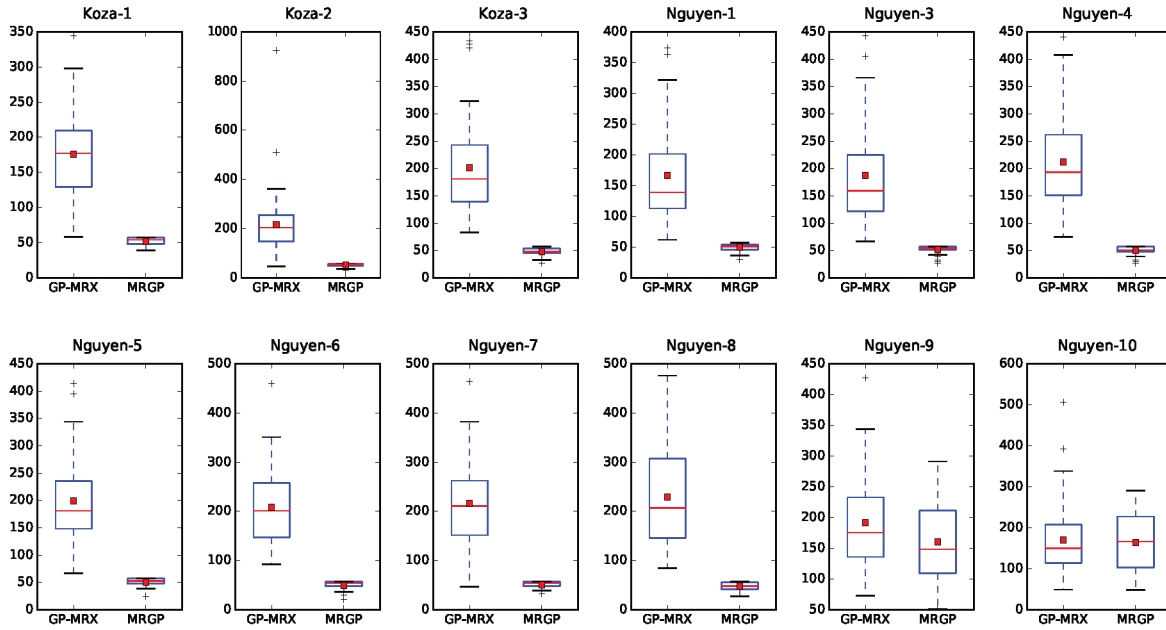


Figura 49 – Distribuição dos tamanhos (quantidade de nós) observados nos melhores indivíduos encontrados ao final treinamento nas funções dos *benchmarks* Koza e Nguyen.

4.6 Programação *Kaizen*

Analizamos os resultados de aprendizado e testes do *benchmark* Keijzer por parte da GP-MRX, comparando-os com os resultados fornecidos pela Programação *Kaizen*.

4.6.1 Configuração do experimento

O experimento realizado para comparar a PK com a GP-MRX baseou-se nos dados de treinamento e testes disponíveis em De Melo (2014), uma vez que não tivemos acesso ao código fonte original da Programação *Kaizen*. No trabalho utilizado com base na comparação, o *benchmark* Keijzer foi analisado em termos de treinamento e teste, com configurações de constantes e funções iguais ao adotado neste trabalho (Tabela 9).

Assim, realizamos 50 execuções da GP-MRX e, com base na descrição do experimentos realizados por De Melo (2014), adotamos os seguintes parâmetros para a GP-MRX:

- Tamanho da população: 500;
- Taxa de recombinação: 100%;
- Taxa de mutação: 0%;
- Quantidade de gerações: 51;

- Torneio: 5;
- Profundidade inicial: 6;
- Funções e constantes: adaptadas a cada *dataset*, de acordo com os dados da Tabela 9.

Da mesma forma que como foi feito no artigo original, organizamos os resultados obtidos em tabelas que contém as estatísticas de máximo, mediana e mínimo dos valores de R^2 , R^2 -ajustado e RMSE de cada função. Organizamos as informações em tabelas comparando resultados de treinamento e de teste. Porém, devido a ausência dos dados originais da PK, não foi possível realizar testes de significância estatística.

4.6.2 Resultados

Quando comparamos os resultados de treinamento fornecidos pela execução da GP-MRX, vemos que a Programação Kaizen obteve sucesso em onze das quinze funções do *benchmark* Keijzer, como é exibido na Tabela 44. Nota-se porém que na maioria das funções em que a GP-MRX apresentou desempenho inferior, mas sua diferença de RMSE para a PK foi pequena.

Embora a GP-MRX tenha obtido resultados medianos na fase de treinamento, a comparação com os resultados dos testes foram bem diferentes. Como pode ser observado na Tabela 45, a PK foi superada em onze das funções que compõem o *benchmark*. Esse resultado indica que GP-MRX apresenta maior capacidade de generalização quando comparada à PK.

Tabela 44 – Comparação entre os resultados obtidos pela GP-MRX no treinamento com as funções do *benchmark* Keijzer e comparação com os valores divulgados da PK em De Melo (2014).

Função	Estatística	GP-MRX		KP	
		R ²	RMSE	R ² -Ajustado	RMSE
Keijzer-1	Mínimo	9.949e-01	2.408e-04	9.996e-01	1.926e-06
	Mediana	9.989e-01	3.728e-03	1.000e+00	2.210e-04
	Máximo	1.000e+00	7.910e-03	1.000e+00	2.279e-02
Keijzer-2	Mínimo	4.524e-01	8.795e-02	1.252e-01	5.419e-04
	Mediana	6.874e-01	1.338e-01	8.994e-01	6.747e-02
	Máximo	8.650e-01	1.771e-01	1.000e+00	2.194e-01
Keijzer-3	Mínimo	7.613e-02	2.386e-01	1.259e-01	9.983e-02
	Mediana	3.693e-01	2.889e-01	3.830e-01	2.654e-01
	Máximo	5.695e-01	3.496e-01	9.132e-01	3.359e-01
Keijzer-4	Mínimo	6.131e-01	8.825e-02	5.120e-01	3.533e-02
	Mediana	8.714e-01	1.141e-01	7.408e-01	1.631e-01
	Máximo	9.231e-01	1.979e-01	9.887e-01	2.658e-01
Keijzer-5	Mínimo	8.568e-01	1.050e-02	1.000e+00	8.122e-04
	Mediana	9.894e-01	5.326e-02	1.000e+00	1.524e-03
	Máximo	9.996e-01	1.931e-01	1.000e+00	9.060e-03
Keijzer-6	Mínimo	0.000e+00	5.311e-09	1.000e+00	6.985e-16
	Mediana	1.000e+00	1.088e-07	1.000e+00	9.216e-16
	Máximo	1.000e+00	1.827e-06	1.000e+00	1.179e-14
Keijzer-7	Mínimo	0.000e+00	1.489e-08	1.000e+00	2.964e-06
	Mediana	1.000e+00	4.177e-07	1.000e+00	4.613e-04
	Máximo	1.000e+00	5.994e-06	1.000e+00	1.776e-02
Keijzer-8	Mínimo	0.000e+00	2.768e-13	1.000e+00	7.911e-16
	Mediana	1.000e+00	2.788e-13	1.000e+00	3.499e-15
	Máximo	1.000e+00	2.822e-13	1.000e+00	1.340e-02
Keijzer-9	Mínimo	0.000e+00	4.387e-08	1.000e+00	9.773e-06
	Mediana	1.000e+00	2.045e-06	1.000e+00	1.587e-03
	Máximo	1.000e+00	7.323e-06	1.000e+00	2.912e-02
Keijzer-10	Mínimo	9.935e-01	4.784e-03	1.000e+00	2.057e-03
	Mediana	9.973e-01	1.390e-02	1.000e+00	2.998e-03
	Máximo	9.997e-01	2.175e-02	1.000e+00	1.946e-02
Keijzer-11	Mínimo	9.704e-01	3.596e-01	9.709e-01	8.244e-02
	Mediana	9.791e-01	4.551e-01	9.936e-01	1.934e-01
	Máximo	9.869e-01	5.699e-01	9.985e-01	4.542e-01
Keijzer-12	Mínimo	9.592e-01	2.930e-12	9.999e-01	3.960e-15
	Mediana	9.988e-01	4.775e-01	1.000e+00	2.628e-02
	Máximo	1.000e+00	2.810e+00	1.000e+00	7.772e-01
Keijzer-13	Mínimo	6.399e-01	2.751e-01	8.947e-01	8.033e-02
	Mediana	7.920e-01	1.327e+00	9.765e-01	3.482e-01
	Máximo	9.911e-01	1.793e+00	9.990e-01	8.525e-01
Keijzer-14	Mínimo	9.254e-01	1.599e-02	9.890e-01	2.354e-03
	Mediana	9.966e-01	4.167e-02	9.984e-01	5.319e-02
	Máximo	9.995e-01	1.954e-01	1.000e+00	1.294e-01
Keijzer-15	Mínimo	9.670e-01	1.021e-12	1.000e+00	8.585e-16
	Mediana	9.991e-01	9.310e-02	1.000e+00	3.923e-15
	Máximo	1.000e+00	5.708e-01	1.000e+00	3.308e-02

Tabela 45 – Comparação entre os resultados obtidos pela GP-MRX no teste com as funções do *benchmarck* Keijzer e comparação com os valores divulgados da PK em De Melo (2014).

