UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

DIOGO MACHADO DE FREITAS

Geração Evolucionária de Heurísticas para Localização de Defeitos de Software

Goiânia
2018

sistema de bibliotecas ufg

PRPG
PRÓ-REITORIA DE PÓS-GRADUAÇÃO

UFG

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO PARA DISPONIBILIZAR VERSÕES ELETRÔNICAS DE TESES E DISSERTAÇÕES NA BIBLIOTECA DIGITAL DA UFG

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio da Biblioteca Digital de Teses e Dissertações (BDTD/UFG), regulamentada pela Resolução CEPEC nº 832/2007, sem ressarcimento dos direitos autorais, de acordo com a Lei nº 9610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou *download*, a título de divulgação da produção científica brasileira, a partir desta data.

1. Identificação do material bibliográfico: **[X] Dissertação** **[ ] Tese**

2. Identificação da Tese ou Dissertação:

Nome completo do autor:

Título do trabalho:

3. Informações de acesso ao documento:

Concorda com a liberação total do documento [X] SIM [ ] NÃO[1]

Havendo concordância com a disponibilização eletrônica, torna-se imprescindível o envio do(s) arquivo(s) em formato digital PDF da tese ou dissertação.

Assinatura do(a) autor(a)[2]

Ciente e de acordo:

Assinatura do(a) orientador(a)[2]

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. A extensão deste prazo suscita justificativa junto à coordenação do curso. Os dados do documento não serão disponibilizados durante o período de embargo.

Casos de embargo:
- Solicitação de registro de patente
- Submissão de artigo em revista científica
- Publicação como capítulo de livro
- Publicação da dissertação/tese em livro

[2] A assinatura deve ser escaneada.

DIOGO MACHADO DE FREITAS

Geração Evolucionária de Heurísticas para Localização de Defeitos de Software

Dissertação apresentada ao Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás, como requisito parcial para obtenção do título de Mestre em Programa de Pós-Graduação em Ciência da Computação.

Área de concentração: Ciência da Computação.

Orientador: Prof. Dr. Plínio de Sá Leitão-Júnior

Co-Orientador: Prof. Dr. Celso Gonçalves Camilo Junior

Goiânia
2018

Ficha de identificação da obra elaborada pelo autor, através do
Programa de Geração Automática do Sistema de Bibliotecas da UFG.

Machado de Freitas, Diogo
Geração Evolucionária de Heurísticas para Localização de Defeitos de Software [manuscrito] / Diogo Machado de Freitas, Plínio de Sá Leitão-Júnior, Celso Gonçalves Camilo Junior. - 2018.
LXXXIX, 89 f.: il.

Orientador: Prof. Dr. Plínio de Sá Leitão-Júnior ; co-orientador Dr. Celso Gonçalves Camilo Junior .
Dissertação (Mestrado) - Universidade Federal de Goiás, Instituto de Informática (INF), Programa de Pós-Graduação em Ciência da Computação, Goiânia, 2018.
Bibliografia. Apêndice.
Inclui tabelas, algoritmos, lista de figuras, lista de tabelas.

1. localização de defeitos baseada em espectro. 2. localização de defeitos baseada em busca. 3. engenharia de software baseada em busca. 4. programação genética. 5. meta-heurísticas evolucionárias. I. de Sá Leitão-Júnior, Plínio. II. Gonçalves Camilo Junior, Celso. III. , Plínio de Sá Leitão-Júnior, orient. IV. , Celso Gonçalves Camilo Junior, co-orient. V. Título.

CDU 004

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UFG

ATA Nº 10/2018

ATA DA SESSÃO DE JULGAMENTO DA DISSERTAÇÃO DE MESTRADO DE DIOGO MACHADO DE FREITAS

Aos vinte e quatro dias do mês de setembro de dois mil e dezoito, às dezenove horas, na sala 151 do Instituto de Informática da Universidade Federal de Goiás, Campus Samambaia, reuniu-se a banca examinadora designada na forma regimental pela Coordenação do Curso para julgar a dissertação de mestrado intitulada **"Geração Evolucionária de Heurísticas para Localização de Defeitos de Software"**, apresentada pelo aluno Diogo Machado de Freitas como parte dos requisitos necessários à obtenção do grau de Mestre em Ciência da Computação, área de concentração Ciência da Computação. A banca examinadora foi presidida pelo orientador do trabalho de dissertação, Professor Doutor Plínio de Sá Leitão Júnior (INF/UFG), tendo como membros os Professores Doutores Celso Gonçalves Camilo Júnior (INF/UFG – coorientador), Anderson da Silva Soares (INF/UFG) e Sílvia Regina Vergílio (DINF/UFPR). A professora Sílvia Regina Vergílio participou à distância por webconferência. Aberta a sessão, o candidato expôs seu trabalho. Em seguida, o aluno foi arguido pelos membros da banca e:

(X) tendo demonstrado suficiência de conhecimento e capacidade de sistematização do tema de sua dissertação, a banca concluiu pela **aprovação** do candidato, sem restrições.

( ) não tendo demonstrado suficiência de conhecimento e capacidade de sistematização do tema de sua dissertação, a banca concluiu pela **reprovação** do candidato.

Os trabalhos foram encerrados às 21:15 horas. Nos termos do Regulamento Geral dos Cursos de Pós-Graduação desta Universidade, lavrou-se a presente ata que, lida e julgada conforme, segue assinada pelos membros da banca examinadora.

Prof. Dr. Plínio de Sá Leitão Júnior _____

Prof. Dr. Celso Gonçalves Camilo Júnior _____

Prof. Dr. Anderson da Silva Soares _____

Profa. Dra. Sílvia Regina Vergílio _____

Diogo Machado de Freitas

Graduou–se em Engenharia de Computação pela UFG - Universidade Federal de Goiás. Durante sua graduação foi bolsista do Programa Institucional de Iniciação Científica (PIBIC) no Instituto de Informática e trabalhou com critérios de seleção de dados de teste. Durante o mestrado, na UFG - Universidade Federal de Goiás, foi bolsista CAPES e trabalhou com Localização de Defeitos e Engenharia de Software Baseada em Busca. Integrou o grupo de pesquisa I4Soft - Intelligence for Software.

À minha família.

Agradecimentos

Primeiramente a Deus, por ter me dado tudo o que precisei e mais do que mereço, sempre na hora certa.

À minha família, especialmente meus pais pelo exemplo de conduta, plena confiança e apoio, meu irmão pela amizade incondicional e minha namorada, Jacqueline, pela constante motivação e companheirismo sincero.

Aos meus amigos sempre presentes, especialmente Axel Miguez, Guilherme Porto e Matheus Feliciano, que compreenderam muitas ausências.

Ao meu orientador de longa data, professor Plínio Leitão, pelos ensinamentos e instrução desde a graduação até o mestrado.

Ao meu co-orientador, professor Celso Camilo, pela perspicácia e valiosos direcionamentos.

A todos os colegas de grupo de pesquisa e de laboratório, especialmente meu amigo Altino Dantas, que sempre contribuíram com importantes discussões e direcionamentos.

Aos meus co-autores de artigos por compartilhar do trabalho árduo, frustrações e conquistas.

À Universidade Federal de Goiás, o Instituto de Informática, seu corpo docente, diretores e técnicos administrativos pela estrutura e manutenção das condições necessárias ao estudo. À professora Telma Woerle e a Mariana Santana, sempre solícitas e responsáveis diante de suas atribuições administrativas no programa de pós-graduação. À CAPES pelo fomento.

Aos professores que compõem a banca examinadora, por contribuir com o trabalho e seu desenvolvimento.

A educação tem raízes amargas, mas os seus frutos são doces.

Aristóteles,

.

Resumo

de Freitas, Diogo Machado. **Geração Evolucionária de Heurísticas para Localização de Defeitos de Software**. Goiânia, 2018. 89p. Dissertação de Mestrado. Instituto de Informática, Universidade Federal de Goiás.

Localização de Defeitos é uma etapa do ciclo de vida de software, que demanda recursos importantes tais como o tempo e o esforço gastos em um projeto. Existem diversas iniciativas na direção da automação do processo de localização de defeitos e da redução dos recursos associados. Muitas técnicas são baseadas heurísticas que utilizam informação obtida (espectro) a partir da execução de casos de teste, visando a medir a suspeita de cada elemento de programa para ser defeituoso. Os dados de espectro referem-se, em geral, à cobertura de código e aos resultados dos teste (positivo ou negativo). O presente trabalho apresenta duas abordagens baseadas no algoritmo *Programação Genética* para o problema de Localização de Defeitos: um método para compor automaticamente novas heurísticas a partir de um conjunto de heurísticas existentes; e um método para a construção de heurísticas baseadas em dados oriundos da análise de mutação de programas. Os aspectos inovadores de ambos os métodos referem-se à investigação conjunta de: (i) especialização de heurísticas para determinados programas; (ii) aplicação de abordagem evolutiva para a geração de heurísticas com equações não lineares; (iii) criação de heurísticas a partir da combinação de heurísticas tradicionais; (iv) uso de espectro de cobertura e de mutação extraídos da atividade de teste; (v) análise e comparação da eficácia de métodos que usam os espectros de cobertura e de mutação para a localização de defeitos; e (vi) análise da qualidade dos espectros de mutação como fonte de dados para a localização de defeitos. Os resultados apontaram competitividade de ambas as abordagens em seus contextos.

Palavras–chave

localização de defeitos baseada em espectro, localização de defeitos baseada em busca, engenharia de software baseada em busca, programação genética, meta-heurísticas evolucionárias

Abstract

de Freitas, Diogo Machado. **Evolutionary Generation of Heuristics for Software Fault Localization**. Goiânia, 2018. 89p. MSc. Dissertation. Instituto de Informática, Universidade Federal de Goiás.

Fault Localization is one stage of the software life cycle, which demands important resources such as time and effort spent on a project. There are several initiatives towards the automation of the fault localization process and the reduction of the associated resources. Many techniques are based on heuristics that use information obtained (spectrum) from the execution of test cases, in order to measure the suspiciousness of each program element to be defective. Spectrum data generally refers to code coverage and test results (positive or negative). The present work presents two approaches based on the *Genetic Programming* algorithm for the problem of Fault Localization: a method to compose a new heuristic from a set of existing ones; and a method for constructing heuristics based on data from program mutation analysis. The innovative aspects of both methods refer to the joint investigation of: (i) specialization of heuristics for certain programs; (ii) application of an evolutionary approach to the generation of heuristics with non-linear equations; (iii) creation of heuristics based on the combination of traditional heuristics; (iv) use of coverage and mutation spectra extracted from the test activity; (v) analyzing and comparing the efficacy of methods that use coverage and mutation spectra for fault localization; and (vi) quality analysis of the mutation spectra as a data source for fault localization. The results have pointed to the competitiveness of both approaches in their contexts.

Keywords

spectrum-based fault localization, search-based fault localization, search-based software engineering, genetic programming, evolutionary metaheuristics

Sumário

Lista de Figuras **13**

Lista de Tabelas **14**

1 Introdução **16**
- 1.1 Motivação 17
- 1.2 Objetivos Gerais 18
- 1.3 Objetivos Específicos 18
- 1.4 Organização do Trabalho 19

2 Contexto e Estudos Relacionados **20**
- 2.1 Depuração de Software 20
- 2.2 Localização de Defeitos 21
- 2.3 Localização de Defeitos Baseada em Espectro de Cobertura 22
- 2.4 Localização de Defeitos Baseada em Espectro de Mutação 24
- 2.5 Engenharia de Software Baseada em Busca 26
- 2.6 Meta-heurísticas 28
  - 2.6.1 Algoritmos Evolucionários 29
    - Algoritmo Genético 30
    - Programação Genética 31
- 2.7 Localização de Defeitos Baseada em Busca 32
- 2.8 Considerações Finais 34

3 Composição de Heurísticas Baseadas em Espectro de Cobertura **36**
- 3.1 Método Proposto: Aplicação de Programação Genética para Combinar Heurísticas SBFL 37
- 3.2 Experimentos 41
  - 3.2.1 Os *Benchmarks* 42
  - 3.2.2 Parâmetros das Meta-heurísticas 43
  - 3.2.3 Avaliação do Método 43
  - 3.2.4 Métricas de Avaliação 44
- 3.3 Análise 45
  - 3.3.1 Avaliação do Conjunto de Programas Completo 46
  - 3.3.2 Avaliação de *Rank* Médio 49
  - 3.3.3 Avaliação de Programas Individuais 51
  - 3.3.4 Análise Estatística 54
  - 3.3.5 Ameaças à Validade 58
- 3.4 Considerações Finais 58

4 Geração de Heurísticas Baseadas em Espectro de Mutação **60**
4.1 Método Proposto: Aplicação de Programação Genética à Heurísticas MBFL 61
4.2 Experimentos 62
4.2.1 Os *Benchmarks* 63
4.2.2 Parâmetros das Meta-heurísticas 65
4.2.3 Avaliação do Método 65
4.2.4 Métricas de Avaliação 65
4.3 Análise 66
4.3.1 Eficácia de Heurísticas MBFL Treinadas Evolucionariamente 66
Avaliação do Conjunto de Programas Completo 67
Avaliação de *Rank* Médio 68
Avaliação de Programas Individuais 69
4.3.2 Qualidade do Espectro de Mutação 70
4.3.3 Análise Estatística 72
4.3.4 Ameaças à Validade 73
4.4 Considerações Finais 74

5 Conclusão **76**

Referências Bibliográficas **79**

A Dados da Métrica *Rank* Médio **85**

B Dados da Métrica *Expense Médio* **88**

Lista de Figuras

2.1 Exemplo de espectro de cobertura de programa [3]. 22
2.2 Exemplo de aplicação de heurística MBFL ao cálculo de suspeita de elementos de código. 25
2.3 Esquema iterativo dos algoritmos evolucionários. 29

3.1 Exemplo de solução encontrada (Fórmula 3-2). 40
3.2 Exemplo de solução encontrada (Fórmula 3-3). 40
3.3 Esquema de experimentação utilizado em composição de heurísticas baseadas em espectro de cobertura. 42

4.1 Esquema de experimentação utilizado para geração de heurísticas baseadas em espectro de mutação. 63

Lista de Tabelas

2.1 Exemplo de um programa e sua matriz de cobertura 24

3.1 Métricas de associação utilizadas pelos algoritmos de composição 38
3.2 Heurísticas SBFL utilizadas pelos algoritmos de composição 39
3.3 Conjunto de programas utilizado nos experimentos 42
3.4 Principais medidas de avaliação para o conjunto completo de programas ao avaliar PG_{T1}, AG_{T1} e as heurísticas SBFL individuais que compõem o Conjunto de Terminais T1 47
3.5 Principais medidas de avaliação para o conjunto completo de programas ao avaliar PG_{T2}, AG_{T2} e as heurísticas SBFL individuais que compõem o Conjunto de Terminais T2 48
3.6 Cinco melhores valores de *rank* médio para as heurísticas que compõem o conjunto de terminais $T1$ e os métodos de composição a ele relacionados (PG_{T1} e AG_{T1}) 49
3.7 Cinco melhores valores de *rank* médio para as heurísticas que compõem o conjunto de terminais $T2$ e os métodos de composição a ele relacionados (PG_{T2} e AG_{T2}) 50
3.8 Cinco melhores valores de *rank* médio para todas as heurísticas apresentadas (presentes nos conjuntos de terminais $T1$ e $T2$) e as técnicas de composição 51
3.9 Proporção média de código investigado para encontrar todos os defeitos: *Expense* Médio 51
3.10 Proporção média de código investigado para encontrar todos os defeitos: *Expense* Médio 52
3.11 Cinco melhores técnicas em valores de *precision* (*acc@n*) e *wasted effort* (*wef@n*) dentre as heurísticas SBFL individuais que compõem o conjunto de terminais T1, PG_{T1} e AG_{T1} 53
3.12 Cinco melhores técnicas em valores de *precision* (*acc@n*) e *wasted effort* (*wef@n*) dentre as heurísticas SBFL individuais que compõem o conjunto de terminais T2, PG_{T2} e AG_{T2} 55
3.13 Teste estatístico de *Wilcoxon* e de $\hat{A}_{12}$ *Vargha* e *Delaney* com AG_{T1} (A_1) e PG_{T1} (A_2) 56
3.14 Teste estatístico de *Wilcoxon* e de $\hat{A}_{12}$ *Vargha* e *Delaney* com AG_{T2} (A_1) e PG_{T2} (A_2) 57

4.1 Conjunto de programas usado nos experimentos 64
4.2 Medidas de avaliação para o conjunto de programas avaliados 67
4.3 *Rank* médio de *expense* médio, *acc@n* e *wef@n* 68
4.4 Comparação dos métodos avaliados em programas individuais 69

4.5 Versões do programa *Math* em relação ao número médio mínimo de mutantes K em comandos defeituosos 71
4.6 Medidas de avaliação: Programa *Math* (Defects4j) 72
4.7 Teste de Wilcoxon e de Vargha e Delaney $\hat{A}_{12}$ para PG-mut (A_1) e PG-cov (A_2) 73

A.1 *Rank* médio para as heurísticas que compõem o conjunto de terminais $T1$ e os métodos de composição a ele relacionados (GP_{T1} e GA_{T1}) 85
A.2 *Rank* médio para as heurísticas que compõem o conjunto de terminais $T2$ e os métodos de composição a ele relacionados (GP_{T2} e GA_{T2}) 86
A.3 *Rank* médio para todas as heurísticas apresentadas (presentes nos conjuntos de terminais $T1$ e $T2$) e as duas variações de ambas as técnicas de composição (GP_{T1}, GA_{T1}, GP_{T2} e GA_{T2}) 87

B.1 Proporção média de código investigado para encontrar todos os defeitos (*Expense* Médio) para as heurísticas que compõem o conjunto de terminais $T1$ e os métodos de composição a ele relacionados (GP_{T1} e GA_{T1}) 88
B.2 Proporção média de código investigado para encontrar todos os defeitos (*Expense* Médio) para as heurísticas que compõem o conjunto de terminais $T2$ e os métodos de composição a ele relacionados (GP_{T2} e GA_{T2}) 89

CAPÍTULO 1

Introdução

O desenvolvimento e posterior manutenção de um software costuma ser realizado pela colaboração de diversas pessoas, que naturalmente têm diferentes níveis de habilidades, entendimento sobre o projeto, responsabilidades e objetivos. Por estes e outros fatores, existe a inegável disposição para que defeitos sejam introduzidos no produto e que o mesmo não corresponda às suas especificações. Dada a inerente propensão do software conter defeitos, há demanda pelas atividades de teste, verificação e depuração de código, que podem consumir de 50% a 75% do orçamento de um projeto [17].

Conforme a norma internacional *ISO/IEC/IEEE 29119-1* [24], defeito é a característica incorreta do programa que em determinadas condições leva à falha. Um defeito não tem impacto sobre o funcionamento do software, se o mesmo não provocar uma falha. Por sua vez, falha é algum tipo de anomalia que se manifesta durante a execução de um programa decorrente de algum defeito ou combinação de defeitos alcançados. É a observação da falha que permite a asserção de que o programa contém defeitos.

Segundo Hailpern *et al.* [17], depuração é um processo que envolve analisar e modificar um software que não atende suas especificações para que o mesmo as satisfaça. A depuração inicia-se após a identificação de um defeito pelo teste de software e compreende principalmente localização e correção de defeitos.

Localização de defeitos refere-se à atividade de identificação da localidade dos defeitos correspondentes às falhas que se tornaram conhecidas durante a etapa de teste de software. Devido à crescente complexidade dos projetos de software, localização de defeitos torna-se cada vez mais uma tarefa onerosa e delongada [50].

Existe uma variedade de métodos para localização de defeitos [53]. Dentre os métodos mais tradicionais estão: *Program Logging*, em que variáveis ou pontos de execução são monitorados com inserção de comandos *print*; ou *Breakpoints*, nos quais a execução do programa é interrompida para que o seu comportamento seja analisado. Os métodos tradicionais são simplistas e representam o modo mais comum de localização de defeitos empregado, e têm alto custo.

Há ainda métodos avançados, que foram classificadas por Wong *et al.* [53] em: *slices*; espectro de código; estatística; estados de programa; aprendizado de máquina; mi-

neração de dados; modelos *etc*. Estes métodos representam esforços para automatizar o trabalho de localização de defeitos e reduzir o custo demandado pelos métodos tradicionais.

Métodos baseados em espectro de código são o tema do presente trabalho. Estes métodos consistem na aplicação de heurísticas para inferir a probabilidade de cada elemento do código ser defeituoso a partir de seu espectro. Espectro de código refere-se a uma coleção de dados sobre o comportamento da execução de um programa pelo respectivo conjunto de teste [3]. A maioria dos métodos de localização de defeitos baseada em espectro utilizam dados do *Espectro de Cobertura* do código, que consiste em informações sobre quantos testes positivos e negativos executaram ou não cada elemento.

Além do espectro de cobertura, outras publicações trabalharam a utilização de outros tipos de dados. Os dados de análise de mutantes são utilizados para compor o *Espectro de Mutação*, que é aplicado aos métodos de localização de defeitos baseada em mutação [42].

A problemas de diversas áreas da Engenharia de Software, otimização baseada em busca tem sido amplamente aplicada em uma disciplina conhecida como *Engenharia de Software Baseada em Busca*. Harman et. al. demonstraram que os problemas em Engenharia de Software são tipicamente relacionados a otimização e podem ser modelados para a aplicação de técnicas de busca [18]. Localização de defeitos apresenta problemas que podem ser tratados como problemas de busca, como construir a melhor heurística para um determinado programa. Assim, muitas atividades de localização de defeitos, como geração e seleção de casos de teste e geração e combinação de heurísticas de localização de defeitos, têm sido abordados pela literatura ao longo dos anos.

1.1 Motivação

Wang *et al.* [51] apresentaram um método seminal para localização de defeitos baseada em busca, pela combinação linear de heurísticas conhecidas da literatura, o qual é uma expressão construída a partir de um conjunto de termos: cada termo é multiplicado por um peso, cujo valor é determinado utilizando-se um Algoritmo Genético. O método é estruturado em duas fases, treinamento e implantação. Durante o treinamento, o algoritmo busca a heurística de melhor desempenho para um conjunto de programas com defeitos conhecidos. Posteriormente, inicia-se a fase de implantação, em que a heurística obtida pelo método durante o treinamento é aplicada às novas versões defeituosas do programa, da mesma forma que qualquer outra heurística baseada em espectro de cobertura.

Outra publicação importante para o campo de localização de defeitos baseada em busca foi a de Yoo [55], que introduziu a aplicação do algoritmo de Programação Genética à construção de heurísticas ao combinar operações matemáticas básicas e as

principais variáveis do espectro de cobertura. O estudo mais recente acerca da análise teórica e empírica da utilização de Programação Genética à construção de heurísticas de localização de defeitos apontou os seguintes trabalhos futuros [56]:

- *construir fórmulas que sejam eficazes em certos contextos*, de modo que as heurísticas sejam geradas para projetos específicos (heurísticas orientadas a programas) e não para conjuntos de projetos (heurísticas genéricas).
- *utilizar sinais além de espectro de programa*, ou seja, utilizar outras fontes de dados diferentes dos espectros de cobertura, tais como o espectro de mutação.

1.2 Objetivos Gerais

O objetivo do presente trabalho é contribuir à área de localização de defeitos baseada em busca, ao abordar diretamente as lacunas demarcadas por [56]. Para tal, dois métodos foram propostos e avaliados: (i) combinação não linear de heurísticas baseadas em espectro de cobertura treinadas para projetos específicos; e (i) construção de heurísticas baseadas em espectro de mutação treinadas para projetos específicos.

A avaliação dos métodos propostos traça uma investigação quanto à eficácia dos mesmos, sob duas perspectivas: *perspectiva relativa*, pela análise do custo médio para localização de defeitos em relação ao total de comandos existentes; e *perspectiva absoluta*, pela análise do esforço total em termos da quantidade comandos investigados para localizar defeitos ou atingir um limite de tentativas. Vale ressaltar que as métricas de avaliação de eficácia absoluta melhor se adaptam ao interesse do profissional que executa a depuração de software.

1.3 Objetivos Específicos

O primeiro método proposto trata a composição de heurísticas baseadas em espectro de cobertura e culminou em publicações no 8^o Workshop de Engenharia de Software Baseada em Busca (*WESB 2017*) [8] e no *26th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning* (*ESANN 2018*) [9]. Os objetivos desta proposta consiste na conciliação das seguintes premissas em uma abordagem única:

- composição não linear de heurísticas baseadas em espectro de cobertura;
- heurísticas baseadas em espectro de cobertura combinadas e customizadas para projetos específicos;
- fórmulas compostas por heurísticas existentes de localização de defeitos e variáveis básicas de espectro de cobertura.

O segundo método proposto trata a geração de heurísticas baseadas em espectro de mutação e culminou em uma publicação no *2018 IEEE Congress on Evolutionary Computation* (IEEE CEC 2018) [10]. Os objetivos desta proposta circundam as lacunas apontadas em [56]:

- um método treinado a projetos específicos, para geração de heurísticas baseadas em espectro de mutação, visando à localização de defeitos possíveis;
- uma comparação da eficácia entre métodos de geração de heurísticas para espectros de cobertura e de mutação;
- uma investigação inicial acerca da qualidade dos dados do espectro de mutação e seu impacto ao método.

1.4 Organização do Trabalho

O texto está estruturado da seguinte maneira: O Capítulo 2 apresenta os principais trabalhos relacionados; O Capítulo 3 propõe e analisa o método de composição não linear de heurísticas baseadas em espectro de cobertura; O Capítulo 4 propõe e analisa o método para geração de heurísticas baseadas em espectro de mutação e o Capítulo 3.4 conclui o trabalho.

CAPÍTULO 2

Contexto e Estudos Relacionados

O aumento da complexidade dos projetos de software e da quantidade de pessoas envolvidas nos processo de desenvolvimento e manutenção torna a introdução de defeitos no produto final altamente provável. A estratégia adotada para aprimorar a confiabilidade do software é garantir que sua operação esteja em conformidade com a especificação de requisitos funcionais e não funcionais. Para validar o software conforme sua especificação são aplicadas as atividades de depuração, teste e verificação que, segundo Hailpern *et al.* [17], podem consumir de 50% a 75% do custo total de um projeto. Assim, técnicas para otimizar estas tarefas são desejáveis.

Teste de software envolve esforços que buscam revelar comportamentos de um programa que violem as especificações de requisitos [17]. O teste pode contemplar ou não a execução do código, de modo que atividades como análise de modelos e execução de casos de teste são consideradas teste. Assim, o teste está presente em diversas etapas do ciclo de desenvolvimento e de validação do software.

Verificação consiste no processo de provar ou demonstrar que um programa satisfaz suas especificações corretamente [36]. Verificação formal é pouco praticada na industria [17], visto que poucos projetos contam com uma especificação formal de requisitos do sistema para a comparação das funções necessárias às que de fato foram implementadas no software.

Depuração é o processo de diagnosticar a natureza de um defeito conhecido e então corrigi-lo [36]. Ou seja, trata-se da eliminação dos defeitos detectados e compreende localizá-lo e posteriormente repará-lo [37]. É extremamente ligado ao teste de software, visto que defeitos necessitam ser identificados previamente ao seu reparo.

2.1 Depuração de Software

Segundo Hailpern *et al.* [17], depuração de software envolve analisar e modificar um programa que não corresponde a sua especificação. Assim, o objetivo básico é encontrar uma nova versão do programa que seja próxima do original, mas que satisfaça os requisitos antes violados.

Depuração é um processo de dois passos que se inicia quando um defeito é identificado por um teste negativo, ou seja, um teste que revela a existência de um defeito [37]. O primeiro passo consiste na determinação precisa da localização do defeito no programa, enquanto o segundo passo consiste no reparo do mesmo.

Assim, depuração busca reduzir o número de defeitos existentes no software e consome uma significativa fatia do tempo gasto durante o ciclo de desenvolvimento [17]. Demanda tempo de trabalho e envolve alta complexidade, tornando-se assim oneroso ao custo do projeto [50]. Isso motivou a pesquisa e o desenvolvimento de novas técnicas que permitem automatizar a depuração, de modo que a intervenção humana é minimizada e eventualmente desnecessária.

Myers *et al.* [37] estimam que a localização de defeitos representa 95% do problema de depuração, ou seja, o problema de localização de defeitos na prática consome mais recursos e tempo que o reparo do defeito em si.

2.2 Localização de Defeitos

Localização de defeitos é uma parte importante em depuração que obrigatoriamente precede o reparo de qualquer defeito. Dentre as tarefas inerentes à depuração, localizar defeitos é uma das mais difíceis e tediosas [25]. A localização de defeitos afeta diretamente a qualidade e o custo do produto e refere-se à atividade de identificação da localidade dos defeitos correspondentes às falhas que se tornaram conhecidas durante a etapa de teste de software.

Wong *et al.* [53] realizaram um estudo que cobriu 331 artigos publicados entre Janeiro de 1977 e Novembro de 2014, que representam as principais publicações relacionadas à localização de defeitos no período. A pesquisa dividiu as técnicas de localização de defeitos em dois grupos: as tradicionais, que são técnicas manuais e intuitivas, e as avançadas, que são técnicas de maior complexidade e eficiência quando aplicadas a sistemas de grande porte.

As técnicas tradicionais, conforme classificadas por Wong *et al.* [53], são: *Program Logging* (monitorar variáveis ou pontos de execução através de comandos como *print*), *Assertions* (condições adicionadas para terminar a execução do programa quando um valor incorreto é atingido), *Breakpoints* (pontos em que a execução do programa é interrompida de modo que permita ao programador avaliar seu estado) e *Profiling* (monitoramento de métricas como utilização de memória, velocidade de execução, chamada de funções, etc.).

As técnicas avançadas, conforme classificação da publicação [53], são baseadas em: *slices*; espectro de código; estatística; estados de programa; aprendizado de máquina; mineração de dados; modelos e outras técnicas. As técnicas avançadas têm por objetivo

automatizar e reduzir a intervenção humana envolvida, ainda que parcialmente, no processo de localização de defeitos de software. A necessidade pelo desenvolvimento de tais técnicas se dá em razão do aumento do custo e do volume das atividades de localização de defeitos diante da crescente complexidade dos projetos de software.

2.3 Localização de Defeitos Baseada em Espectro de Cobertura

De modo geral, as principais propostas para automatizar o processo de localização de defeitos baseiam-se na execução dos casos de teste para inferir a probabilidade de cada elemento de programa ser o responsável pela(s) falha(s) observada(s). As heurísticas de localização de defeitos baseada em espectro (*SBFL*, do Inglês *Spectrum-based Fault Localization*) compõem uma das principais vertentes de pesquisa quanto à automatização do processo de localização de defeitos. Trata-se da proposição de fórmulas baseadas em espectro de cobertura para calcular a probabilidade de que um elemento de código seja defeituoso.