Função	Estatística	GP-MRX	KP
Keijzer-1	Mínimo	4.187e-03	4.528e-02
	Mediana	1.755e-01	8.260e-02
	Máximo	1.307e+04	2.423e+03
Keijzer-2	Mínimo	1.080e-01	1.729e-01
	Mediana	1.674e-01	2.391e-01
	Máximo	4.689e+01	2.345e+01
Keijzer-3	Mínimo	2.570e-01	2.950e-01
	Mediana	3.647e-01	3.889e-01
	Máximo	5.178e+02	9.999e+08
Keijzer-4	Mínimo	8.637e-02	2.516e-01
	Mediana	1.138e-01	3.165e-01
	Máximo	1.979e-01	9.736e+00
Keijzer-5	Mínimo	1.242e-02	1.946e-02
	Mediana	5.592e-02	1.928e-01
	Máximo	1.902e+00	2.236e+08
Keijzer-6	Mínimo	5.500e-06	9.958e-01
	Mediana	3.271e-04	9.958e-01
	Máximo	3.545e-03	9.958e-01
Keijzer-7	Mínimo	2.589e-08	7.352e-02
	Mediana	2.652e-06	2.040e-01
	Máximo	1.408e-04	9.995e+09
Keijzer-8	Mínimo	2.949e-13	0.000e+00
	Mediana	3.012e-13	0.000e+00
	Máximo	7.990e-13	0.000e+00
Keijzer-9	Mínimo	1.060e-06	1.452e-01
	Mediana	2.314e-05	3.650e-01
	Máximo	4.855e-04	9.995e+08
Keijzer-10	Mínimo	7.850e-03	3.179e-02
	Mediana	2.557e-02	3.492e-01
	Máximo	1.397e-01	3.570e-01
Keijzer-11	Mínimo	6.870e-01	6.091e-01
	Mediana	1.163e+00	6.425e-01
	Máximo	2.179e+05	4.624e+26
Keijzer-12	Mínimo	1.737e-11	1.024e+00
	Mediana	2.915e+00	1.342e+01
	Máximo	1.554e+03	7.184e+42
Keijzer-13	Mínimo	2.082e+00	1.985e+00
	Mediana	9.611e+00	8.184e+00
	Máximo	2.745e+03	5.658e+42
Keijzer-14	Mínimo	1.985e-01	9.317e-01
	Mediana	4.963e-01	2.094e+01
	Máximo	3.127e+01	1.803e+28
Keijzer-15	Mínimo	5.184e-13	8.889e-01
	Mediana	4.385e-01	3.559e+00
	Máximo	6.073e+01	5.868e+13

4.7 Considerações sobre o capítulo

Este capítulo apresentou experimentos com a finalidade de verificar a viabilidade do operador MRX, proposto neste trabalho. Com base em 35 funções de quatro *datasets* e com medidas de qualidade que envolvem valores de erro no treinamento, erro de teste e tamanho da resposta, foi possível avaliar as principais características do método desenvolvido. Os resultados mostram que o MRX é capaz de obter respostas de alta qualidade e com baixa dispersão. Além de ser superior à recombinação sintática, nosso operador foi superior também na comparação com as abordagens semânticas e estatísticas.

Os três operadores semânticos analisados na comparação (AGX, RDO e SSGX), mesmo tendo sido criados para inibir o crescimento exponencial inerente ao SGX, não conseguiram lidar com o *bloat* com a mesma desenvoltura do MRX. Entretanto, reparamos que o MRX possui comportamento característico no início das gerações, quando o tamanho médio dos indivíduos cresce muito rapidamente até atingir certo limite, a partir do qual segue sempre aproximadamente com o mesmo valor médio.

Uma das formas de avaliar a qualidade dos resultados obtidos é a análise dos gráficos gerados. Como utilizamos funções provindas de *benchmarks*, é possível comparar as curvas geradas pelas funções originais e pelas curvas geradas pelas funções evoluídas por meio do MRX. Por exemplo, os gráficos das figuras 50 e 51. Em ambos os casos, a curva original o mapeamento feito pela GP-MRX coincidem, ao contrário do que observamos nos resultados gerados pela GP.

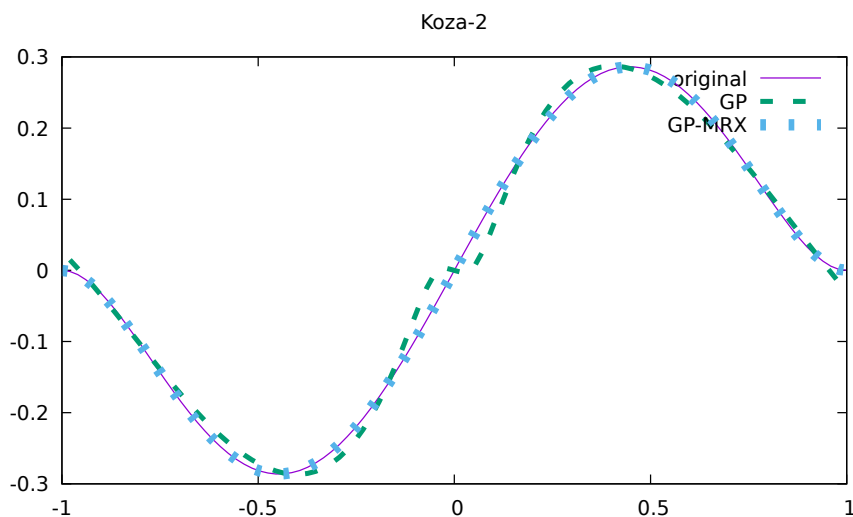


Figura 50 – Koza-2: comparação entre as curvas geradas pela recombinação de subárvore e pelo operador MRX.

Um interessante resultado obtido pela comparação da GP-MRX com a MRGP foi a qualidade dos resultados em ambas as abordagens. Mesmo com a MRGP tenho gerado resultados inferiores ao método proposto neste trabalho, sua qualidade (tanto

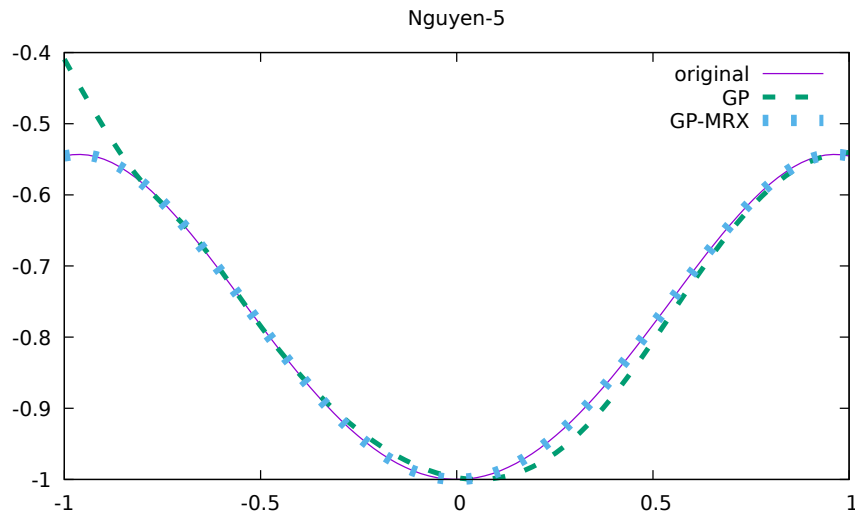


Figura 51 – Ngueyn-5: comparação entre as curvas geradas pela recombinação de subárvore e pelo operador MRX.

em valores de erro quanto em dispersão) foi muitas vezes superior às demais técnicas de CE analisadas neste trabalho. Acreditamos que este bom desempenho seja devido à combinação de regressão simbólica via PG com regressão linear múltipla.