Define-se um *Espectro de Programa* como um conjunto de informações acerca da execução do mesmo. Esses dados são coletados em tempo de execução (teste de software) e podem consistir de diversos contadores ou *flags* para as diferentes partes de um programa [3]. Dentre os sinais que compreendem o espectro de um programa, estão os dados de cobertura de código pelos casos de teste: o espectro de cobertura.

$$\begin{array}{cc} & \overbrace{\qquad\qquad\qquad}^{n \text{ elementos}} \\ m \text{ execuções} & \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix} \end{array} \quad \begin{array}{c} \text{falhas} \\ \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_m \end{bmatrix} \end{array}$$

$$S(e_1) \quad S(e_2) \quad \cdots \quad S(e_n)$$

Figura 2.1: *Exemplo de espectro de cobertura de programa [3].*

A Figura 2.1, extraída de [3], apresenta os dados de espectro de cobertura de um programa. Um conjunto de m execuções de um programa composto por n elementos de código é representado por uma matriz $m \times n$, que indica quais elementos foram cobertos por quais execuções. Um vetor de tamanho m indica quais execuções resultaram em falhas. $S(e)$ representa a propensão a defeitos (suspeita) de e, ou seja, probabilidade de o elemento e ser defeituoso. Para cada elemento de código e, a probabilidade de o mesmo ser defeituoso é calculada por uma heurística SBFL. A partir dos dados de espectro de cobertura, as seguintes variáveis são comumente derivadas:

- c_{ep}: número de execuções bem-sucedidas do elemento e;
- c_{ef}: número de execuções malsucedidas do elemento e;
- c_{np}: o número de execuções bem-sucedidas do programa e que não executam o elemento e
- c_{nf}:o número de execuções malsucedidas do programa e que não executam o elemento e.

A ferramenta proposta por Jones *et al.* [26] foi pioneira ao coletar informações de cobertura de código pela execução de casos de teste para tentar guiar a atenção do programador aos elementos com maior chance de conter o defeito sob análise. Ela se guia por uma heurística conhecida como Tarantula, uma fórmula matemática que relaciona a ocorrência da falha observada à execução de cada elemento de código pelo caso de teste e calcula uma suspeita de que o mesmo contenha o defeito investigado. Assim, a ferramenta aplica um código de cores no espectro de verde a vermelho para destacar os elementos de menor e maior suspeita respectivamente.

A heurística Tarantula (Fórmula 2-1) é precursora das técnicas de **Localização de Defeitos Baseada em *Espectro***, ou Espectro de Cobertura, e foi introduzida por Jones *et al.* [26] através de sua ferramenta para visualização de código de suspeito. A heurística foi construída visando a penalizar os elementos de código que são executados com maior frequência por testes negativos em relação aos elementos que são menos executados por este tipo de teste.

$$S(e) = \frac{\frac{c_{ef}}{(c_{ef}+c_{nf})}}{\frac{c_{ep}}{(c_{ep}+c_{np})} + \frac{c_{ef}}{(c_{ef}+c_{nf})}} \tag{2-1}$$

Para cada elemento e de um programa, uma heurística SBFL calcula a suspeita $S(e)$, que efetivamente representa a probabilidade de que o elemento seja responsável pela falha observada e portanto defeituoso. Assim, a suspeita $S(e)$ reflete a associação entre e e a ocorrência de falhas. Após calcular $S(e)$ para todo e de um programa, um ranqueamento é construído em ordem não crescente de modo que o programador responsável pela depuração possa analisar todos os elementos, iniciando pelos de maior suspeita, até encontrar o que de fato contém o defeito em questão.

A Tabela 2.1 apresenta um exemplo de um programa simples e o comportamento da execução de seu conjunto de teste, ou seja, sua matriz de cobertura. Cada ● representa uma linha de código que foi executada pelo dado de teste correspondente. O resultado do teste pode ser bem-sucedido (✓) ou malsucedido (✗). De posse destes dados, uma heurística SBFL pode ser utilizada para se construir um ranking ordenado das linhas de código de modo a priorizar as de maior suspeita ao analisar o código. A Fórmula 2-1 retorna suspeita $0,5$ para a linha 1, $1,0$ para a linha número 2, defeituosa e $0,0$ para as linhas 3 e 4. Portanto, o ranking obtido tem a seguinte ordem: linha 2 no topo, linha 1 na segunda posição e linhas 3 e 4 na terceira posição. Assim como neste exemplo, espera-

se que as heurísticas SBFL sejam capazes de colocar a linha defeituosa mais próxima do topo do ranking, com valores maiores de suspeita para indicar quais elementos tem a maior probabilidade de terem sua execução associada ao defeito. A ideia é poupar tempo e esforço do desenvolvedor ao priorizar a análise da linha defeituosa durante o processo de localização do defeito.

Table 2.1: *Exemplo de um programa e sua matriz de cobertura*

ID	Código	n = 1	n = 2	n = 3	n = 4
1	if (n % 2 == 0):	●	●	●	●
2	x = n + 1 # bug		●		●
3	else:	●		●	
4	x = n - 1	●		●	
	Result:	✓	✗	✓	✗

Diversas outras heurísticas SBFL foram propostas ao longo do tempo, dentre elas está o coeficiente de Ochiai, amplamente referenciada na literatura. Adaptado da biologia molecular, onde originalmente foi utilizada para o cálculo de similaridade genética. O coeficiente de Ochiai foi introduzido como heurística de localização de defeitos por Abreu *et al.* [1], conforme definida na Fórmula 2-2.

$$S(e) = \frac{c_{ef}}{\sqrt{(c_{ef} + c_{nf}) \times (c_{ef} + c_{ep})}} \tag{2-2}$$

Outro trabalho importante para a adaptação de heurísticas SBFL, por Lucia *et al.* [32], investigou 20 métricas de associação. As métricas de associação são ferramentas da estatística utilizadas para medir a força de associação entre duas variáveis, por exemplo a associação é entre a execução (ou não execução) de um elemento de código e o resultado do teste (se positivo ou negativo). Lucia *et al.* [32] compararam as 20 métricas de associação a Tarântula e Ochiai.

2.4 Localização de Defeitos Baseada em Espectro de Mutação

Além das técnicas de localização de defeitos baseadas em espectro de cobertura, algumas pesquisas trabalharam a aplicação da análise de mutantes ao problema de localização de defeitos [42, 35, 43]. Buscando melhorar o desempenho de heurísticas de localização de defeitos, dados de análise de mutantes foram utilizados para compor **Espectro de Mutação**, análogo ao Espectro de Cobertura em sua aplicação às heurísticas de localização de defeitos. Esta área é conhecida como **Localização de Defeitos Baseada em Mutação** (*MBFL*, do Inglês: *Mutation-based Fault Localization*).

A Análise de Mutantes é uma ferramenta utilizada para avaliar a qualidade de conjuntos de casos de teste [11, 4]. São aplicadas diversas modificações (mutações) pontuais a um programa, cada modificação gera um mutante, isto é, uma versão modificada do programa original. Após executar cada mutante com as entradas do conjunto de casos de teste, estes são avaliados e comparados ao programa original. Se a saída de um mutante é diferente do programa original, diz-se que o mutante foi "*morto*". O indicador resultante da análise de mutantes do conjunto de casos de teste é o **Escore de mutação**, que é a proporção de mutantes mortos em relação aos mutantes não equivalentes. Um mutante é chamado de *equivalente* se não houver um caso de teste capaz distinguir suas saídas em relação ao programa original.

MBFL foi proposta em [42] como uma combinação de análise de mutantes e SBFL. Os operadores de mutação são aplicados a todas as linhas de código que foram cobertas por algum caso de teste negativo. Posteriormente cada mutante obtido é executado pelo conjunto de testes e informações acerca de quais testes mataram quais mutantes são coletadas. A partir dos dados de mutação, um escore de suspeita $S(e)$ é calculado para cada mutante de um comando e o maior valor encontrado corresponde à suspeita do próprio comando. Assim como em SBFL, após calcular-se o $S(e)$, uma lista é organizada de forma descendente para guiar a investigação à localização do defeito.

O princípio fundamental das heurísticas MBFL é atribuir suspeitas aos mutantes gerados para cada elemento de código, ele se baseia na suposição de que os casos de teste que matam mutantes tem poder de diagnóstico, ou seja, quanto mais um comando afeta testes negativos e quanto menos o mesmo afeta testes positivos, maior será a suspeita do comando.

A Figura 2.2 exemplifica o processo de cálculo de suspeita $S(e)$ de um elemento e, destacado em vermelho. Para cada elemento, são gerados n mutantes. A suspeita atribuída ao elemento será a maior suspeita calculada, para os mutantes do mesmo, por uma heurística MBFL.

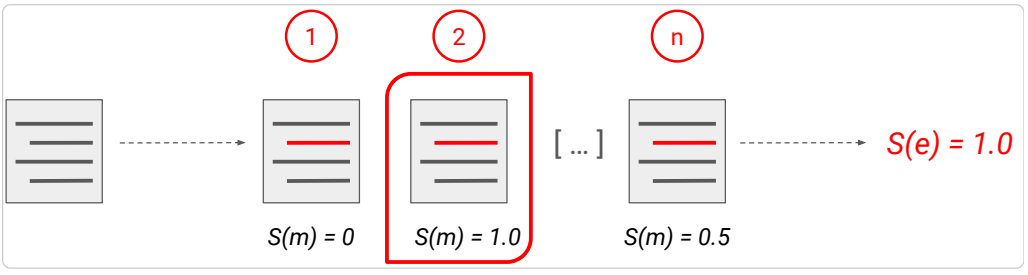

Figura 2.2: *Exemplo de aplicação de heurística MBFL ao cálculo de suspeita de elementos de código.*

Papadakis *et al.* [42] apresentaram uma abordagem em que as suspeitas são calculadas adaptando-se heurísticas SBFL conhecidas da literatura, como Tarantula ou Ochiai ao tratar mutantes mortos como comandos cobertos e mutantes vivos como

comandos não executados. Assim, as formulações matemáticas das heurísticas MBFL adaptam fórmulas SBFL com as seguintes notações: m_{kf} é o número de casos de teste negativos que mataram o mutante (análogo a c_{ef}), m_{kp} é o número de casos de teste positivos que mataram o mutante (análogo a c_{ep}), m_{nf} é o número de casos de teste negativos que não mataram o mutante (análogo a c_{nf}) e m_{np} é o número de casos de teste positivos que não mataram o mutante (análogo a c_{np}). As Fórmulas 2-3 e 2-4 correspondem às versões MBFL das heurísticas Tarantula e Ochiai respectivamente.

$$S(e) = \frac{\frac{m_{kf}}{m_{kf}+m_{nf}}}{\frac{m_{kf}}{m_{kf}+m_{nf}} + \frac{m_{kp}}{m_{kp}+m_{np}}} \tag{2-3}$$

$$S(e) = \frac{m_{ef}}{\sqrt{(m_{ef}+m_{nf}) \times (m_{ef}+m_{es})}} \tag{2-4}$$

2.5 Engenharia de Software Baseada em Busca

Problemas de otimização estão presentes em diversas áreas da ciência, como medicina, engenharia, economia, logística, entre outras. Soluções para estes problemas precisam ser encontradas e diversos métodos de otimização capazes de encontrar soluções ótimas ou sub-ótimas estão disponíveis [46].

Formalmente, segundo a definição posta em [47], um problema de otimização pode ser definido pelo par (S, f), tal que:

- S representa o conjunto de soluções factíveis, que estão presentes no *espaço de busca*, onde há as soluções possíveis para o problema; e
- $f : S \to \mathbb{R}$, denominada *função objetivo*.

A função objetivo atribui um número real a cada solução factível no espaço de busca, de modo que seja possível comparar soluções e tomar decisões para a escolha daquela que é a melhor para a resolução do problema.

Os métodos de otimização podem ser divididos em duas classes de algorítimos [47]: Métodos Exatos (ou Algoritmos Completos) e Métodos Aproximados. **Métodos Exatos** obtêm soluções garantidamente ótimas, entretanto, são algoritmos de tempo não polinomial para problemas NP-completos. **Métodos Aproximados** geram soluções de boa qualidade em tempo razoável, mas não garantem soluções ótimas globais (a melhor solução possível).

A categoria dos Métodos Aproximados pode ser subdividida entre Algoritmos de Aproximação e Algoritmos Heurísticos (Heurísticas) [47]. Os **Algoritmos de Aproximação** rendem soluções de qualidade demonstrável em limites de tempo de execução demonstráveis, contrapondo-se às **Heurísticas**, que são geralmente capazes de encontrar

soluções boas em tempo razoável. As Heurísticas permitem obter desempenho aceitável com custo aceitável para ampla variedade de problemas, mas não permitem garantias de aproximação da solução encontrada.

As Heurísticas podem ser classificadas em duas famílias: Heurísticas Específicas e Meta-heurísticas [47]. **Heurísticas específicas** são projetadas "sob medida" para resolver problemas e/ou instâncias específicas. **Meta-heurísticas**, por sua vez, são algoritmos de propósito geral que podem ser aplicadas à resolução de quase todos os problema de otimização. Segundo Talbi [47], elas podem ser vistas como métodos generalistas de alto nível que podem ser utilizadas como uma estratégia para guiar o projeto de uma heurística apropriada à resolução de problemas de otimização específicos.

A Engenharia de Software é uma das áreas que contém problemas de otimização aos quais as Meta-heurísticas podem ser aplicadas. Na visão de Harman e Jones [18], a Engenharia de Software se assemelha às outras engenharias ao se buscar soluções quase ótimas ou que estejam dentro de um limite de tolerância aceitável e estes são os principais fatores que permitem a aplicação direta de técnicas de otimização baseadas em busca, especialmente as meta-heurísticas.

Assim, Harman e Jones apresentaram três itens que devem ser definidos para que um problema da Engenharia de Software seja resolvido como um problema de busca [18]:

- Uma representação para as soluções do problema que permita manipulação modular em sua estrutura;
- Uma função de aptidão capaz de avalizar as soluções conforme sua qualidade, considerando a representação definida;
- Um conjunto de operadores de manipulação.

Engenharia de Software Baseada em Busca (*SBSE*, do Inglês *Search-based Software Engineering*) é o nome dado ao campo de pesquisa que aplica técnicas de otimização baseadas em busca à Engenharia de Software [19]. Segundo Harman e Jones [18], é possível aplicar meta-heurísticas a diversos problemas da Engenharia de Software em que as representações de problema, funções objetivo e operadores são naturais.

Dentre as características de SBSE que motivam sua aplicação, estão: generalidade, robustez, escalabilidade por paralelismo, reunificação e cálculo direto de aptidão [19].

- **Generalidade**: SBSE é amplamente aplicável e pode ser empregado com sucesso partindo-se apenas duas definições: uma **representação do problema** e uma **função de avaliação** que captura o objetivo a ser otimizado.
- **Robustez**: Os resultados obtidos pelos algoritmos de otimização de SBSE costumam ser robustos à escolha de parâmetros, isto é, escolhas de parâmetros razoáveis para os algoritmos superam confortavelmente busca aleatória.

- **Escalabilidade por paralelismo**: Meta-heurísticas executam a função de avaliação de forma paralela, o que indica possibilidade de escalabilidade. Muitos autores demonstraram que este paralelismo pode ser explorado com computação distribuída.
- **Reunificação**: Sob o ponto de vista de SBSE, subáreas distintas da Engenharia de Software podem ser aproximadas. Por exemplo, Engenharia de Requisitos e Teste de Regressão, que são áreas que não costumam se relacionar, podem ser tratadas como problemas de otimização de estrutura similar.
- **Cálculo direto de aptidão**: Ao contrário dos problemas de engenharia tradicionais, em Engenharia de Software, meta-heurísticas podem ser aplicadas diretamente ao problema, sem a necessidade de construções de modelos ou simulações.

2.6 Meta-heurísticas

Luke [33] define *Meta-heurísticas* como uma palavra *infeliz* para descrever a principal subárea em otimização estocástica. Luke coloca as Meta-heurísticas como o algorítimo de Otimização Estocástica mais geral, que é a classe dos algoritmos e técnicas de otimização que utilizam algum grau de aleatoriedade na busca pela melhor solução possível de um problema.

Segundo Talbi [47], a palavra heurística tem sua origem na palavra grega *heuriskein*, que significa "*a arte da descoberta de novas estratégias (regras) para a resolução de problemas*". O sufixo meta, também de origem grega, significa "*metodologia de nível superior*". O termo meta-heurística foi utilizado pela primeira vez em 1986 por Glover [15]. Outro termo para Meta-heurísticas é **Heurísticas Modernas**, definidas por Rothlauf como Heurísticas de Melhoria que utilizam estratégias de busca gerais [46].

Duas fases importantes do processo de busca de Meta-heurísticas são intensificação e diversificação[46, 47]. **Diversificação** refere-se à fase de ampla exploração do espaço de busca, ou seja, novas áreas são exploradas de modo a garantir que regiões diversas sejam igualitariamente visitadas e que a busca não se concentre em poucas regiões. **Intensificação** refere-se à exploração pontual de regiões promissoras, isto é, a busca concentra-se em regiões onde boas soluções foram encontradas com o intuito de encontrar melhores soluções.

A utilização de meta-heurísticas não é recomendada para problemas em que métodos exatos podem ser utilizados em tempo hábil, já que as meta-heurísticas não garantem encontrar a solução ótima global, a melhor solução possível.

2.6.1 Algoritmos Evolucionários

Dentre as diversas classes de meta-heurísticas, os algoritmos evolucionários têm inspiração na síntese evolutiva moderna, que combina principalmente os conceitos da seleção natural das espécies de Charles Darwin às descobertas de Gregor Mendel, acerca de hereditariedade biológica e genética [13]. Em síntese, a seleção natural ocorre na natureza quando os indivíduos mais adaptados ao meio têm melhores chances de sobrevivência e, consequentemente, melhores chances de se reproduzirem. Ao se reproduzirem, os indivíduos mais aptos geram descendentes pela recombinação genética e eventuais mutações, alterações aleatórias de material genético.

Em populações isoladas, indivíduos que nascem com novas características que permitem melhores condições de competição por recursos e de sobrevivência, são capazes de se reproduzirem mais vezes e, portanto, passar estas características aos seus descendentes. Assim, com o passar das gerações, a população torna-se especializada às características do meio na qual ela está. Em geral, este comportamento competitivo é emulado pelo funcionamento dos algoritmos evolucionários.

Em algoritmos evolucionários, alguns termos da biologia são utilizados para denotar elementos de otimização, de modo que gerações são iterações, indivíduos são soluções, população é o conjunto de soluções de cada iteração e função de aptidão (ou *fitness*) é a função objetivo do algoritmo. A Figura 2.3, extraída de [47], representa o esquema iterativo que define um algoritmo evolucionário.

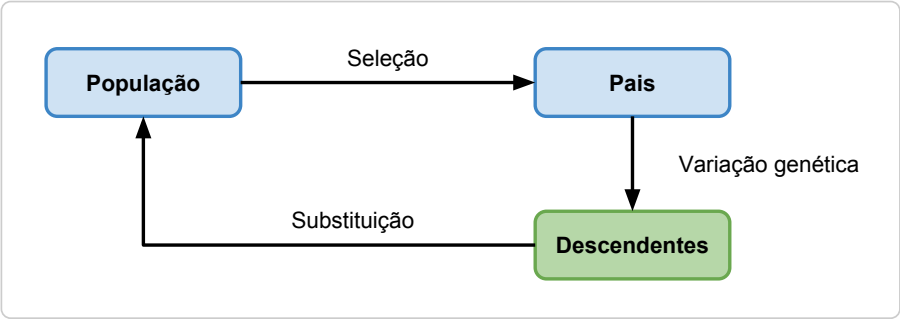

Figura 2.3: *Esquema iterativo dos algoritmos evolucionários.*

A cada geração, a população passa por uma operação de seleção dos indivíduos que se tornarão pais. Esta operação seleciona os indivíduos para as operações de geração de descendentes, utilizando os valores calculados pela função de aptidão, de modo que esta operação é responsável pela pressão seletiva do algoritmo.

Com o conjunto de pais selecionado, novos indivíduos, os descendentes, são gerados por operações de variação genética, como reprodução, cruzamento ou mutação. Posteriormente, regras de substituição de pais por filhos são aplicadas para que uma nova população seja constituída e uma nova geração se inicie.

Algoritmo 2.1: Algoritmo Evolucionario [47]

Entrada: Problema.

Saída: Melhor solução encontrada.

```
1  Gerar(P(0)); /*População Inicial*/
2  t = 0;
3  enquanto não Criterio_de_Parada(P(t)) faça
4      Avaliar(P(t));
5      P'(t) = Selecao(P(t));
6      P'(t) = Variacao_Genetica(P'(t));
7      Avaliar(P'(t));
8      P(t+1) = Subistituir(P(t),P'(t));
9      t = t+1;
10 fim
```

O Algoritmo 2.1, extraído de [47], representa um modelo para um algoritmo evolucionário. Dado o problema, uma população inicial é gerada, comumente de forma aleatória, e avaliada conforme a função de aptidão. Posteriormente, o ciclo apresentado na Figura 2.1 é repetido até que o critério de parada seja atingido. Este critério pode ser um limite para o número de gerações decorridas ou um valor mínimo de aptidão encontrada em uma solução. Após o término da execução, o melhor indivíduo encontrado é retornado como a solução.

Outro aspecto fundamental para os algoritmos evolucionários é a representação genotípica e fenotípica dos indivíduos [47]. **Genótipo** é a codificação da solução, geralmente sob a forma de uma estrutura de dados como um vetor ou uma árvore. Todas as operações de variação genética ocorrem sob o genótipo do indivíduo, são as operações de manipulação de material genético. **Fenótipo** corresponde ao que a solução representa, ou seja é a decodificação do genótipo. Tipicamente, a função de aptidão é aplicada ao fenótipo de um indivíduo.

Algoritmo Genético

O Algoritmo Genético surgiu na década de 70, proposto por Holland [20] e pertence à classe dos algoritmos evolucionários. Nele, cada iteração inicia-se com a seleção de dois pais da população da geração anterior, estes que por sua vez são cruzados para gerar descendentes, que sofrem mutação [33]. Assim, o algoritmo genético destaca-se pela utilização de operações de cruzamento seguidas de mutação na etapa de variação genética.

Originalmente, o algoritmo genético utiliza um método de seleção por roleta, em que um sorteio é realizado e os indivíduos têm probabilidade de serem selecionados

proporcional à sua aptidão. No algoritmo genético canônico, a população de pais é substituída integralmente pela população de filhos. Ainda, as representações genotípicas mais comuns são binárias, isto é, o genótipo do indivíduo é representado por uma cadeia de bits [47].

Diversos operadores de cruzamento e mutação estão disponíveis. O cruzamento dá-se pela troca do material genético de dois indivíduos pais para gerar dois descendentes novos. A mutação é uma operação que troca um bit no genótipo do indivíduo, escolhido aleatoriamente. A mutação e o cruzamento ocorrem conforme uma taxa de probabilidade previamente definida.

Programação Genética

O algoritmo de Programação Genética foi proposto por Koza [29] ao estender o algoritmo genético para a evolução de programas na linguagem LISP. A principal inovação, dentre os algoritmos evolucionários, está no genótipo dos indivíduos, baseado em árvores, para a representação de programas (fenótipo). Assim, a representação é não linear e contrapõe-se aos algoritmos prévios, como algoritmo genético, que aplica cadeias de tamanho fixo em uma representação linear.

Assim, a programação genética representa uma forma de indução para permitir a geração (síntese) de programas para resolver uma dada tarefa [47]. Os indivíduos são representados como árvores, em que os nós folha são terminais (constantes ou variáveis) e os nós internos são funções (operadores).

Assim como no algoritmo genético, a programação genética comumente emprega a seleção de pais proporcional à aptidão e substituição da população de pais pelos descendentes gerados. Uma inovação dá-se nas operações de mutação e cruzamento. A mutação ocorre, por exemplo, ao substituir um nó por um novo nó ou subárvore gerada aleatoriamente, enquanto o cruzamento ocorre pela troca de subárvores entre os indivíduos pais. Ainda, há um operador de reprodução, em que os pais são copiados para a nova população, sem qualquer alteração.

Além da criação de programas, o algoritmo é utilizado em problemas de regressão simbólica. Neste caso, as árvores são uma representação de funções matemáticas, com o conjunto de funções constituído por operadores matemáticos, e o conjunto de terminais composto por variáveis ou constantes numéricas. Neste caso, a função de avaliação pode ser o erro em relação à função que se deseja aproximar. Segundo Talbi [47], a programação genética tem sido amplamente utilizada em aplicações de mineração de dados e aprendizado de máquina, como predição, classificação e regressão.

2.7 Localização de Defeitos Baseada em Busca

Diversos problemas de localização de defeitos foram abordados pela literatura ao longo dos anos, tais como geração e seleção de casos de teste e geração e combinação de heurísticas de localização de defeitos. Em geral, trabalhos que tratam de seleção e geração de casos de teste para localização de defeitos objetivam dados de espectro de cobertura mais completos e que consequentemente a precisão das heurísticas SBFL seja potencializado [44] [7] [21] [22] ou a redução do custo do teste sem perda da capacidade de localização de defeitos pelo conjunto de teste [14].

Parsa *et al.* [44] partem da premissa de que ao comparar caminhos de execução similares de testes bem-sucedidos e malsucedidos, os comandos diferentes nas duas execuções são possivelmente defeituosos. Assim, Algoritmo Genético foi aplicado para buscar os testes positivos e negativos com os caminhos de execução mais semelhante possíveis.

Campos *et al.* [7] apresentaram uma técnica baseada em Algoritmo Genético para a busca por um conjunto de casos de teste que ofereça a melhor capacidade de diagnóstico para heurísticas SBFL. Esta capacidade é medida pela quantidade de elementos que podem ser diferenciados ao observar a matriz de cobertura do conjunto de teste. Segundo os autores, a abordagem foi capaz de reduzir em média 91% dos elementos necessários a investigar no ranking de suspeitas calculadas por heurísticas SBFL até encontrar o defeito.

Outros trabalhos como Huang *et al.* [21] e Hui [22] utilizaram Algoritmo Colônia de Abelhas e Algoritmo Genético respectivamente para buscar por casos de teste que propiciam melhores resultados às heurísticas SBFL.

Geng *et al.* [14] propuseram uma abordagem para combinar detecção e localização de defeitos ao gerar casos de teste visando a ambos os propósitos. O trabalho é descrito pelos autores como uma abordagem de minimização multiobjetivo de conjuntos de teste, ao buscar ao mesmo tempo reduzir a quantidade de testes e selecionar o conjunto que otimiza simultaneamente as capacidades de localização e detecção de defeitos. Foi utilizada o algoritmo multiobjetivo NSGA-II (*Non-dominated Sorting Genetic Algorithm*), para balancear os objetivos distintos da busca, que são:

- **Objetivo 1**: minimizar o conjunto de teste;
- **Objetivo 2**: maximizar a taxa de cobertura de código dos testes;
- **Objetivo 3**: maximizar a capacidade de diagnóstico para heurísticas SBFL, semelhante a [7].

Estes são exemplos de trabalhos de localização de defeitos baseada em busca que abrangem a seleção e geração de dados de cobertura, ou testes, para alimentar as heurísticas SBFL padrão. Outras iniciativas trabalham a utilização de ferramentas de

busca aplicadas à confecção automática de heurísticas SBFL. Assim, a solução não representa um conjunto de teste, mas sim uma heurística SBFL.

Wang *et al.* [51] propuseram um modelo baseado em busca para encontrar a combinação linear de 22 heurísticas SBFL conhecidas. Esta abordagem foi intitulada pelos autores "*localização de defeitos baseada em busca*", de modo que a combinação de heurísticas foi tratada como um problema de otimização. A estratégia adotada consiste na busca pelos pesos w_n associados a cada heurística w_n na Equação 2-5. Assim, o método consiste em uma soma linear das 22 heurísticas SBFL ponderadas por pesos, de modo que as soluções possíveis são restritas às combinações lineares das heurísticas.

$$H_C(e) = w_1 \times H_1(e) + w_2 \times H_2(e) + \cdots + w_{22} \times H_{22}(e) \tag{2-5}$$

As soluções de Wang *et al.* [51] são representadas por uma cadeia com sete bits para cada peso, de modo que cada peso tem valor decimal mínimo zero e máximo um. Ademais, cada solução é avaliada conforme o ranking de suspeitas calculadas pela heurística combinada correspondente, com função de avaliação definida como a proporção de elementos de código investigados até a localização de todos os defeitos.

A proposta empregou Algoritmo Genético para a busca pelos pesos que melhor combinam as heurísticas SBFL Tarantula, Ochiai e as 20 métricas de associação de Lucia *et al.* [32]. Assim, a heurística combinada $M_C(e)$ (Equação 2-5) é uma combinação linear de heurísticas SBFL prévias. O texto também não é claro quanto à especialização de projetos em seus experimentos.

A metodologia estrutura-se em duas fases: treinamento e implantação. Na primeira, o motor de busca treina uma nova heurística para um conjunto de versões defeituosas de um mesmo programa. Na seguinte, a heurística resultante é aplicada a um conjunto diferente de versões do mesmo programa, ou seja, os conjuntos de versões usados para treinamento e implantação são disjuntos. Conforme afirmado por Wang *et al.* [51], uma boa heurística deve ser capaz de localizar muitos defeitos no conjunto de treinamento com uma alta precisão [51]. Em um cenário do mundo real, a fase de treinamento é feita para um conjunto de versões anteriores com defeitos conhecidas e previamente catalogados, enquanto a fase de implantação consiste em aplicar a heurística treinada na depuração das versões mais recentes do programa.

Outro trabalho importante acerca de construção automática de heurísticas SBFL é o de Yoo [55], que introduziu a aplicação do algoritmo de Programação Genética à construção de heurísticas através da combinação de operadores matemáticos simples e as principais variáveis presentes no espectro de cobertura. As heurísticas foram construídas de forma genérica, isto é, não customizadas a projetos específicos, de modo que a fase de treinamento não foi conduzida utilizando-se versões defeituosas do mesmo programa.

A abordagem de Yoo [55] utiliza Programação Genética para sintetizar heurísti-

cas SBFL a partir das quatro variáveis básicas de cobertura de código $\{c_{ef}, c_{ep}, c_{nf}$ e $c_{np}\}$ e a constante inteira 1 (um) e o conjunto de funções matemáticas composto pelas quatro operações básicas e raiz quadrada. O objetivo é a minimização da média da métrica *Expense* (a posição do comando defeituoso em relação ao número de comandos), em relação a n versões defeituosas, conforme representado na Equação 2-6. A aptidão do indivíduo é calculada a partir do ranking de suspeitas obtido por sua heurística correspondente.

$$Expense\ médio = \frac{\sum_{i=1}^{n} expense(i)}{n} \tag{2-6}$$

Yoo [55] apresentou 30 fórmulas obtidas por 30 execuções do algoritmo de Programação Genética, das quais 8 obtiveram eficácia equivalente ou superior às fórmulas criadas por humanos.