O que mais de destacou na comparação com a Programação *Kaizen* foi a capacidade de generalização observada na GP-MRX, o que foi refletido nos resultados referentes à etapa de teste. Uma possível explicação para esse fenômeno é que as respostas obtidas pela PK tenham sofrido de “supertreinamento”, um efeito comum em técnicas de Aprendizado de Máquina. Foi relatado por De Melo (2014) que foram executados 2000 ciclos da PK. Talvez, os modelos gerados tenham “decorado” o padrão observado, tornando mais difícil a etapa de mapear padrões que não estavam presentes em seus casos de treinamento.

É importante ressaltar que nos experimentos apresentados neste capítulo foi adotada a estratégia de selecionar dos pais uma quantidade aleatória de subárvores para compor os descendentes. Essa mesma quantidade foi utilizada para determinar a quantidade de ramos aleatórios seriam inseridos nos modelos.

A quantidade de subárvores adotada como padrão ainda requer discussões mais detalhadas, uma vez que esse parâmetro pode influenciar diretamente na queda do erro e no *bloat*. A Figura 52, por exemplo, apresenta os resultados obtidos quando diferentes valores de subárvores (2, 4, 6 e 8 de cada pai) são adotados. É possível observar que sistemas com maior quantidade de nós podem fornecer erros mais baixos, mas com maior quantidade média de nós em suas populações. Em todos os casos, porém, o comportamento assintótico do *bloat* se mantém.

Nos experimentos realizados nesta tese, optamos por não fixar a quantidade de subárvores retiradas dos pais a fim de poder explorar diferentes locais do espaço de

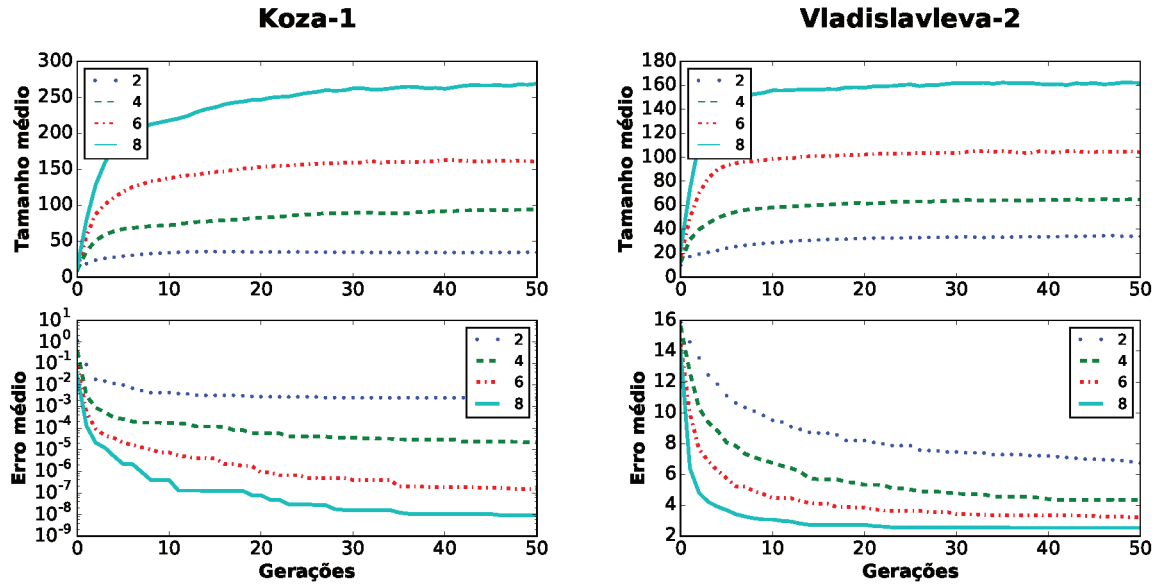


Figura 52 – Comportamento do MRX para diferentes quantidades de subárvores.

respostas. Assim, conseguimos proporcionar maior variedade genética, trabalhando com árvores de diferentes tamanhos, e ainda manter o *bloat* controlado.

Conclusão

Esta tese tem como principal contribuição a criação de um novo operador para Programação Genética capaz de gerar indivíduos de alta qualidade e evoluir populações com controle natural de *bloat*. Concluímos o trabalho discutindo alguns dos pontos mais relevantes apresentados ao longo do texto.

5.1 Considerações finais

O novo operador, projetado a princípio como um método de recombinação genética que proporciona o ajuste ótimo entre subárvores vindas dos pais por meio de constantes reais que aumentam a qualidade do indivíduo gerado, proporcionando também a remoção de subárvores irrelevantes, acabou por se tornar um operador híbrido e autoajustável.

Por acrescentar ao descendente certa quantidade de subárvores aleatórias, o MRX pode gerar indivíduos com características não mapeadas em qualquer ancestral e conduzir a evolução para regiões do espaço de busca até então inexploradas. Assim, o MRX se comporta como operador de recombinação, a produzir novos indivíduos por meio da herança variacional, e também como operador de mutação “consciente”, gerando perturbações na criação dos descendentes, mas com o cuidado de preservar apenas novas subárvores com contribuições benéficas.

O MRX foi criado como extensão de alguns conceitos explorados principalmente em duas técnicas de Computação Evolucionária que empregam Análise de Regressão: a MRGP e a Programação *Kaizen*. Questões específicas, porém significativas, podem ser apontadas como diferenciais entre essas abordagens e a técnica proposta. Um dos desafios enfrentados pela MRGP é a possibilidade de precisar trabalhar com modelos com grande número de sub-expressões, razão pela qual emprega uma implementação multiobjetivo, por meio do NSGA-II, para produzir (por todas as gerações) árvores pequenas e de baixo erro. Já o operador proposto neste trabalho limita a quantidade de sub-expressões aplicadas à análise de regressão e ainda verifica quais delas são realmente relevantes à solução buscada.

Diferente do MRX, a Programação *Kaizen* não é uma técnica baseada em população. Ela evolui um único modelo por vez, sendo que este é substituído quando um modelo melhor (que possua características do anterior e novas características, vindas de “ideias” criadas por especialistas) surgir. O fato é que em algum momento, os especialistas que propõem ideias podem não ser mais capazes de promover melhorias, causando estagnação, o que só é resolvido pela criação de novos especialistas que substituem a antiga equipe. Além disso, a criação de modelos via PK requer milhares de ciclos de processamento para encontrar a resposta com a qualidade desejada. Por trabalhar com populações de modelos e por promover a aleatoriedade por meio da inserção de componentes aleatórias ao modelo, algo que se assemelha à criação de “novas ideias” na PK, a utilização do MRX raramente apresenta indícios de estagnação, como pode ser visto nos resultados apresentados no Capítulo 4.

Outra questão relevante é a discussão acerca da possibilidade de se obter matrizes singulares durante o processamento da regressão. A inversão de matrizes é passo fundamental para a aplicação do método dos mínimos quadrados. Embora não haja menção a esta questão no que se refere à PK, o algoritmo MRX utiliza o conceito da matriz pseudo inversa neste ponto, a fim de gerar respostas aceitáveis independentemente dos características das matrizes processadas.

No que se refere à qualidade, nesta tese adotamos uma métrica centrada no erro obtido pelos melhores indivíduos encontrados ao final da etapa de treinamento, na capacidade de generalização desses indivíduos, pela utilização de *datasets* de teste, e na quantidade de nós que os compõem. Dessa maneira, foi possível comparar o operador desenvolvido com as recombinações sintáticas e semânticas. Nosso operador mostrou-se superior em quase todas as 35 funções analisadas nos *benchmarks*.