No contexto de aplicação de técnicas evolucionárias à localização de defeitos, o último estudo acerca de sua análise teórica e empírica apontou os seguintes trabalhos futuros [56]:

- *construir fórmulas que sejam eficazes em certos contextos*, de modo que as heurísticas sejam geradas para projetos específicos (heurísticas treinadas para contextos de programas específicos) e não para conjuntos de projetos (heurísticas genéricas).
- *utilizar sinais além de espectro de programa*, isto é, utilizar outras fontes de dados diferentes dos espectros de cobertura, tais como o espectro de mutação.

Um desenvolvimento importante do trabalho de Yoo [55] foi posteriormente apresentado por Xie *et al.* [54]. Trata-se de uma avaliação teórica e empírica das heurísticas treinadas por Yoo [55] no âmbito de programas com defeitos únicos. Dentre suas principais conclusões, a publicação afirmou que a Programação Genética é uma ferramenta adequada para se construir heurísticas de localização de defeitos que são competitivas em relação às melhores heurísticas desenvolvidas por humanos.

2.8 Considerações Finais

Localização de defeitos é uma tarefa essencial ao desenvolvimento de software, visto que sua demanda cresce com o aumento da complexidade dos projetos de software. Assim, o alto custo e o tempo despendidos com localização de defeitos estimula o desenvolvimento de técnicas que auxiliam e otimizam o processo. Algumas destas técnicas foram descritas neste capítulo, dentre elas as heurísticas de localização de defeitos baseadas em espectro são as mais populares na literatura, com poucas iniciativas no âmbito da Engenharia de Software Baseada em Busca.

O presente trabalho se propõe a contribuir para os trabalhos de localização de defeitos baseada em busca sob dois aspectos: composição de heurísticas existentes com espaço de busca ampliado, construção de novas heurísticas expandindo a utilização de espectro de cobertura pelo emprego de variáveis do espectro de mutação e treinamento de heurísticas especializadas (treinadas) para projetos únicos. Assim, os trabalhos futuros elencados por Yoo [56] foram explorados.

CAPÍTULO 3

Composição de Heurísticas Baseadas em Espectro de Cobertura

O objetivo das técnicas SBFL é descobrir quais são os elementos responsáveis por defeitos com base nos espectros de programa obtidos para execuções bem-sucedidas, oriundas de testes positivos, e malsucedidas, obtidas de testes negativos [32], ou seja, relacionar elementos de código à ocorrência do defeito. Elementos de software referem-se a uma granularidade específica, como linha de código ou bloco de instruções. Para tal, são utilizadas heurísticas para calcular a propensão a defeitos de um dado elemento. Esta propensão representa a força da associação entre as execuções de um elemento e as ocorrências do defeito. Uma lista pode ser organizada em ordem decrescente de suspeita para que o desenvolvedor possa analisar os elementos, de maior para menor suspeita, até que todos os defeitos sejam localizados.

Conforme apresentado no Capítulo 2, diversas heurísticas foram propostas com base em variáveis de cobertura como: c_{ep} (número de execuções bem-sucedidas de um elemento) e c_{ef} (número de execuções malsucedidas de um determinado elemento). No entanto, as heurísticas SBFL tradicionais aplicadas individualmente tem desempenho semelhante e não são suficientes. Além disso, não há uma única heurística que seja boa em todas as situações. Então, seguiu-se o raciocínio de que a composição de diferentes heurísticas poderia ser melhor que apenas uma.

O método apresentado por Wang *et al.* [51] combina linearmente heurísticas de localização de defeitos. Sua proposta é encontrar combinações personalizadas para heurísticas conhecidas da literatura. Este capítulo apresenta um método para adicionar à capacidade de combinação de heurísticas SBFL do método apresentado por Wang *et al.* [51] permitindo outras combinações matemáticas com um grupo selecionado de funções e o algoritmo de Programação Genética. O aspecto inovador desta proposta caracteriza-se pela conciliação das seguintes premissas em uma abordagem única:

- equações não lineares de heurísticas SBFL;
- heurísticas SBFL combinadas customizadas para projetos específicos;

- fórmulas compostas por outras heurísticas de localização de defeitos e variáveis básicas de espectro de cobertura.

A abordagem apresentada culminou em publicações no 8^o Workshop de Engenharia de Software Baseada em Busca (*WESB 2017*) [8] e no *26th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning* (*ESANN 2018*) [9].

A proposta enquadra-se no contexto de combinações de heurísticas, assim como a abordagem de Wang *et al.* [51], e divide-se em duas partes. A primeira investiga a composição de 2 heurísticas SBFL e 16 métricas de associação de Lucia *et al.* [32]. A segunda parte estende a pesquisa ao combinar 34 heurísticas SBFL que estão dentre as mais conhecidas e utilizadas na literatura [53], sendo este um conjunto diferente do apresentado por Wang *et al.* [51]. Cada um dos segmentos da pesquisa foi explorado em um artigo científico, publicados no *WESB 2017* [8] e no *ESANN 2018* [9].

O restante deste capítulo está estruturado da seguinte maneira: A Seção 3.1 descreve o método proposto. A Seção 3.2 apresenta o projeto de experimentação, os parâmetros e as métricas de avaliação utilizadas. A Seção 3.3 apresenta e analisa os resultados; e a Seção 3.4 conclui o capítulo.

3.1 Método Proposto: Aplicação de Programação Genética para Combinar Heurísticas SBFL

A representação adotada para o problema de combinação de heurísticas SBFL se caracteriza pela organização de um indivíduo que representa uma combinação de diferentes heurísticas em estrutura de árvore. Assim, o espaço de busca compreende o conjunto de todas as fórmulas possíveis de serem obtidas pelas funções e terminais disponibilizados. Observa-se que fórmulas não lineares são comumente encontradas nesse processo. Em resumo, a proposta deste trabalho se constrói a partir da premissa de que a Programação Genética pode ser aplicada ao processo de busca pelas melhores heurísticas SBFL especializadas para um projeto único.

O **conjunto de funções** permitidas para as soluções de Programação Genética foi definido como: as quatro operações matemáticas básicas (adição, subtração, multiplicação e divisão), raiz quadrada, logaritmo e função máxima. Estas são as funções observadas internamente nas heurísticas SBFL utilizadas como terminais. Dois **conjuntos de terminais** foram utilizados em duas configurações do método proposto:

- **Conjunto T1**: O conjunto das heurísticas SBFL utilizadas por Wang *et al.* [51], composto das heurísticas estudadas por Lucia *et al.* [32] (apresentadas na Tabela 3.1) além de Tarantula e Ochiai (H_{32} e H_{33} na Tabela 3.2 respectivamente);

- **Conjunto T2**: Um conjunto de heurísticas SBFL relevantes e mais comumente citadas da literatura (Tabela 3.2), listado por Wong *et al.* [53] em uma publicação recente, além das quatro variáveis do espectro de cobertura utilizadas nas próprias heurísticas (c_{ef}, c_{ep}, c_{nf} e c_{np}).

Table 3.1: *Métricas de associação utilizadas pelos algoritmos de composição*

ID	Nome	Fórmula
AM_1	ϕ-Coefficient	$\frac{P(E,F)-P(E)\times P(F)}{\sqrt{P(E)\times P(F)\times(1-P(E))\times(1-P(F))}}$
AM_2	Yule's Q	$\frac{P(E,F)\times P(\overline{E},\overline{F})-P(E,\overline{F})\times P(\overline{E},F)}{P(E,F)\times P(\overline{E},\overline{F})+P(E,\overline{F})\times P(\overline{E},F)}$
AM_3	Yule's Y	$\frac{\sqrt{P(E,F)\times P(\overline{E},\overline{F})}-\sqrt{P(E,\overline{F})\times P(\overline{E},F)}}{\sqrt{P(E,F)\times P(\overline{E},\overline{F})}+\sqrt{P(E,\overline{F})\times P(\overline{E},F)}}$
AM_4	Kappa	$\frac{P(E,F)+P(\overline{E},\overline{F})-P(E)\times P(F)-P(\overline{E})\times P(\overline{F})}{1-P(E)\times P(F)-P(\overline{E})\times P(\overline{F})}$
AM_5	J-Measure	$max\left(P(E,F)\times\log\left(P(F\|E)/P(F)\right)+P(E,\overline{F})\times\log\left(P(\overline{F}\|E)/P(\overline{F})\right),\right.$ $\left.P(E,F)\times\log\left(P(E\|F)/P(E)\right)+P(\overline{E},F)\times\log\left(P(\overline{E}\|F)/P(\overline{E})\right)\right)$
AM_6	Gini Index	$max(P(E)\times[P(F\|E)^2+P(\overline{F}\|E)^2]+P(\overline{E})\times[P(F\|\overline{E})^2+P(\overline{F}\|\overline{E})^2]-P(F)^2-P(\overline{F})^2,$ $P(F)\times[P(E\|F)^2+P(\overline{E}\|F)^2]+P(\overline{F})\times[P(E\|\overline{F})^2+P(\overline{E}\|\overline{F})^2]-P(E)^2-P(\overline{E})^2)$
AM_7	Support	$P(E\|F)$
AM_8	Confidence	$max(P(F\|E),P(E\|F))$
AM_9	Laplace	$max\left(\frac{P(E,F)+1}{P(E)+2},\frac{P(E,F)+1}{P(F)+2}\right)$
AM_{10}	Cosine	$\frac{P(E,F)}{\sqrt{P(E)\times P(F)}}$
AM_{11}	Piatetsky-Shapiro's	$P(E,F)-P(E)\times P(F)$
AM_{12}	Certainty Factor	$max\left(\frac{P(F\|E)-P(F)}{1-P(F)},\frac{P(E\|F)-P(E)}{1-P(E)}\right)$
AM_{13}	Added Value	$max(P(F\|E)-P(F),P(E\|F)-P(E))$
AM_{14}	Jaccard	$\frac{P(E,F)}{P(E)+P(F)-P(E,F)}$
AM_{15}	Klosgen	$\sqrt{P(E,F)}\times max(P(F\|E)-P(F),P(E\|F)-P(E))$
AM_{16}	Information Gain	$(-P(F)\times\log P(F)-P(\overline{F})\times\log P(\overline{F}))-$ $(P(E)\times(-P(F\|E)\times\log P(F\|E))-P(\overline{F}\|E)\times\log P(\overline{F}\|E)-$ $P(\overline{E})\times(-P(F\|\overline{E})\times\log P(F\|\overline{E}))-P(\overline{F}\|\overline{E})\times\log P(\overline{F}\|\overline{E}))$

A função de avaliação, empregada durante a execução do algoritmo de Programação Genética, é calculada a partir dos rankings de suspeitas obtidos pela solução para cada um dos programas do conjunto de treinamento. Assim como Wang *et al.* [51], a aptidão de cada indivíduo é aferida utilizando-se a média da métrica *expense*, que retorna a posição relativa do comando defeituoso no ranking de suspeita em relação a todas as versões defeituosas no conjunto de treinamento. A Subseção 3.2.4 detalha a métrica *expense* médio. A Equação 3-1 representa a função de avaliação calculada para uma solução s e

Tabela 3.2: *Heurísticas SBFL utilizadas pelos algoritmos de composição*

ID	Nome	Fórmula
H_1	Braun-Banquet	$\frac{n_f(e)}{max(n_f(e)+n_s(e),n_f(e)+n_f(\overline{e}))}$
H_2	Dennis	$\frac{(n_f(e)\times n_s(\overline{e}))-(n_s(e)\times n_f(\overline{e}))}{\sqrt{n\times(n_f(e)+n_s(e))\times(n_f(e)+n_f(\overline{e}))}}$
H_3	Mountford	$\frac{n_f(e)}{0{,}5\times((n_f(e)\times n_s(e))+(n_f(e)\times n_f(\overline{e})))+(n_s(e)\times n_f(\overline{e}))}$
H_4	Fossum	$\frac{n\times(n_f(e)-0{,}5)^2}{(n_f(e)+n_s(e))\times(n_f(e)+n_f(\overline{e}))}$
H_5	Pearson	$\frac{n\times((n_f(e)\times n_s(\overline{e}))-(n_s(e)\times n_f(\overline{e})))^2}{n(e)\times n(\overline{e})\times n_s\times n_f}$
H_6	Gower	$\frac{n_f(e)+n_s(\overline{e})}{\sqrt{n_f\times n(e)\times n(\overline{e})\times n_s}}$
H_7	Michael	$\frac{4\times((n_f(e)\times n_s(\overline{e}))-(n_s(e)\times n_f(\overline{e})))}{(n_f(e)+n_s(\overline{e}))^2+(n_s(e)+n_f(\overline{e}))^2}$
H_8	Pirce	$\frac{(n_f(e)\times n_f(\overline{e}))+(n_f(\overline{e})\times n_s(e))}{(n_f(e)\times n_f(\overline{e}))+(2\times(n_f(\overline{e})\times n_s(\overline{e})))+(n_s(e)\times n_s(\overline{e}))}$
H_9	Baroni-Urbani & Buser	$\frac{\sqrt{n_f(e)\times n_s(\overline{e})}+n_f(e)}{\sqrt{n_f(e)+n_s(\overline{e})}+n_f(e)+n_s(e)+n_f(\overline{e})}$
H_{10}	Tarwid	$\frac{(n\times n_f(e))-(n_f\times n(e))}{(n\times n_f(e))+(n_f\times n(e))}$
H_{11}	Ample	$\left\lvert\frac{n_f(e)}{n_f(e)+n_f(\overline{e})}-\frac{n_s(e)}{n_s(e)+n_s(\overline{e})}\right\rvert$
H_{12}	Phi (Geometric Mean)	$\frac{n_f(e)\times n_s(\overline{e})-n_f(\overline{e})\times n_s(e)}{\sqrt{(n_f(e)+n_s(e))\times(n_f(e)+n_f(\overline{e}))\times(n_s(e)+n_s(\overline{e}))\times(n_f(\overline{e})+n_s(\overline{e}))}}$
H_{13}	Arithmetic Mean	$\frac{2\times(n_f(e)\times n_s(\overline{e})-n_f(\overline{e})\times n_s(e))}{(n_f(e)+n_s(e))\times(n_s(\overline{e})+n_f(\overline{e}))+(n_f(e)+n_f(\overline{e}))\times(n_s(e)+n_s(\overline{e}))}$
H_{14}	Cohen	$\frac{2\times(n_f(e)\times n_s(\overline{e})-n_f(\overline{e})\times n_s(e))}{(n_f(e)+n_s(e))\times(n_s(\overline{e})+n_s(e))+(n_f(e)+n_f(\overline{e}))\times(n_f(\overline{e})+n_s(\overline{e}))}$
H_{15}	Fleiss	$\frac{4\times(n_f(e)\times n_s(\overline{e})-n_f(\overline{e})\times n_s(e))-(n_f(\overline{e})-n_s(e))^2}{(2\times n_f(e)+n_f(\overline{e})+n_s(e))+(2\times n_s(\overline{e})+n_f(\overline{e})+n_s(e))}$
H_{16}	Zoltar	$\frac{n_f(e)}{n_f(e)+n_f(\overline{e})+n_s(e)\frac{10000\times n_f(\overline{e})\times n_s(e)}{n_f(e)}}$
H_{17}	Harmonic Mean	$\frac{(n_f(e)\times n_s(\overline{e})-n_f(\overline{e})\times n_s(e))\times((n_f(e)+n_s(e))\times(n_s(\overline{e})+n_f(\overline{e}))+(n_f(e)+n_f(\overline{e}))\times(n_s(e)+n_s(\overline{e})))}{(n_f(e)+n_s(e))\times(n_s(\overline{e})+n_f(\overline{e}))\times(n_f(e)+n_f(\overline{e}))\times(n_s(e)+n_s(\overline{e}))}$
H_{18}	Rogot2	$\frac{1}{4}\left(\frac{n_f(e)}{n_f(e)+n_s(e)}+\frac{n_f(e)}{n_f(e)+n_f(\overline{e})}+\frac{n_s(\overline{e})}{n_s(\overline{e})+n_s(e)}+\frac{n_s(\overline{e})}{n_s(\overline{e})+n_f(\overline{e})}\right)$
H_{19}	Simple Matching	$\frac{n_f(e)+n_s(\overline{e})}{n_f(e)+n_s(\overline{e})+n_s(\overline{e})+n_f(\overline{e})}$
H_{20}	Rogers & Tanimoto	$\frac{n_f(e)+n_s(\overline{e})}{n_f(e)+n_s(\overline{e})+2\times(n_f(\overline{e})+n_s(e))}$
H_{21}	Hamming	$n_f(e)+n_s(\overline{e})$
H_{22}	Hamann	$\frac{n_f(e)+n_s(\overline{e})-n_f(\overline{e})-n_s(e)}{n_f(e)+n_f(\overline{e})+n_s(e)+n_s(\overline{e})}$
H_{23}	Sokal	$\frac{2\times(n_f(e)+n_s(\overline{e}))}{2\times(n_f(e)+n_s(\overline{e}))+n_f(\overline{e})+n_s(e)}$
H_{24}	Scott	$\frac{4\times(n_f(e)\times n_s(\overline{e})-n_f(\overline{e})\times n_s(e))-(n_f(\overline{e})-n_s(e))^2}{(2\times n_f(e)+n_f(\overline{e})+n_s(e))\times(2\times n_s(\overline{e})+n_f(\overline{e})+n_s(e))}$
H_{25}	Rogot1	$\frac{1}{2}\left(\frac{n_f(e)}{2\times n_f(e)+n_f(\overline{e})+n_s(e)}+\frac{n_s(\overline{e})}{2\times n_s(\overline{e})+n_f(\overline{e})+n_s(e)}\right)$
H_{26}	Kulczynski	$\frac{n_f(e)}{n_f(\overline{e})+n_s(e)}$
H_{27}	Anderberg	$\frac{n_f(e)}{n_f(e)+2\times(n_f(\overline{e})+n_s(e))}$
H_{28}	Dice	$\frac{2\times n_f(e)}{n_f(e)+n_f(\overline{e})+n_s(e)}$
H_{29}	Goodman	$\frac{2\times n_f(e)-n_f(\overline{e})-n_s(e)}{2\times n_f(e)+n_f(\overline{e})+n_s(e)}$
H_{30}	Jaccard	$\frac{n_f(e)}{n_f(e)+n_f(\overline{e})+n_s(e)}$
H_{31}	Sorensen-Dice	$\frac{2\times n_f(e)}{n_f(e)+n_f(\overline{e})+n_s(e)}$
H_{32}	Tarantula	$\frac{\frac{n_f(e)}{n_f}}{\frac{n_s(e)}{n_s}+\frac{n_f(e)}{n_f}}$
H_{33}	Ochiai	$\frac{n_f(e)}{\sqrt{n_f\times(n_f(e)+n_s(e))}}$
H_{34}	Ochiai2	$\frac{n_f(e)\times n_s(\overline{e})}{\sqrt{(n_f(e)+n_s(e))\times(n_s(\overline{e})+n_f(\overline{e}))\times(n_f(e)+n_f(\overline{e}))\times(n(e)p+n_s(\overline{e}))}}$