Por fim, embora não se caracterize exatamente como uma técnica de controle de *bloat*, o MRX parece conduzir a evolução para as regiões do espaço de busca onde se encontram as árvores mais compactas e de mais alta qualidade, promovendo assim o controle natural do tamanho dos indivíduos sem negligenciar a aptidão. De fato, vemos que as principais causas teóricas do *bloat*, que estão associadas aos operadores genéticos básicos, não parecem se aplicar ao algoritmo do MRX. Ou seja, o controle do tamanho médio da população ocorre como um agradável *efeito colateral*.

5.2 Trabalhos futuros

Os seguintes assuntos relacionados com a pesquisa apresentada neste trabalho ainda precisam ser explorados:

- a) Aplicar paradigmas de paralelismo e verificar se há ganhos reais de desempenho.

- b) Verificar a relação entre diversidade populacional no MRX promovida pela inserção de árvores aleatórias e a diversidade promovida pela adoção do paradigma de ilhas.
- c) Implementar o MRX com outros algoritmos de determinação dos coeficientes de regressão linear. Experimentos preliminares mostraram que o algoritmo Lasso, utilizado na MRGP, produz bons resultados para alguns grupos de funções, mas estudos mais aprofundados fazem-se necessários.
- d) Aplicar a GP-MRX para modelagem de problemas da Engenharia.
- e) Com o intuito de popularizar a utilização da Programação Genética em pesquisas de diferentes áreas, disponibilizar implementações da GP-MRX com interface amigável, para pesquisadores que tiveram pouco ou nenhum contato com técnicas de programação, e disponibilizar também uma API, destinada a programadores.

Referências

- ALTENBERG, L. The evolution of evolvability in genetic programming. In: Kinnear, Jr., K. E. (Ed.). **Advances in Genetic Programming**. [S.l.]: MIT Press, 1994. cap. 3, p. 47–74.
- ANGELINE, P. J. Genetic programming and emergent intelligence. In: Kinnear, Jr., K. E. (Ed.). **Advances in Genetic Programming**. [S.l.]: MIT Press, 1994. cap. 4, p. 75–98.
- ARNALDO, I.; KRAWIEC, K.; O'REILLY, U.-M. Multiple regression genetic programming. In: **Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation**. New York, NY, USA: ACM, 2014. (GECCO '14), p. 879–886. ISBN 978-1-4503-2662-9. Disponível em: <<http://doi.acm.org/10.1145/2576768.2598291>>.
- BANZHAF, W. Genetic programming for pedestrians. In: FORREST, S. (Ed.). **Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93**. University of Illinois at Urbana-Champaign: Morgan Kaufmann, 1993. p. 628. Disponível em: <<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.38.1144>>.
- BEADLE, L.; JOHNSON, C. Semantically driven crossover in genetic programming. In: WANG, J. (Ed.). **Proceedings of the IEEE World Congress on Computational Intelligence**. Hong Kong: IEEE Press, 2008. p. 111–116. Disponível em: <<https://doi.org/10.1109/CEC.2008.4630784>>.
- BLEULER, S. et al. Multiobjective genetic programming: Reducing bloat using SPEA2. In: **Proceedings of the 2001 Congress on Evolutionary Computation CEC2001**. COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea: IEEE Press, 2001. p. 536–543. ISBN 0-7803-6658-1. Disponível em: <<https://doi.org/10.1109/CEC.2001.934438>>.
- BLICKLE, T. Evolving compact solutions in genetic programming: A case study. In: VOIGT, H.-M. et al. (Ed.). **Parallel Problem Solving From Nature IV. Proceedings of the International Conference on Evolutionary Computation**. Berlin, Germany: Springer-Verlag, 1996. (LNCS, v. 1141), p. 564–573. ISBN 3-540-61723-X. Disponível em: <https://doi.org/10.1007/3-540-61723-X_1020>.
- BLICKLE, T.; THIELE, L. Genetic programming and redundancy. In: HOPF, J. (Ed.). **Genetic Algorithms within the Framework of Evolutionary Computation (Workshop at KI-94, Saarbrücken)**. Im Stadtwald, Building 44, D-66123 Saarbrücken, Germany: Max-Planck-Institut für Informatik (MPI-I-94-241), 1994. p. 33–38.

_____. **A Comparison of Selection Schemes Used in Genetic Algorithms.**

Gloriastrasse 35, 8092 Zurich, Switzerland, 1995. Disponível em: <<https://doi.org/10.1162/evco.1996.4.4.361>>.

BRAMEIER, M.; BANZHAF, W. Neutral variations cause bloat in linear gp. **Applied Statistics**, 2003. Disponível em: <https://doi.org/10.1007/3-540-36599-0_26>.

BURKE, E. K.; GUSTAFSON, S.; KENDALL, G. Diversity in genetic programming: An analysis of measures and correlation with fitness. **IEEE Transactions on Evolutionary Computation**, IEEE Press, v. 8, n. 1, p. 47–62, fev. 2004. ISSN 1089-778X. Disponível em: <<https://doi.org/10.1109/TEVC.2003.819263>>.

CASTELLI, M. et al. The influence of population size in geometric semantic GP. **Swarm and Evolutionary Computation**, v. 32, p. 110–120, 2017. ISSN 2210-6502. Disponível em: <<https://doi.org/10.1016/j.swevo.2016.05.004>>.

CHELLAPILLA, K. Evolutionary programming with tree mutations: Evolving computer programs without crossover. In: KOZA, J. R. et al. (Ed.). **Genetic Programming 1997: Proceedings of the Second Annual Conference**. Stanford University, CA, USA: Morgan Kaufmann, 1997. p. 431–438.

CHEN, Q.; ZHANG, M.; XUE, B. Geometric semantic genetic programming with perpendicular crossover and random segment mutation for symbolic regression. In: SHI, Y. et al. (Ed.). **Simulated Evolution and Learning**. Cham: Springer International Publishing, 2017. p. 422–434. ISBN 978-3-319-68759-9. Disponível em: <https://doi.org/10.1007/978-3-319-68759-9_35>.

CORMEN, T. H. et al. **Introduction to Algorithms**. [S.l.]: McGraw-Hill Higher Education, 2001. ISBN 0070131511.

De Jong, K. A. **Evolutionary computation - a unified approach**. MIT Press, 2006. I-IX, 1-256 p. ISBN 978-0-262-04194-2. Disponível em: <<https://doi.org/10.1007/s10710-007-9035-9>>.

De Melo, V. V. Kaizen programming. In: IGEL, C. et al. (Ed.). **GECCO '14: Proceedings of the 2014 conference on Genetic and evolutionary computation**. Vancouver, BC, Canada: ACM, 2014. p. 895–902. Disponível em: <<http://doi.acm.org/10.1145/2576768.2598264>>.

de Melo, V. V.; BANZHAF, W. Kaizen programming for feature generation. In: RIOLO, R. et al. (Ed.). **Genetic Programming Theory and Practice XIII**. Ann Arbor, USA: Springer, 2015. (Genetic and Evolutionary Computation). Disponível em: <<https://doi.org/10.1007/978-3-319-34223-8>>.

_____. Improving logistic regression classification of credit approval with features constructed by kaizen programming. In: FRIEDRICH, T. et al. (Ed.). **GECCO '16 Companion: Proceedings of the Companion Publication of the 2016 Annual Conference on Genetic and Evolutionary Computation**. Denver, USA: ACM, 2016. p. 61–62. Disponível em: <<https://doi.org/10.1145/2908961.2908963>>.

DEB, K. et al. A fast and elitist multiobjective genetic algorithm: Nsga-ii. **Evolutionary Computation**, **IEEE Transactions on**, v. 6, n. 2, p. 182–197, 2002. ISSN 1089-778X. Disponível em: <<https://doi.org/10.1109/4235.996017>>.

DERRAC, J. et al. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. **Swarm and Evolutionary Computation**, v. 1, n. 1, p. 3 – 18, 2011. ISSN 2210-6502. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S2210650211000034>>.

DEUFLHARD, P.; SAUTTER, W. On rank-deficient pseudoinverses. **Linear Algebra and its Applications**, v. 29, p. 91 – 111, 1980. ISSN 0024-3795. Special Volume Dedicated to Alson S. Householder. Disponível em: <[https://doi.org/10.1016/0024-3795\(80\)90232-3](https://doi.org/10.1016/0024-3795(80)90232-3)>.

D'HAESELEER, P. Context preserving crossover in genetic programming. In: **Proceedings of the 1994 IEEE World Congress on Computational Intelligence**. Orlando, Florida, USA: IEEE Press, 1994. v. 1, p. 256–261. Disponível em: <<https://doi.org/10.1109/ICEC.1994.350006>>.