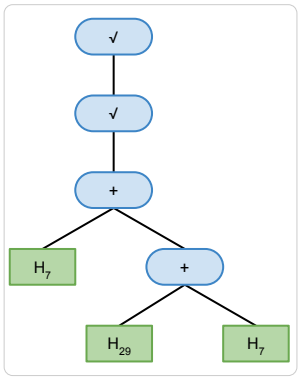

Figura 3.1: *Exemplo de solução encontrada (Fórmula 3-2).*

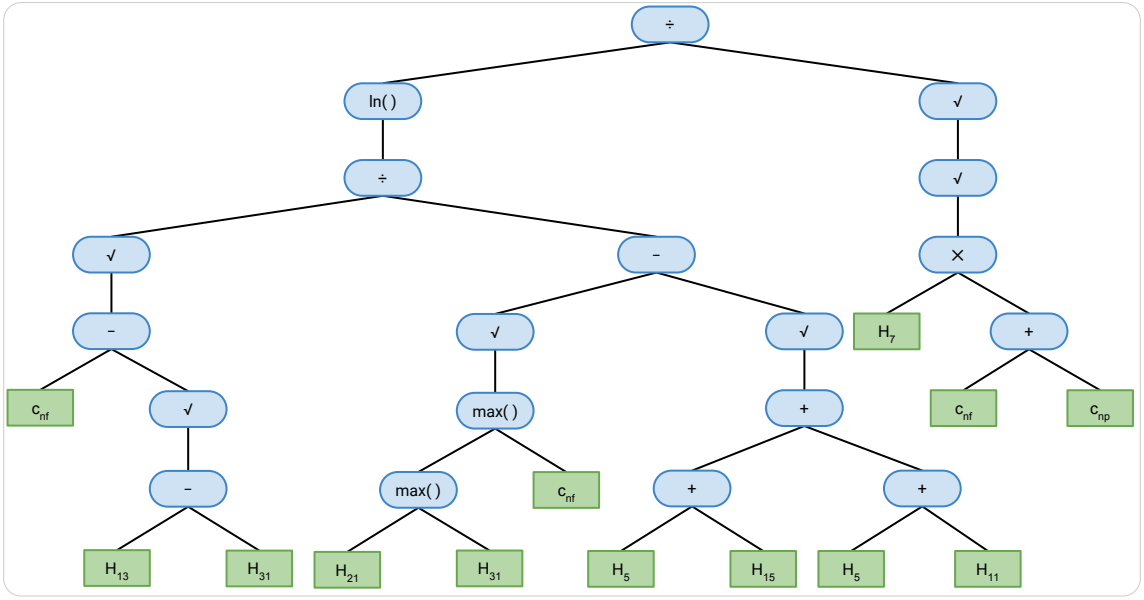

Figura 3.2: *Exemplo de solução encontrada (Fórmula 3-3).*

um conjunto de n versões defeituosas.

$$fitness(s) = \frac{\sum_{v=1}^{n} expense(s,v)}{n} \tag{3-1}$$

As Fórmulas 3-2 e 3-3 apresentam exemplos de soluções obtidas pelo algoritmo e estão representadas nas Figuras 3.1 e 3.2 respectivamente. Ressalta-se que a árvore poderia ter qualquer profundidade (altura), a depender da solução representada; os operandos (terminais) estão nos nós folha, e diversos operadores matemáticos (funções) estão nos nós internos. Existem diversos operadores representados na árvore abstrata, de modo que as equações têm complexidade matemática superior às combinações lineares de Wang *et al.*. [51].

$$\sqrt{\sqrt{(H_{29} + H_7)} + H_7} \tag{3-2}$$

$$\frac{\log\left(\frac{\left(\sqrt{\max\left(\max(H_{21},H_{30}),n_f(\overline{e})\right)}-\sqrt{M_5+H_{15}+M_5+H_{11}}\right)}{\sqrt{n_f(\overline{e})-\sqrt{H_{13}-H_{31}}}}\right)}{\sqrt[4]{M_7\times n(\overline{e})}} \tag{3-3}$$

A proposta apresentada evolui os trabalhos supracitados principalmente em dois aspectos:

1. As heurísticas combinadas resultantes não estão limitadas à Fórmula 2-5 de [51] (soma das heurísticas ponderada por pesos), com combinações mais complexas permitidas no espaço de busca;
2. A formulação de heurísticas SBFL *orientadas a projetos*, isto é, o treinamento de heurísticas para programas específicos, diferentemente dos trabalhos anteriores baseados em Programação Genética [55].

O método proposto é estruturado na metodologia de treinamento e implantação, conforme descrito no Capítulo 2. Durante a fase de treinamento, o algoritmo de Programação Genética busca a combinação de heurísticas SBFL que rende o menor *expense* médio em um conjunto de versões prévias com defeitos conhecidos. Na fase de implantação a melhor combinação de heurísticas encontrada ao final da fase de treinamento é aplicada às demais versões defeituosas do programa.

3.2 Experimentos

Devido ao caráter estocástico das técnicas avaliadas, os resultados obtidos durante os experimentos variam. Assim, especificou-se a quantidade de 30 execuções para cada meta-heurística, de modo a compensar este efeito. Adicionalente, foi realizada validação cruzada tripla (*three-fold cross validation*) ao dividir o conjunto de todas as versões de um programa aleatoriamente em três grupos, cada um com um terço do total. Destes, um grupo é utilizado para a fase de implantação enquanto os restantes são utilizados na fase de treinamento. Três iterações de experimentos são realizadas de modo que cada grupo de programas seja utilizado uma vez como o conjunto de implantação.

O esquema de experimentação está ilustrado na Figura 3.3, em que k e i representam respectivamente contadores da validação cruzada e do número de execuções da técnica. Conforme a validação cruzada tripla, cada k representa a utilização de uma das três partes do conjunto total de versões do programa como conjunto de implantação. Para cada k são realizadas 30 execuções da técnica. Ao final de cada execução, a melhor solução encontrada é aplicada ao conjunto de implantação correspondente a k e os resultados são coletados para análise conforme as métricas de avaliação. Ressalta-se que

a busca é realizada sempre no conjunto de treinamento, que é disjunto do conjunto de implantação.

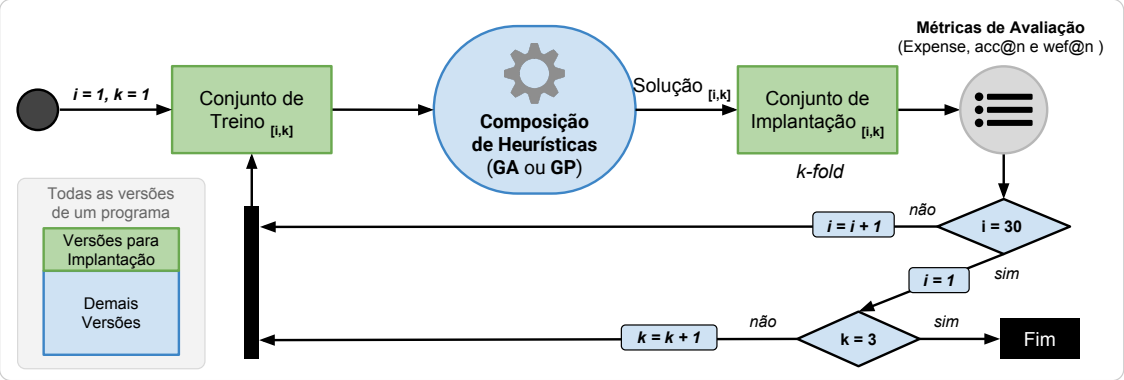

Figura 3.3: *Esquema de experimentação utilizado em composição de heurísticas baseadas em espectro de cobertura.*

O restante desta seção apresenta aspectos importantes da configuração dos experimentos conduzidos. Subseção 3.2.1 apresenta o conjunto de programas utilizado para experimentação; Subseção 3.2.2 mostra os parâmetros utilizados nas meta-heurísticas; Subseção 3.2.3 apresenta quais métodos foram selecionados para basear a análise da proposta; e Subseção 3.2.4 elenca as métricas de avaliação utilizadas.

3.2.1 Os *Benchmarks*

Para a condução dos experimentos foi utilizado um conjunto de programas obtidos no repositório SIR (*Software-artifact Infrastructure Repository*) [12], que é um projeto cujo objetivo é disponibilizar artefatos relacionados a software para experimentação relacionada a análise de programas, teste de software e educação. Foram selecionados os sete programas do *Simens Suite* [12]. Estes programas foram escolhidos por terem sido amplamente utilizados pela comunidade de localização de defeitos (vide [51] e [32]). Os programas do *Siemens Suite* estão sumarizados na Tabela 3.3.

Tabela 3.3: *Conjunto de programas utilizado nos experimentos*

Id	Programa	Linhas de Código	Versões Defeituosas	Conjunto de Teste
P1	printtokens	472	7 (4 utilizadas)	4030
P2	printtokens2	399	10	4115
P3	replace	512	32 (28 utilizadas)	5542
P4	schedule	292	9 (7 utilizadas)	2650
P5	schedule2	301	10 (9 utilizadas)	2710
P6	tcas	141	41 (30 utilizadas)	1608
P7	tot_info	440	23 (19 utilizadas)	1051

Para cada programa, existem diversas versões que contêm uma linha defeituosa. Em conformidade com as publicações de Wang *et al.* [51] e Lucia *et al.* [32], algumas versões não foram utilizadas nos experimentos pois o defeito encontrava-se em uma linha não instrumentalizável (defeitos de declaração de variável, por exemplo); tais versões foram: as versões 4 e 5 de *printtokens*; 12 de *replace*; 13, 14, 15, 36 e 38 de *tcas*; e 6, 10, 19 e 21 de *tot_info*. Ainda, alguams versões não foram utilizadas porque o conjunto de teste não revelou o defeito (não há teste negativo) ou o defeito não estava concentrado em uma única linha de código, elas são: 1 de *printtokens*; 6, 21 e 32 de *replace*; 2 e 7 de *schedule*; 9 de *schedule2*; e 10, 11, 31 ,32, 33 e 40 de *tcas*. O espectro de cobertura das demais 107 versões foram geradas com auxílio das ferramentas *lcov*, *gcov* (uma ferramenta GNU para análise de cobertura de testes) e um programa customizado.

Outra ferramenta utilizada foi o framework Java JGAP, que permite a abstração dos operadores básicos necessários para a implementação dos algoritmos evolucionários utilizados. Para a implementação do Algoritmo Genético e da Programação Genética, foram utilizadas as configurações mínimas necessárias pelo framework. Assim, os algoritmos foram utilizados em suas configurações canônicas com parâmetros mais conservadores possíveis.

3.2.2 Parâmetros das Meta-heurísticas

Os parâmetros aplicados aos algoritmos PG e AG foram os seguintes: 1000 gerações; população de 100 indivíduos; 90% de probabilidade de cruzamento; 10% de probabilidade de mutação; 10% de probabilidade de reprodução (somente PG); profundidade máxima de cruzamento 10 (somente PG); profundidade inicial máxima 10 (somente PG); profundidade inicial mínima 2 (somente PG).

Dado que de modo geral as meta-heurísticas são sensíveis a escolha dos parâmetros, foram utilizados os parâmetros sugeridos pela ferramenta JGAP para ambos algoritmos. Assim, somente configurações básicas de parâmetros necessários foram definidas e o mesmo conjunto de parâmetros foi aplicado a ambas abordagens (AG e PG) para uma comparação mais justa de resultados.

3.2.3 Avaliação do Método

Os experimentos realizados por Lucia *et al.* [32] e Wang *et al.* [51] foram desenvolvidos com uma abordagem de granularidade em blocos de código, isto é, o componente mínimo para a instrumentalização do código. Diferentemente, os experimentos aplicados na presente pesquisa utilizam a granularidade mínima de linha de código.

Conforme mencionado na Subseção 3.2.1, o framework JGAP foi utilizado para a implementação das meta-heurísticas Algoritmo Genético e Programação Genética.

Ambos os algoritmos foram utilizados com função de avaliação conforme utilizada por Wang *et al.* [51] e Yoo [55], definida na Equação 2-6.