DOU, T.; ROCKETT, P. Comparison of semantic-based local search methods for multiobjective genetic programming. **Genetic Programming and Evolvable Machines**, v. 19, n. 4, p. 535–563, Dec 2018. ISSN 1573-7632. Disponível em: <<https://doi.org/10.1007/s10710-018-9325-4>>.

EFRON, B. et al. Least angle regression. **The Annals of Statistics**, Institute of Mathematical Statistics, v. 32, n. 2, p. 407–451, 2004. ISSN 00905364. Disponível em: <<https://doi.org/10.1214/009053604000000067>>.

FOGEL, L. J. Autonomous automata. **Industrial Research**, v. 4, p. 14–19, 1962.

GELLY, S. et al. A statistical learning theory approach of bloat. In: BEYER, H.-G. et al. (Ed.). **GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation**. Washington DC, USA: ACM Press, 2005. v. 2, p. 1783–1784. ISBN 1-59593-010-8. Disponível em: <<https://doi.org/10.1145/1068009.1068309>>.

_____. Universal consistency and bloat in GP. **Revue d'Intelligence Artificielle, HAL - CCSD - CNRS**, v. 20, n. 6, p. 805–827, 2006. ISSN 0992-499X. Issue on New Methods in Machine Learning. Theory and Applications. Disponível em: <<https://doi.org/10.3166/ria.20.805-827>>.

GITLOW, H.; HOWARD, S. **Tools and Methods for the Improvement of Quality**. [S.l.]: Taylor & Francis, 1989. (Irwin series in quantitative analysis for business). ISBN 9780256056808.

GOODRICH, M. T.; TAMASSIA, R. **Data structures and algorithms in Java (3. ed.)**. [S.l.]: Wiley, 2003. I-XVII, 1-681 p. ISBN 978-0-471-64452-1.

GRAHAM, P. **ANSI Common Lisp**. Upper Saddle River, NJ, USA: Prentice Hall Press, 1996. ISBN 0-13-370875-6.

GRIFFITHS, T. D.; EKÁRT, A. Improving the tartarus problem as a benchmark in genetic programming. In: MCDERMOTT, J. et al. (Ed.). **Genetic Programming**. Cham: Springer International Publishing, 2017. p. 278–293. ISBN 978-3-319-55696-3.

HARRIES, K.; SMITH, P. Exploring alternative operators and search strategies in genetic programming. In: KOZA, J. R. et al. (Ed.). **Genetic Programming 1997: Proceedings of the Second Annual Conference**. Stanford University, CA, USA: Morgan Kaufmann, 1997. p. 147–155.

HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. **The elements of statistical learning: data mining, inference and prediction**. 2. ed. Springer, 2009. Disponível em: <<https://doi.org/10.1007/978-0-387-84858-7>>.

HINES, W.; MONTGOMERY, D.; BORROR, D. **PROBABILITY AND STATISTICS IN ENGINEERING, 4TH ED.** [S.l.]: Wiley India Pvt. Limited, 2008. ISBN 9788126516469.

HOLLAND, J. H. **Adaptation in Natural and Artificial Systems**. Ann Arbor, MI, USA: University of Michigan Press, 1975.

IBA, H.; de Garis, H.; SATO, T. Genetic programming using a minimum description length principle. In: Kinnear, Jr., K. E. (Ed.). **Advances in Genetic Programming**. [S.l.]: MIT Press, 1994. cap. 12, p. 265–284.

IBA, H.; DEGARIS, H.; SATO, T. A numerical approach to genetic programming for system identification. **Evolutionary Computation**, v. 3, n. 4, p. 417–452, 1995. Disponível em: <<https://doi.org/10.1162/evco.1995.3.4.417>>.

IGEL, C.; CHELLAPILLA, K. Investigating the influence of depth and degree of genotypic change on fitness in genetic programming. In: BANZHAF, W. et al. (Ed.). **Proceedings of the Genetic and Evolutionary Computation Conference**. Orlando, Florida, USA: Morgan Kaufmann, 1999. v. 2, p. 1061–1068. ISBN 1-55860-611-4. Disponível em: <<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.15.6512>>.

KEIJZER, M. Improving symbolic regression with interval arithmetic and linear scaling. In: **Proceedings of the 6th European Conference on Genetic Programming**. Berlin, Heidelberg: Springer-Verlag, 2003. (EuroGP'03), p. 70–82. ISBN 3-540-00971-X. Disponível em: <https://doi.org/10.1007/3-540-36599-0_7>.

Kinnear, Jr., K. E. Generality and difficulty in genetic programming: Evolving a sort. In: FORREST, S. (Ed.). **Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93**. University of Illinois at Urbana-Champaign: Morgan Kaufmann, 1993. p. 287–294.

_____. Fitness landscapes and difficulty in genetic programming. In: **Proceedings of the 1994 IEEE World Conference on Computational Intelligence**. Orlando, Florida, USA: IEEE Press, 1994. v. 1, p. 142–147. ISBN 0-7803-1899-4. Disponível em: <<https://doi.org/10.1109/ICEC.1994.350026>>.

KOZA, J. R. Genetic evolution and co-evolution of computer programs. In: LANGTON, C. G. et al. (Ed.). **Artificial Life II**. Addison Wesley Publishing Company, 1992. p. 313–324. Disponível em: <<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.49.648>>.

_____. **Genetic Programming: On the Programming of Computers by Means of Natural Selection**. Cambridge, MA, USA: MIT Press, 1992. ISBN 0-262-11170-5.

KRAWIEC, K.; PAWLAK, T. Approximating geometric crossover by semantic backpropagation. In: BLUM, C. et al. (Ed.). **GECCO '13: Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference**. Amsterdam, The Netherlands: ACM, 2013. p. 941–948. Disponível em: <<https://doi.org/10.1145/2463372.2463483>>.

- LANGDON, W. B. The evolution of size in variable length representations. In: **1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)**. [s.n.], 1998. p. 633–638. Disponível em: <<https://doi.org/10.1109/ICEC.1998.700102>>.
- _____. Size fair and homologous tree genetic programming crossovers. **Genetic Programming and Evolvable Machines**, v. 1, n. 1/2, p. 95–119, abr. 2000. ISSN 1389-2576. Disponível em: <<https://doi.org/10.1023/A:1010024515191>>.
- LANGDON, W. B.; BANZHAF, W. Genetic programming bloat without semantics. In: SCHOENAUER, M. et al. (Ed.). **Parallel Problem Solving from Nature PPSN VI**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000. p. 201–210. ISBN 978-3-540-45356-7. Disponível em: <https://doi.org/10.1007/3-540-45356-3_20>.
- LANGDON, W. B.; POLI, R. Fitness causes bloat. In: CHAWDHRY, P. K.; ROY, R.; PANT, R. K. (Ed.). **Soft Computing in Engineering Design and Manufacturing**. London: Springer London, 1998. p. 13–22. ISBN 978-1-4471-0427-8. Disponível em: <https://doi.org/10.1007/978-1-4471-0427-8_2>.
- _____. Fitness causes bloat: Mutation. In: BANZHAF, W. et al. (Ed.). **Proceedings of the First European Workshop on Genetic Programming**. Paris: Springer-Verlag, 1998. (LNCS, v. 1391), p. 37–48. ISBN 3-540-64360-5. Disponível em: <<https://doi.org/10.1007/BFb0055926>>.
- _____. Genetic programming bloat with dynamic fitness. In: BANZHAF, W. et al. (Ed.). **Proceedings of the First European Workshop on Genetic Programming**. Paris: Springer-Verlag, 1998. (LNCS, v. 1391), p. 97–112. ISBN 3-540-64360-5. Disponível em: <<https://doi.org/10.1007/BFb0055931>>.
- LINDEN, R. (Ed.). **Algoritmos Genéticos: uma importante ferramenta da Inteligência Computacional**. Rio de Janeiro, RJ, Brasil: Brasport, 2008. ISBN 978-85-7452-373-6.
- LISKOWSKI, P.; BLADEK, I.; KRAWIEC, K. Neuro-guided genetic programming: Prioritizing evolutionary search with neural networks. In: **Proceedings of the Genetic and Evolutionary Computation Conference**. New York, NY, USA: ACM, 2018. (GECCO '18), p. 1143–1150. ISBN 978-1-4503-5618-3. Disponível em: <<http://doi.acm.org/10.1145/3205455.3205629>>.
- LUKE, S. Code growth is not caused by introns. In: WHITLEY, D. (Ed.). **Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference**. Las Vegas, Nevada, USA: [s.n.], 2000. p. 228–235.
- LUKE, S. **Issues in Scaling Genetic Programming: Breeding Strategies, Tree Generation, and Code Bloat**. Tese (Doutorado) — Department of Computer Science, University of Maryland, A. V. Williams Building, University of Maryland, College Park, MD 20742 USA, 2000.
- _____. Modification point depth and genome growth in genetic programming. **Evol. Comput.**, MIT Press, Cambridge, MA, USA, v. 11, n. 1, p. 67–106, mar. 2003. ISSN 1063-6560. Disponível em: <<https://doi.org/10.1162/106365603321829014>>.