Existem diversas heurísticas SBFL na literatura, de modo que se faz necessário definir um conjunto para a comparação com as técnicas estocásticas avaliadas. O presente trabalho apresenta dois grupos de heurísticas, associados às heurísticas SBFL que compõem os conjuntos de terminais T1 e T2 apresentados na Seção 3.1, além das heurísticas compostas obtidas pela técnica de composição linear baseada em Algoritmo Genético de Wang *et al.* [51]. Assim, as heurísticas utilizadas para análise são:

1. As heurísticas SBFL que compõem o conjunto de terminais $T1$ (ver Tabela 3.1) e a composição linear das heurísticas que compõem o conjunto $T1$ para o Algoritmo Genético ;
2. As heurísticas SBFL presentes no conjunto de terminais $T2$ (ver Tabela 3.2) e a composição linear das heurísticas pertencentes ao conjunto $T2$ para o Algoritmo Genético;

A notação AG_{T1} e AG_{T2} é utilizada para referenciar a técnica de composição linear de heurísticas SBFL baseada em Algoritmo Genético de Wang *et al.* quando configurada para o conjunto de heurísticas de $T1$ (Métricas de Associação apresentadas na Tabela 3.1 além das heurísticas Tarantula e Ochiai) e de $T2$ (todas as heurísticas SBFL da Tabela 3.2), respectivamente. Analogamente, a notação PG_{T1} e PG_{T2} é utilizada em menção à técnica de composição de heurísticas por Programação Genética proposta pelo presente trabalho quando configurada com os conjuntos de terminais $T1$ e $T2$, respectivamente.

3.2.4 Métricas de Avaliação

A função de avaliação, empregada durante a execução do algoritmo de Programação Genética, é calculada a partir dos ranqueamentos de suspeitas obtidos pela solução para cada um dos programas do conjunto de treinamento. Assim como Wang *et al.* [51], a aptidão de cada indivíduo é aferida utilizando-se a média da métrica *expense*, que retorna a posição relativa do comando defeituoso no ranking de suspeita em relação a todas as versões defeituosas no conjunto de treinamento. A Equação 3-1 representa a função de avaliação calculada para uma solução s e um conjunto de n versões defeituosas.

Para avaliar o desempenho das heurísticas SBFL em relação à proporção e ao número absoluto de elementos de código investigados para encontrar o defeito, foram aplicadas as seguintes métricas de avaliação:

- ***Expense* Médio**: afere o custo médio em termos da posição relativa dos comandos defeituosos nas listas de elementos de código ordenadas por suspeitas. Esta é a

métrica que baseia o cálculo da função de avaliação para os métodos baseados em busca utilizados. O *expense* é uma medida de custo pela quantidade média de linhas investigadas até que o defeito de um programa seja encontrado. O *expense* médio, por sua vez, calcula a média das medidas de *expense* em relação ao conjunto das versões defeituosas de um programa, conforme expresso pela Equação 3-4, em que: n representa o número de versões do programa analisado; d_v representa a posição do comando defeituoso na lista de elementos suspeitos da versão v; e c_v representa o número de comandos da versão v.

$$\textit{expense médio} = \frac{\sum_{v=1}^{n} \frac{d_v}{c_v}}{n} \tag{3-4}$$

A métrica *expense* é amplamente utilizada em trabalhos de localização de defeitos baseada em busca, como [31, 38, 41, 55, 56].

- ***Precision*** (**acc**@n): o número de defeitos que foram encontrados dentre os n elementos de maior suspeita (medidas maiores são melhores). Esta métrica, utilizada em trabalhos como [6, 28], calcula a precisão de uma técnica nas n primeiras posições da lista de elementos de código ordenada pelas suspeitas obtidas ao aplicar a mesma. Foram utilizados 1, 3 e 5 para n.
- ***Wasted Effort*** (**wef**@n): a quantidade de trabalho desperdiçado ao analisar cada elemento de código do programa que não é responsável pelo defeito antes de localizá-lo (valores menores são melhores). É obtida ao contar a quantidade de elementos investigados até que o primeiro defeito seja localizado ou n elementos sejam investigados. Assim, n representa um teto para suas medidas. Aparece em publicações como [6, 7, 28]. Também foram utilizados 1, 3 e 5 para n.
- ***Rank* Médio**: a posição média de cada técnica segundo seus resultados em cada programa para alguma outra métrica de avaliação. Para uma medida de avaliação, o *rank* médio de um método de localização de defeitos é a média de suas posições (primeiro, segundo, terceiro, etc.) calculadas para todos os programas. Se a medida para o método é igual a outra, ambas têm a mesma posição relativa: o pior caso (*e.g.* se três heurísticas têm a segunda melhor medida de $wef@3$, então a todas é atribuída a quarta posição no ranking). Foi utilizada na análise de trabalhos como [9, 10].

3.3 Análise

A eficácia do método em localizar defeitos foi analisada sob perspectivas relativas e absolutas, que são formalizadas na forma das seguintes questões de pesquisa:

- **QP1:** *Quão eficaz é o método em localizar defeitos sob uma perspectiva relativa?* A capacidade que a abordagem tem de atribuir valores de suspeita maiores às linhas que de fato são defeituosas é avaliada por uma métrica que calcula a proporção média de linhas de código do programa investigadas até encontrar o defeito.
- **QP2:** *Quão eficaz é o método em localizar defeitos sob uma perspectiva absoluta?* A eficácia do método em destacar as linhas defeituosas é medida por métricas que indicam a quantidade de defeitos dentre os n comandos de maior suspeita. Este tipo de análise pode perder a precisão ao comparar programas com quantidades de linhas diferentes, mas melhor se aproxima da realidade dos desenvolvedores, que independente do tamanho do projeto, necessitam investigar poucas linhas de código.

Para responder às questões de pesquisa **QP1** e **QP2** foram analisados: (i) as medidas de avaliação para o conjunto completo dos programas avaliados (Subseção 3.3.1); (ii) as classificações médias das heurísticas conforme todas as medidas de avaliação (Subseção 3.3.2); e (iii) a comparação de técnicas de localização de defeitos em programas individuais (Subseção 3.3.3).

Nas análises foram utilizadas as métricas de avaliação *Expense* Médio e *Rank* Médio para responder a questão QP1 e *Precision* e *Wasted Effort* para responder a questão QP2.

3.3.1 Avaliação do Conjunto de Programas Completo

A Tabela 3.4 resume os valores da métrica de avaliação relativa *Expense Médio* além das métricas de avaliação absolutas *acc@n* e *wef@n* em relação ao conjunto de todos os programas para o método baseado em Programação Genética configurado com conjunto de terminais $T1$ (PG_{T1}) e os compara ao método de Wang *et al.* [51] (AG_{T1}), utilizado para compor as heurísticas integrantes do conjunto de terminais $T1$, além das heurísticas SBFL individuais que compõem o próprio conjunto $T1$. Ressalta-se que os resultados de PG_{T1} e AG_{T1} representam valores médios de 30 execuções para cada um dos programas. Os resultados foram obtidos conforme o esquema de experimentação apresentado na Figura 3.3.

O método PG_{T1} foi capaz de obter heurísticas de melhores resultados ao observar o conjunto completo de programas. Observa-se que o *expense* médio de PG_{T1} (16,43%) é melhor em relação ao AG_{T1} (21,79%) e às heurísticas individuais do conjunto T1, o *expense* médio da melhor heurística individual (AM_{10}) é de 27,05%. Ainda, a técnica PG_{T1} obteve melhores valores de *precision* e *wasted effort*, com 11,24% de todos os defeitos na primeira posição do ranking de suspeitas e 53,80% de todos os defeitos dentre as primeiras dez posições.

Tabela 3.4: *Principais medidas de avaliação para o conjunto completo de programas ao avaliar PG_{T1}, AG_{T1} e as heurísticas SBFL individuais que compõem o Conjunto de Terminais T1*

Método	*Expense* Médio	acc@n				wef@n			
		1	**3**	**5**	**10**	**1**	**3**	**5**	**10**
AM_1	28,82%	7	24	27	36	100	273	435	796
AM_2	55,03%	1	7	10	21	106	310	506	960
AM_3	55,03%	1	7	10	21	106	310	506	960
AM_4	29,15%	7	24	27	39	100	271	433	786
AM_5	30,65%	8	22	36	43	99	272	418	760
AM_6	32,50%	7	20	32	41	100	278	431	784
AM_7	59,21%	1	5	8	18	106	312	512	976
AM_8	57,09%	1	5	10	17	106	312	508	972
AM_9	57,02%	1	5	10	17	106	312	508	972
AM_{10}	27,05%	7	24	27	36	100	272	434	795
AM_{11}	28,97%	7	24	27	39	100	271	433	786
AM_{12}	54,84%	1	7	10	21	106	310	506	960
AM_{13}	30,22%	7	24	27	38	100	272	434	788
AM_{14}	30,42%	7	21	24	31	100	279	447	833
AM_{15}	28,46%	7	25	31	41	100	270	424	772
AM_{16}	69,84%	0	0	0	1	107	321	535	1067
H_{32}	33,99%	7	21	24	39	100	279	447	804
H_{33}	30,25%	7	24	29	45	100	272	432	752
AG_{T1}	21,79%	11,07	29,07	34,77	52,17	95,93	258,90	405,37	705,17
PG_{T1}	16,43%	12,03	27,70	37,93	57,57	94,97	262,00	405,63	687,53

A Tabela 3.5 apresenta os valores de *expense* médio, $acc@n$ e $wef@n$ em relação ao conjunto de programas completo obtidos pela PG_{T2} (técnica de composição baseada em Programação Genética configurada com o conjunto de terminais $T2$) e os compara a AG_{T2} (método de Wang *et al.* [51] utilizado para compor as heurísticas integrantes de $T2$) e as heurísticas SBFL individuais que integram o conjunto $T2$.

Assim como no conjunto de heurísticas de $T1$, PG_{T2} rendeu o menor *expense* médio com o conjunto de terminais $T2$. As heurísticas compostas obtidas através de PG_{T2} alcançaram *expense* médio de 20,57%, enquanto o segundo melhor resultado foi de 23,59%, para a heurística H_29. AG_{T2} obteve *expense* médio de 28,50%, resultado este inferior em relação a proposta e a outras duas heurísticas SBFL individuais (H_16 e H_29).

Para as métricas de avaliação absolutas de $acc@n$ e $wef@n$ de $n = 1$, os resultados de PG_{T2} foram os melhores em relação ao conjunto de heurísticas de $T2$ e AG_{T2}. Isto representa melhores resultados para o esforço mínimo para localizar defeitos, isto é, quando a heurística indica a linha defeituosa com a maior suspeita. Apesar do desempenho superior de H_{16} e H_{29} com valores mais altos de n, os resultados de PG_{T2} se

Tabela 3.5: *Principais medidas de avaliação para o conjunto completo de programas ao avaliar PG_{T2}, AG_{T2} e as heurísticas SBFL individuais que compõem o Conjunto de Terminais T2*

Método	*Expense* Médio	acc@n				wef@n			
		1	**3**	**5**	**10**	**1**	**3**	**5**	**10**
H_1	33,86	7	21	25	39	100	279	446	802
H_2	30,17	7	23	28	43	100	276	438	771
H_3	56,55	0	0	3	15	107	321	531	1023
H_4	30,37	0	5	6	20	107	316	518	979
H_5	36,16	7	24	27	41	100	274	436	775
H_6	52,58	0	0	0	3	107	321	535	1066
H_7	30,31	5	20	30	47	102	282	441	764
H_8	96,11	1	2	2	2	106	316	526	1051
H_9	39,50	7	20	24	35	100	280	449	830
H_{10}	34,05	7	21	24	39	100	279	447	804
H_{11}	34,67	7	25	28	41	100	269	429	775
H_{12}	29,02	7	24	29	45	100	273	433	753
H_{13}	28,77	7	21	28	45	100	278	442	769
H_{14}	30,85	7	21	26	40	100	279	445	794
H_{15}	48,30	7	15	17	22	100	288	469	906
H_{16}	26,67	8	28	38	55	99	263	405	685
H_{17}	29,06	7	24	29	46	100	272	432	750
H_{18}	29,03	7	24	29	46	100	272	432	750
H_{19}	50,85	4	6	8	11	103	306	505	997
H_{20}	49,84	0	0	3	15	107	321	531	1023
H_{21}	50,85	4	6	8	11	103	306	505	997
H_{22}	50,85	4	6	8	11	103	306	505	997
H_{23}	50,85	4	6	8	11	103	306	505	997
H_{24}	29,14	7	24	29	48	100	271	431	743
H_{25}	48,10	7	15	17	21	100	288	469	905
H_{26}	33,18	7	21	26	40	100	279	445	792
H_{27}	33,18	7	21	26	40	100	279	445	792
H_{28}	33,18	7	21	26	40	100	279	445	792
H_{29}	23,59	8	28	38	56	99	263	405	684
H_{30}	33,18	7	21	26	40	100	279	445	792
H_{31}	33,18	7	21	26	40	100	279	445	792
H_{32}	33,99	7	21	24	39	100	279	447	804
H_{33}	30,25	7	24	29	45	100	272	432	752
H_{34}	35,63	7	22	28	43	100	277	439	772
AG_{T2}	28,50%	0,33	6,77	9,00	22,43	106,67	313,20	509,33	961,20
PG_{T2}	**20,57%**	**8,77**	**23,27**	**35,53**	**53,53**	**98,23**	**272,23**	**419,67**	**722,47**

aproximam de H_{16} e H_{29}. Os resultados de AG_{T2} são ruins em relação a maior parte das heurísticas. Isto se deve à linearidade das equações do AG_{T2} e ao tamanho do espaço de busca maior em relação ao conjunto $T1$.

Ao comparar os resultados presentes nas Tabelas 3.4 e 3.5 nota-se que PG_{T1} foi mais interessante em perspectivas relativas e absolutas. Os resultados de PG_{T1} foram superiores em relação a todas as alternativas (Tabelas 3.4 e 3.5) para as medidas de *expense* médio, *acc@1*, *acc@10*, *wef@1* e *wef@10*. O melhor resultado de *acc@3*, *acc@5*, *wef@3* e *wef@5* foi obtido pela heurística H_{16}, mas apenas marginalmente superior aos valores de PG_{T1}.

3.3.2 Avaliação de *Rank* Médio

Para capturar o desempenho relativo das heurísticas SBFL em cada projeto individualmente foi aferido o *rank* médio, que representa a classificação média de cada técnica em relação às medidas de *expense* médio *acc@n* e *wef@n* em cada programa. O *rank* médio foi mensurado para três combinações dos conjuntos de heurísticas avaliadas, são elas:

1. As heurísticas SBFL que compõem o conjunto de terminais $T1$ e os dois métodos de composição configurados com o conjunto de terminais $T1$ (PG_{T1} e AG_{T1});
2. As heurísticas SBFL presentes no conjunto de terminais $T2$ e os dois métodos de composição configurados com o conjunto de terminais $T2$ (PG_{T2} e AG_{T2});
3. Todas as heurísticas apresentadas (presentes nos conjuntos de terminais $T1$ e $T2$) e as duas variações de ambas as técnicas de composição (PG_{T1}, AG_{T1}, PG_{T2} e AG_{T2}).

As Tabelas 3.6, 3.7 e 3.8 apresentam os cinco melhores valores de *rank* médio para os três escopos em relação às medidas de *expense* médio, *precision* (*acc@m*) e *wasted effort* (*wef@n*). O Apêndice A apresenta tabelas com todos os valores de *rank* médio.