LUKE, S.; PANAIT, L. Lexicographic parsimony pressure. In: LANGDON, W. B. et al. (Ed.). **GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference**. New York: Morgan Kaufmann Publishers, 2002. p. 829–836. ISBN 1-55860-878-8. Disponível em: <<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.19.2371>>.

_____. A comparison of bloat control methods for genetic programming. **Evolutionary Computation**, v. 14, n. 3, p. 309–344, Fall 2006. ISSN 1063-6560. Disponível em: <<https://doi.org/doi:10.1162/evco.2006.14.3.309>>.

LUKE, S.; SPECTOR, L. A comparison of crossover and mutation in genetic programming. In: KOZA, J. R. et al. (Ed.). **Genetic Programming 1997: Proceedings of the Second Annual Conference**. Stanford University, CA, USA: Morgan Kaufmann, 1997. p. 240–248.

MARQUES, L. G. **Programação genética paralela com Pareto: uma ferramenta para modelagem via regressão simbólica**. Dissertação (Mestrado) — Faculdade de Engenharia Elétrica da Universidade Federal de Uberlândia, 2013.

MCDERMOTT, J. et al. Genetic programming needs better benchmarks. In: SOULE, T. et al. (Ed.). **GECCO '12: Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference**. Philadelphia, Pennsylvania, USA: ACM, 2012. p. 791–798. Disponível em: <<https://doi.org/10.1145/2330163.2330273>>.

MCKAY, B. et al. Non-linear continuum regression using genetic programming. In: **Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 2**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999. (GECCO'99), p. 1106–1111. ISBN 1-55860-611-4. Disponível em: <<http://citeseerx.ist.psu.edu/viewdoc/summary;jsessionid=AB3DCDE422964E80672EBAE831F5836F?doi=10.1.1.153.732>>.

MCPHEE, N. F.; MILLER, J. D. Accurate replication in genetic programming. In: ESHELMAN, L. J. (Ed.). **Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)**. Pittsburgh, PA, USA: Morgan Kaufmann, 1995. p. 303–309. ISBN 1-55860-370-0. Disponível em: <<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.51.6390>>.

MCPHEE, N. F.; OHS, B.; HUTCHISON, T. **Semantic Building Blocks in Genetic Programming**. 600 East 4th Street, Morris, MN 56267, USA, 2007. Disponível em: <https://doi.org/10.1007/978-3-540-78671-9_12>.

MEI, Y.; ZHANG, M. Genetic programming hyper-heuristic for multi-vehicle uncertain capacitated arc routing problem. In: **Proceedings of the Genetic and Evolutionary Computation Conference Companion**. New York, NY, USA: ACM, 2018. (GECCO '18), p. 141–142. ISBN 978-1-4503-5764-7. Disponível em: <<http://doi.acm.org/10.1145/3205651.3205661>>.

MELO, V. V. de; BANZHAF, W. Improving the prediction of material properties of concrete using kaizen programming with simulated annealing. **Neurocomputing**, v. 246, p. 25 – 44, 2017. ISSN 0925-2312. Brazilian Conference on Intelligent Systems 2015. Disponível em: <<https://doi.org/10.1016/j.neucom.2016.12.077>>.

MONTGOMERY, D.; RUNGER, G. **Applied Statistics and Probability for Engineers**. [S.l.]: John Wiley & Sons, 2010. ISBN 9780470053041.

MOORE, E. H. On the reciprocal of the general algebraic matrix. **Bulletin of the American Mathematical Society**, v. 26, p. 394–395, 1920.

MORAGLIO, A.; KRAWIEC, K.; JOHNSON, C. Geometric semantic genetic programming. In: IGEL, C.; LEHRE, P. K.; WITT, C. (Ed.). **The 5th workshop on Theory of Randomized Search Heuristics, ThRaSH'2011**. Copenhagen, Denmark: [s.n.], 2011. Disponível em: <https://doi.org/10.1007/978-3-642-32937-1_3>.

NGUYEN, Q. U. et al. Subtree semantic geometric crossover for genetic programming. **Genetic Programming and Evolvable Machines**, v. 17, n. 1, p. 25–53, mar. 2016. ISSN 1389-2576. Disponível em: <<https://doi.org/10.1007/s10710-015-9253-5>>.

NGUYEN, T. H.; NGUYEN, X. H. A brief overview of population diversity measures in genetic programming. In: PHAM, T. L.; LE, H. K.; NGUYEN, X. H. (Ed.). **Proceedings of the Third Asian-Pacific workshop on Genetic Programming**. Military Technical Academy, Hanoi, VietNam: [s.n.], 2006. p. 128–139. ISSN 18590209. Disponível em: <<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.104.912>>.

NORDIN, P.; BANZHAF, W.; FRANCONI, F. D. Efficient evolution of machine code for cisc architectures using instruction blocks and homologous crossover. In: SPECTOR, L. et al. (Ed.). **Advances in Genetic Programming 3**. Cambridge, MA, USA: MIT Press, 1999. cap. 12, p. 275–299. ISBN 0-262-19423-6. Disponível em: <<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.140.9365>>.

NORDIN, P.; FRANCONI, F.; BANZHAF, W. Explicitly defined introns and destructive crossover in genetic programming. In: ROSCA, J. P. (Ed.). **Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications**. Tahoe City, California, USA: [s.n.], 1995. p. 6–22.

O'NEILL, M. et al. Open issues in genetic programming. **Genetic Programming and Evolvable Machines**, v. 11, n. 3/4, p. 339–363, set. 2010. ISSN 1389-2576. Tenth Anniversary Issue: Progress in Genetic Programming and Evolvable Machines. Disponível em: <<https://doi.org/10.1007/s10710-010-9113-2>>.

O'REILLY, U.-M. **An Analysis of Genetic Programming**. Tese (Doutorado) — Carleton University, Ottawa-Carleton Institute for Computer Science, Ottawa, Ontario, Canada, 22 set. 1995.

O'REILLY, U.-M.; OPPACHER, F. The troubling aspects of a building block hypothesis for genetic programming. In: WHITLEY, L. D.; VOSE, M. D. (Ed.). **Foundations of Genetic Algorithms 3**. Estes Park, Colorado, USA: Morgan Kaufmann, 1994. p. 73–88. ISBN 1-55860-356-5. Published 1995. Disponível em: <<https://doi.org/10.1016/B978-1-55860-356-1.50008-X>>.

_____. **Hybridized Crossover-Based Search Techniques for Program Discovery**. 1399 Hyde Park Road Santa Fe, New Mexico 87501-8943 USA, 1995. Disponível em: <<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.21.1499>>.

PAWLAK, T. P.; WIELOCH, B.; KRAWIEC, K. Semantic backpropagation for designing search operators in genetic programming. **IEEE Transactions on Evolutionary Computation**, v. 19, n. 3, p. 326–340, jun. 2015. ISSN 1089-778X. Disponível em: <<https://doi.org/10.1109/TEVC.2014.2321259>>.

PENROSE, R. A generalized inverse for matrices. In: **Proc. Cambridge Philos. Soc.** [S.l.: s.n.], 1955. v. 51, p. 406–413.

PERETTA, I. S. **Evolution of differential models for concrete complex systems through genetic programming**. Tese (info:eu-repo/semantics/doctoralThesis; Theses) — Laboratoire des sciences de l'ingenieur, de l'informatique et de l'imagerie (ICube), Universite de Strasbourg (UNISTRA), France, set. 21 2015. Disponível em: <<https://tel.archives-ouvertes.fr/tel-01332289>>.

POLI, R.; LANGDON, W. B. Genetic programming with one-point crossover. In: CHAUDHRY, P. K.; ROY, R.; PANT, R. K. (Ed.). **Soft Computing in Engineering Design and Manufacturing**. Springer-Verlag London, 1997. p. 180–189. ISBN 3-540-76214-0. Disponível em: <https://doi.org/10.1007/978-1-4471-0427-8_20>.

_____. On the search properties of different crossover operators in genetic programming. In: KOZA, J. R. et al. (Ed.). **Genetic Programming 1998: Proceedings of the Third Annual Conference**. University of Wisconsin, Madison, Wisconsin, USA: Morgan Kaufmann, 1998. p. 293–301. ISBN 1-55860-548-7.

POLI, R.; LANGDON, W. B.; MCPHEE, N. F. **A field guide to genetic programming**. Published via <<http://lulu.com>> and freely available at <<http://www.gp-field-guide.org.uk>>, 2008. (With contributions by J. R. Koza). Disponível em: <<https://doi.org/10.1007/s10710-008-9073-y>>.

POLI, R.; MCPHEE, N. Parsimony pressure made easy. In: KEIJZER, M. et al. (Ed.). **GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation**. Atlanta, GA, USA: ACM, 2008. p. 1267–1274. Disponível em: <<https://doi.org/10.1145/1389095.1389340>>.

SILVA, S. **Controlling Bloat: Individual and Population Based Approaches in Genetic Programming**. Tese (Doutorado) — Coimbra University, Portugal, abr. 2008. Full author name is Sara Guilherme Oliveira da Silva. Disponível em: <http://cisuc.dei.uc.pt/ecos/dlfile.php?fn=1749_pub_phdsara.pdf>.

SILVA, S.; ALMEIDA, J. Dynamic maximum tree depth. In: CANTÚ-PAZ, E. et al. (Ed.). **Genetic and Evolutionary Computation – GECCO-2003**. Chicago: Springer-Verlag, 2003. (LNCS, v. 2724), p. 1776–1787. ISBN 3-540-40603-4. Disponível em: <https://doi.org/10.1007/3-540-45110-2_69>.

SILVA, S.; DIGNUM, S.; VANNESCHI, L. Operator equalisation for bloat free genetic programming and a survey of bloat control methods. **Genetic Programming and Evolvable Machines**, v. 13, n. 2, p. 197–238, jun. 2012. ISSN 1389-2576. Disponível em: <<https://doi.org/10.1007/s10710-011-9150-5>>.

SILVA, S.; SILVA, P. J. N.; COSTA, E. Resource-limited genetic programming: Replacing tree depth limits. In: RIBEIRO, B. et al. (Ed.). **Adaptive and Natural Computing Algorithms**. Coimbra, Portugal: Springer, 2005. (Springer Computer Series), p. 243–246. ISBN 3-211-24934-6. Disponível em: <https://doi.org/doi:10.1007/3-211-27389-1_58>.

SMITH, P. W. H.; HARRIES, K. Code growth, explicitly defined introns, and alternative selection schemes. **Evolutionary Computation**, v. 6, n. 4, p. 339–360, Winter 1998. Disponível em: <<https://doi.org/10.1162/evco.1998.6.4.339>>.

SOBESTER, A.; NAIR, P.; KEANE, A. Evolving intervening variables for response surface approximations. In: **Proceedings of the 10th AIAA/ISSMO multidisciplinary analysis and optimization conference**. American Institute of Aeronautics and Astronautics, 2004. p. 1–12. AIAA 2004-4379. Disponível em: <<https://doi.org/10.2514/6.2004-4379>>.

SOTTO, L. F. dal P.; MELO, V. V. de. Studying bloat control and maintenance of effective code in linear genetic programming for symbolic regression. **Neurocomputing**, v. 180, p. 79–93, 2016. ISSN 0925-2312. Progress in Intelligent Systems Design Selected papers from the 4th Brazilian Conference on Intelligent Systems (BRACIS 2014). Disponível em: <<https://doi.org/10.1016/j.neucom.2015.10.109>>.

SOULE, T. **Code Growth in Genetic Programming**. Tese (Doutorado) — University of Idaho, Moscow, Idaho, USA, 15 maio 1998.

SOULE, T.; FOSTER, J. A. Removal bias: a new cause of code growth in tree based evolutionary programming. In: **1998 IEEE International Conference on Evolutionary Computation**. Anchorage, Alaska, USA: IEEE Press, 1998. p. 781–786. ISBN 0-7803-4869-9. Disponível em: <<https://doi.org/doi:10.1109/ICEC.1998.700151>>.

TACKETT, W. A. **Recombination, Selection, and the Genetic Construction of Computer Programs**. Tese (Doutorado), Los Angeles, CA, USA, 1994. Not available from Univ. Microfilms Int.

TURING, A. M. Computing machinery and intelligence. **Mind**, Oxford University Press on behalf of the Mind Association, v. 59, n. 236, p. 433–460, 1950. ISSN 00264423.

UY, N. Q. et al. Semantically-based crossover in genetic programming: application to real-valued symbolic regression. **Genetic Programming and Evolvable Machines**, v. 12, n. 2, p. 91–119, jun. 2011. ISSN 1389-2576. Disponível em: <<https://doi.org/10.1007/s10710-010-9121-2>>.

VANNESCHI, L. et al. A new implementation of geometric semantic gp and its application to problems in pharmacokinetics. In: KRAWIEC, K. et al. (Ed.). **Proceedings of the 16th European Conference on Genetic Programming, EuroGP 2013**. Vienna, Austria: Springer Verlag, 2013. (LNCS, v. 7831), p. 205–216. Disponível em: <https://doi.org/10.1007/978-3-642-37207-0_18>.

VLADISLAVLEVA, E. **Model-based Problem Solving through Symbolic Regression via Pareto Genetic Programming**. Tese (Doutorado) — Tilburg University, Tilburg, the Netherlands, ago. 2008.

WORM, T.; CHIU, K. Prioritized grammar enumeration: Symbolic regression by dynamic programming. In: **Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation**. New York, NY, USA: ACM, 2013. (GECCO '13), p. 1021–1028. ISBN 978-1-4503-1963-8. Disponível em: <<http://doi.acm.org/10.1145/2463372.2463486>>.

YAMAMOTO, L.; TSCHUDIN, C. F. Experiments on the automatic evolution of protocols using genetic programming. In: STAVRAKAKIS, I.; SMIRNOV, M. (Ed.). **Autonomic Communication, Second International IFIP Workshop, WAC 2005, Revised Selected Papers**. Athens, Greece: Springer, 2005. (Lecture Notes in Computer Science, v. 3854), p. 13–28. ISBN 3-540-32992-7. Disponível em: <https://doi.org/10.1007/11687818_2>.

ZHANG, B.-T.; MÜHLENBEIN, H. Balancing accuracy and parsimony in genetic programming. **Evolutionary Computation**, v. 3, n. 1, p. 17–38, 1995. Disponível em: <<https://doi.org/10.1162/evco.1995.3.1.17>>.

ZITZLER, E.; LAUMANN, M.; THIELE, L. SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. In: GIANNAKOGLU, K. C. et al. (Ed.). **Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems**. Athens, Greece: International Center for Numerical Methods in Engineering. p. 95–100. Disponível em: <<https://doi.org/10.1109/WDDC.2007.4339452>>.

ZITZLER, E.; THIELE, L. Multiobjective evolutionary algorithms: A comparative case study and strength pareto approach. v. 3, n. 4, 1999.

Apêndices

Principais técnicas de combate ao efeito bloat

Descrevemos aqui um resumo das principais abordagens já desenvolvidas para o combate do efeito *bloat*. As técnicas estão organizadas em quatro grupos, conforme seu embasamento: avaliação, seleção, recombinações especializadas e técnicas baseadas em sobrevivência. Muitas dessas abordagens foram também discutidas em Marques (2013).

A.1 Técnicas baseada na avaliação

Métodos que atuam no nível da avaliação dos indivíduos são comuns em Computação Evolucionária. No caso específico da Programação Genética, ao associar o tamanho de uma estrutura à sua aptidão, a probabilidade de seleção é afetada. Tais métodos são conhecidos por *pressão de parcimônia*. Estes métodos avaliam os indivíduos considerando a aptidão bruta e o tamanho, utilizando estas duas informações para penalizar indivíduos maiores (SILVA, 2008).

Seja $f(x)$ a aptidão de um programa x . Quando a pressão por parcimônia é aplicada, uma nova função de aptidão é definida:

$$f_p(x) = f(x) - c\ell(x) \quad (\text{A.1})$$

onde $\ell(x)$ é o tamanho do programa x e c é chamado *coeficiente de parcimônia* (POLI; MCPHEE, 2008)

A pressão por parcimônia já foi utilizada em diferentes pesquisas, com destaque para Blickle (1996), Iba, de Garis e Sato (1994), Kinnear, Jr. (1993) e Soule (1998). Poli e McPhee (2008) apresentaram importante contribuição ao desenvolver um método capaz de definir, de forma prática, ideal, e dinâmica, o coeficiente de parcimônia durante as várias evoluções de uma população

Embora as aplicações mais comuns utilizem a parcimônia como uma função linear e constante ao longo das evoluções, Zhang e Mühlenbein (1995) utilizaram uma versão com penalidade variável sobre a aptidão. Tal ajuste é complexo, pois valores paramétricos que são bons no início da evolução podem dificultar a busca por soluções de qualidade em gerações posteriores.

A.2 Técnicas baseadas na seleção

O combate ao efeito *bloat* em abordagens baseadas em seleção costuma utilizar técnicas de otimização multiobjetivo, com o intuito de maximizar a aptidão (por minimização do erro) e minimização do tamanho da árvore. Em casos como este, o conceito de *dominância de Pareto* pode ser útil: diz-se que a domina b (o que é representado por $a \prec b$) se, e somente se a não é pior que b em qualquer objetivo e a é estritamente melhor que b em pelo menos um objetivo.

A dominância é uma relação de ordem parcial e obedece apenas às propriedades irreflexiva ($a \prec b \Rightarrow a \neq b$), assimétrica ($a \prec b \Rightarrow \neg(b \prec a)$) e transitiva ($a \prec b \wedge b \prec c \Rightarrow a \prec c$). Isso significa que podem existir soluções a e b tais que a não domine b e nem b domine a , formando subconjunto de soluções não dominadas. O subconjunto que possui elementos que não são dominados por quaisquer outro elemento é chamado fronteira ótima de Pareto.

Detre as abordagens multiobjetivos em Computação Evolucionária, destacam-se o SPEA (ZITZLER; THIELE, 1999), o SPEA2 (ZITZLER; LAUMANN; THIELE,), aplicado por Bleuler et al. (2001) para controlar o efeito *bloat* em PG, e o NSGA-II (DEB et al., 2002), que foi utilizado por Vladislavleva (2008) e por Arnaldo, Krawiec e O'Reilly (2014) para controlar o crescimento das árvores em suas aplicações.

Diversos tipos de torneio foram também desenvolvidos para tentar conter o *bloat*. Em Luke e Panait (2002), por exemplo, é proposto uma forma de torneio denominada *pressão de parcimônia lexicográfica*, que modifica a seleção para preferir árvores menores somente quanto as condições físicas são iguais. Segundo Luke e Panait (2002), embora tenha se mostrada adequada para alguns domínios de problema, a pressão de parcimônia lexicográfica não apresentou bons resultados para regressão simbólica. Isso pode ser explicado pelo fato de as árvores, gradativamente maiores, costumam ser superiores apenas em aptidão.

Outra técnica baseada em torneio é o *torneio proporcional* (LUKE; PANAIT, 2006), na qual uma parte dos vencedores serão selecionados com base em seu tamanho ao invés da aptidão.

Por fim, há o *torneio duplo* (LUKE; PANAIT, 2006), que tem se mostrado como

um dos melhores métodos de controle de *bloat* disponíveis. (SILVA, 2008). Esta técnica consiste em utilizar um torneio para selecionar os participantes de um outro torneio sendo que o primeiro com base na aptidão e o segundo no tamanho – ou vice e versa.

A.3 Técnicas baseadas em recombinações especializadas

Diversos operadores genéticos foram propostos como forma de conter o *bloat*. O *Breeding* é uma técnica proposta por Tackett (1994) na qual o operador de recombinação gera muito mais descendentes que o necessário, selecionando diferentes pontos de cruzamento entre os pais. Dos filhos gerados, apenas os melhores são selecionados para sobreviver na próxima geração. Esta técnica promove a redução dos efeitos destrutivos do operador de cruzamento, mas demanda maior poder de processamento.

Há também a *recombinação homóloga*, que parte do princípio que os cruzamentos realizados em cadeias de DNA não são tão aleatórios quanto o que ocorre em PG. Os pontos de cruzamento localizam-se aproximadamente nas mesmas posições dos genes dos pais. Dessa forma, a PG pode produzir indivíduos com estruturas muito diferentes dos pais, o que pode ser problemático no sentido de se afastar de soluções viáveis: um trecho de código (subárvore) “bom” pode gerar resultados medianos ou ruins quando movido de forma puramente aleatória para outros contextos em outras árvores (O'REILLY; OPPACHER, 1994).

Estando entre os primeiros trabalhos que utilizam combinação homóloga, Yamamoto e Tschudin (2005) aplicam esse conceito para evoluir protocolos de comunicação de redes de computadores em uma aplicação no qual a população inicial era formada por programas que correspondiam a soluções de alta qualidade mas ainda não tão boas quanto o necessário. Como a recombinação homóloga consegue manter as propriedades do programa (ou seja, o contexto), foi possível realizar transformações, uma geração após a outra, gerando proles com funcionalidades similares, mas implementadas por diferentes caminhos, sem se afastar demais das características inicialmente propostas.

O conceito de recombinação homóloga foi implementado de diferentes maneiras, levando-se em consideração não somente as posições, mas também os formatos, os tamanhos e as profundidades das árvores envolvidas. Poli e Langdon (1998), por exemplo, propuseram um algoritmo em que os nós escolhidos como ponto de recombinação encontram-se dentro de regiões estruturalmente semelhantes dentro das duas árvores. Versões ainda mais restritas desse operador já haviam sido propostas em Poli e Langdon (1997), onde existe a obrigatoriedade de os pontos de recombinação representarem funções iguais.

Na recombinação de tamanho justo (*Size fair crossover*) a seleção do ponto de

recombinação da primeira árvore T_1 ocorre de maneira semelhante ao cruzamento convencional (com 90% de chance de escolha de nós função e 10% de nós terminais). Essa subárvore selecionada de T_1 dá lugar à subárvore selecionada na segunda árvore T_2 . A diferença reside em uma restrição na escolha do ponto de T_2 : se N_{t_1} é o tamanho da sub-árvore deletada de T_1 , então o nó selecionado em T_2 deve ser escolhido entre os que são raízes de subárvores de tamanho máximo $1 + 2N_{t_1}$ (LANGDON, 2000).

Outra variação para a recombinação é a baseada na preservação de contexto, proposta por D'haeseleer (1994), que utiliza uma definição de localização para determinar a região em que a recombinação trará menos modificações à estrutura original da árvore.

A.4 Técnicas baseadas em sobrevivência

A versão mais simples desta abordagem consiste em verificar algum atributo do novo indivíduo gerado a partir da aplicação dos operadores genéticos e decidir se ele deve ou não ser mantido nas gerações futuras. Koza (1992b) propôs um condição baseada na profundidade da árvore gerada, que seria aceita se o valor não ultrapasse certo limite pré-definido. Em caso negativo, um dos pais é transferido para a geração futura.

Silva e Almeida (2003) apresentaram uma melhoria para a abordagem estática, permitindo que o valor limite de profundidade fosse alterado dinamicamente. Sempre que o melhor indivíduo da população apresentar profundidade superior ao permitido, o valor máximo é atualizado, para comportá-lo.

Silva, Silva e Costa (2005) estendem o conceito de limites dinâmicos através da implementação de recursos limitados. Nesta abordagem, certa quantidade de nós é disponibilizada para formação das árvores que compõem a população. Quando esta quantidade é totalmente consumida, novos indivíduos não são aceitos, permitindo também que a população tenha tamanho variável. Estes métodos, que estão entre os que apresentam melhores resultados, são descritos em detalhes em Silva (2008).