Tabela 3.6: *Cinco melhores valores de rank médio para as heurísticas que compõem o conjunto de terminais $T1$ e os métodos de composição a ele relacionados (PG_{T1} e AG_{T1})*

***Expense* Médio**	**acc@1**	**acc@3**	**acc@5**	**acc@10**	**wef@1**	**wef@3**	**wef@5**	**wef@10**
AG_{T1}: 1,43	**PG**$_{T1}$: 4,57	AG_{T1}: 6,57	**PG**$_{T1}$: 4,43	**PG**$_{T1}$: 2,57	**PG**$_{T1}$: 4,57	**PG**$_{T1}$: 5,14	AG_{T1}: 5,14	AG_{T1}: 4,86
PG$_{T1}$: 3,43	AG_{T1}: 10,00	**PG**$_{T1}$: 6,57	AM_5: 6,29	AG_{T1}: 7,43	AG_{T1}: 10,00	AG_{T1}: 5,86	**PG**$_{T1}$: 5,14	**PG**$_{T1}$: 5,14
AM_{10}: 5,14	AM_5: 12,71	AM_{15}: 11,00	AG_{T1}: 7,00	AM_5: 9,57	AM_5: 12,71	AM_{15}: 9,43	AM_5: 8,29	H_{33}: 8,00
AM_{15}: 5,86	AM_6: 12,86	AM_1: 12,00	AM_6: 9,43	AM_6: 10,00	AM_6: 12,86	AM_4: 10,43	AM_{15}: 9,14	AM_{15}: 8,14
H_{33}: 6,29	AM_1: 17,43	AM_4: 12,00	AM_{15}: 10,71	H_{33}: 10,57	AM_1: 17,43	AM_{11}: 10,43	H_{33}: 10,57	AM_5: 8,29

O *rank* médio para as heurísticas que compõem o conjunto de terminais $T1$ e os métodos de composição a ele relacionados (PG_{T1} e AG_{T1}) demonstram o quão competitivo é PG_{T1} em relação às outras métricas de avaliação, apresentando os menores

valores para todas as variações da métrica de *precision acc@n* e *wef@n*, exceto *acc@3*, em que houve empate com AG_{T1}, e *wef@10* em que perdeu para AG_{T1}. Isto indica que dentre todas os métodos do conjunto, em média PG_{T1} foi o melhor método para colocar defeitos no topo da lista de linhas suspeitas. Este resultado corrobora o resultado obtido para o conjunto total de programas, de modo que o desempenho é consistente para todos os programas quando considerando-se o quanto o método se destacou dos demais.

Para o *rank* médio da métrica *Expense* Médio, PG_{T1} foi o segundo melhor método, atrás de AG_{T1}. Dado o valor baixo do *rank* médio, nota-se que AG_{T1} foi a melhor opção isoladamente para a maioria dos programas quanto ao melhor *expense* médio, mas que PG_{T1} permaneceu competitivo em relação às demais heurísticas do conjunto. Como o valor *expense* médio aferido no conjunto completo de programas para PG_{T1} é melhor que AG_{T1}, é possível afirmar que o as diferenças entre PG_{T1} e AG_{T1} são pequenas nos programas em que AG_{T1} tem melhor *expense* médio. Ou seja, apesar de PG_{T1} não ter sido a melhor opção para *expense* médio tanto quanto AG_{T1}, nas ocasiões em que AG_{T1} foi melhor, a diferença para PG_{T1} é pequena.

Tabela 3.7: *Cinco melhores valores de rank médio para as heurísticas que compõem o conjunto de terminais $T2$ e os métodos de composição a ele relacionados (PG_{T2} e AG_{T2})*

***Expense* Médio**	**acc@1**	**acc@3**	**acc@5**	**acc@10**	**wef@1**	**wef@3**	**wef@5**	**wef@10**
H_{29}: 2,57	**PG**$_{T2}$: 15,57	**PG**$_{T2}$: 9,57	**PG**$_{T2}$: 5,86	**PG**$_{T2}$: 5,29	**PG**$_{T2}$: 15,57	**PG**$_{T2}$: 7,86	H_{16}: 8,14	H_{29}: 3,00
PG$_{T2}$: 5,86	H_7: 25,14	H_{16}: 16,43	H_{16}: 9,00	H_{29}: 9,14	H_7: 25,14	H_{11}: 14,14	H_{29}: 8,14	H_{16}: 3,14
H_{16}: 6,00	H_{16}: 30,00	H_{29}: 16,43	H_{29}: 9,00	H_{16}: 9,57	H_{16}: 30,00	H_{16}: 14,14	**PG**$_{T2}$: 8,14	H_{24}: 8,43
H_{13}: 7,29	H_{29}: 30,00	H_{11}: 17,57	H_7: 18,57	H_7: 16,57	H_{29}: 30,00	H_{29}: 14,14	H_{11}: 14,14	**PG**$_{T2}$: 8,43
H_{33}: 9,14	H_8: 31,00	H_5: 20,57	H_{12}: 19,86	H_{13}: 16,71	H_8: 31,00	H_{24}: 15,86	H_{24}: 14,29	H_{33}: 9,29
H_{24}: 9,86	AG_{T2}: 31,29	H_{12}: 20,86	H_{17}: 19,86	H_{24}: 18,14	AG_{T2}: 31,29	H_{17}: 16,71	H_{17}: 15,14	H_{17}: 9,57

No *rank* médio aferido em relação às heurísticas que compõem o conjunto de terminais $T2$ e os métodos de composição configurados com o mesmo (Tabela 3.7), destaca-se que os valores de PG_{T2} são superiores para quase todas as métricas de avaliação. Isto ocorre devido a consistência entre os melhores resultados quando observando-se programas individualmente para PG_{T2}.

Para a métrica *Expense* Médio, o *rank* médio não confirmou o valor geral aferido no conjunto completo de programas apresentado na Tabela 3.5. Embora PG_{T2} tenha apresentado o menor valor de *expense* médio, a vantagem do *rank* médio da heurística H_{29} indica menor consistência para PG_{T2} nesta métrica, apesar de ser competitiva.

Finalmente, foi aferido o *rank* médio para o conjunto completo das heurísticas analisadas (presentes nos conjuntos de terminais $T1$ e $T2$) e as duas variações de ambas as técnicas de composição (Tabela 3.8). Observa-se que PG_{T1} foi o método mais consistente, com menor *rank* médio em diversas métricas de avaliação. Ainda, PG_{T1} e PG_{T2} foram os

Tabela 3.8: *Cinco melhores valores de rank médio para todas as heurísticas apresentadas (presentes nos conjuntos de terminais $T1$ e $T2$) e as técnicas de composição*

***Expense* Médio**	**acc@1**	**acc@3**	**acc@5**	**acc@10**	**wef@1**	**wef@3**	**wef@5**	**wef@10**
AG_{T1}: 2,57	**PG**$_{T1}$: 9,86	**PG**$_{T1}$: 15,29	**PG**$_{T1}$: 7,86	**PG**$_{T1}$: 5,29	**PG**$_{T1}$: 9,86	**PG**$_{T1}$: 10,71	**PG**$_{T1}$: 11,00	H_{29}: 4,86
H_{29}: 4,86	**PG**$_{T2}$: 23,57	**PG**$_{T2}$: 16,29	**PG**$_{T2}$: 10,29	**PG**$_{T2}$: 7,43	**PG**$_{T2}$: 23,57	**PG**$_{T2}$: 13,57	H_{16}: 12,43	H_{16}: 5,00
PG$_{T1}$: 7,71	AG_{T1}: 24,71	AG_{T1}: 18,00	H_{16}: 14,14	H_{29}: 14,29	AG_{T1}: 24,71	AG_{T1}: 13,71	H_{29}: 12,43	AG_{T1}: 11,14
H_{16}: 9,71	AM_5: 34,43	H_{16}: 24,71	H_{29}: 14,14	H_{16}: 14,86	AM_5: 34,43	H_{16}: 21,71	AG_{T1}: 12,71	**PG**$_{T1}$: 11,14
PG$_{T2}$: 9,71	AM_6: 34,57	H_{29}: 24,71	AG_{T1}: 15,57	AG_{T1}: 16,86	AM_6: 34,57	H_{29}: 21,71	**PG**$_{T2}$: 13,29	**PG**$_{T2}$: 13,86

métodos com melhor *rank* médio para as métricas de *precision* (*acc@n*) e *wasted effort* (*wef@n*).

Novamente, para a métrica *Expense* Médio, o *rank* médio de PG_{T1} e PG_{T2} não foi o menor encontrado, com ambas perdendo de AG_{T1} e da heurística H_{29} e PG_{T2} empatado com a heurística H_{16}. Este resultado é consistente com os *ranks* médios apresentados anteriormente e reforça que AG_{T1} é o método mais consistente em cada programa para *expense* médio, mas PG_{T1} e PG_{T2} permanecem competitivas nesta métrica.

Embora os resultados dos *ranks* médios para PG_{T1} e PG_{T2} (com destaque para PG_{T1}) sejam os menores de modo geral, ainda há valores altos, o que indica que em alguns casos outras heurísticas são superiores. Assim, destaca-se que, embora os resultados sejam bons, há espaço para melhora.

3.3.3 Avaliação de Programas Individuais

As Tabelas 3.9 e 3.10 apresentam os valores de *expense* médio de cada programa para heurísticas que compõem o conjunto de terminais $T1$ e $T2$ respectivamente e os métodos de composição configurados com o mesmos. Estão representados os valores para as cinco heurísticas com melhores medidas de *expense* médio de todos os programas, coluna **Total**. A primeira linha refere-se aos identificadores de cada programa com respetivo número de versões (entre parêntesis), conforme descrito na Tabela 3.3.

Tabela 3.9: *Proporção média de código investigado para encontrar todos os defeitos: Expense Médio*

Método	**P1 (4)**	**P2 (10)**	**P3 (28)**	**P4 (7)**	**P5 (9)**	**P6 (30)**	**P7 (19)**	**Total**
$\boldsymbol{PG}_{T1}$	9,44	21,84	10,38	5,77	30,69	21,77	12,69	16,43
$\boldsymbol{AG}_{T1}$	3,21	11,09	8,45	9,13	51,23	40,34	12,44	21,79
$\boldsymbol{AM}_{10}$	4,23	18,42	12,16	10,54	55,21	44,82	23,04	27,05
$\boldsymbol{AM}_{15}$	4,11	21,79	11,78	10,54	61,43	45,95	25,07	28,46
$\boldsymbol{AM}_{1}$	4,36	22,08	11,82	10,54	60,66	46,26	26,67	28,82

Apesar de PG_{T1} e PG_{T2} aparecerem no topo das Tabelas 3.9 e 3.10 respectivamente, ambas as técnicas não apresentam os melhores resultados em todos os programas. Estes resultados confirmam a análise de *rank* médio apresentada na Subseção 3.3.2, visto

Tabela 3.10: *Proporção média de código investigado para encontrar todos os defeitos: Expense Médio*

Método	P1 (4)	P2 (10)	P3 (28)	P4 (7)	P5 (9)	P6 (30)	P7 (19)	Total
$\boldsymbol{PG_{T2}}$	20,43%	20,23%	9,06%	22,23%	40,05%	26,16%	19,09%	20,57%
$\boldsymbol{H_{29}}$	3,46%	11,91%	8,14%	26,15%	46,40%	41,74%	16,36%	23,59%
$\boldsymbol{H_{16}}$	3,46%	18,71%	9,19%	58,37%	46,66%	41,74%	16,53%	26,67%
$\boldsymbol{AG_{T2}}$	9,81%	25,36%	13,48%	19,37%	33,65%	48,81%	25,10%	28,50%
$\boldsymbol{H_{13}}$	4,11%	24,13%	10,58%	26,43%	57,12%	45,33%	24,51%	28,77%

que havia indícios de que apesar de PG_{T1} e PG_{T2} apresentarem os melhores valores de *expense* médio para o conjunto completo de programas, o mesmo não aconteceu para os valores de *rank* médio referente a mesma métrica de avaliação.

PG_{T1} e PG_{T2} obtiveram o melhor *expense* médio total, mas perderam em alguns programas, mais notável o programa P1 (*printtokens*). Conjectura-se que este fenômeno ocorre, dentre outros fatores, devido à dependência da fase de treinamento, visto que com a validação cruzada tripla PG_{T1} e PG_{T2} encontraram resultados melhores em programas com maior número de versões defeituosas, como no programa P6 (*tcas*), por exemplo. Resultados muito ruins foram obtidos para programas com poucas versões, especialmente o programa P1, que possui apenas 4 versões.

Observa-se que PG_{T1} e PG_{T2} não foram capazes de encontrar uma heurística generalizável ao conjunto de implantação durante a busca no conjunto de treinamento. Entretanto AG_{T1} e AG_{T2}, também baseadas em treinamento, obtiveram boas soluções. Como o método baseado em Programação Genética permite combinações mais diversas e de maior complexidade, soluções muito mais especializadas em relação ao método de Wang *et al.* [51] foram encontradas durante a fase de busca no conjunto de treinamento. Dado o tamanho do conjunto de poucas versões, a busca não dispunha de dados suficientes para derivar uma solução suficientemente generalizável ao conjunto de implantação.

As Tabelas 3.11 e 3.12 resumem as cinco melhores medidas de *precision* (*acc@n*) e *wasted effort* (*wef@n*) para as heurísticas que compõem o conjunto de terminais $T1$ e $T2$ respectivamente e os métodos de composição configurados com o mesmos. As células mostram o par *técnica utilizada : resultado obtido*. Cada coluna apresenta uma versão da métrica de avaliação com um valor diferente para n e é organizada independentemente das demais em ordem não decrescente para *acc@n* e não crescente para *wef@n*.

Os resultados de PG_{T1} (em negrito na Tabela 3.11) para os programas P4 (*schedule*), P5 *schedule2*) e P6 (*tcas*) foram sempre os melhores ou muito próximos das melhores medidas de *precision* e *wasted effort*. No programa P3 (*replace*) e P7 (*tot_info*), AG_{T1} e PG_{T1} apresentaram resultados muito próximos, com ambos em geral superiores às melhores heurísticas SBFL individuais.

Nos programas P1 (*printtokens*) e P2 (*printtokens2*) PG_{T1} apresentou resultados inferiores ou pouco melhores em relação às outras heurísticas, especialmente PG_{T1}.

Tabela 3.11: *Cinco melhores técnicas em valores de precision (acc@n) e wasted effort (wef@n) dentre as heurísticas SBFL individuais que compõem o conjunto de terminais T1, PG_{T1} e AG_{T1}*

P1 – printtokens – (4 versions)							
acc@1	**acc@3**	**acc@5**	**acc@10**	**wef@1**	**wef@3**	**wef@5**	**wef@10**
$\boldsymbol{PG}_{:T1}$: 0,70	H_1: 3	AG_{T1}: 3,00	H_{33}: 4	$\boldsymbol{PG}_{:T1}$: 3,30	H_4: 8	H_4: 10	H_{33}: 10
AG_{T1}: 0,00	H_4: 3	H_1: 3	$\boldsymbol{PG}_{:T1}$: 3,23	AG_{T1}: 4,00	H_{10}: 8	H_{10}: 10	H_4: 15
H_1: 0	H_{10}: 3	H_4: 3	AG_{T1}: 3,00	H_1: 4	H_{11}: 8	H_{11}: 10	H_{10}: 15
H_2: 0	H_{11}: 3	H_5: 3	H_1: 3	H_2: 4	H_{15}: 8	H_{15}: 10	H_{11}: 15
H_3: 0	H_{13}: 3	H_6: 3	H_4: 3	H_3: 4	H_{33}: 8	H_{33}: 10	H_{15}: 15
P2 – printtokens2 – (10 versions)							
acc@1	**acc@3**	**acc@5**	**acc@10**	**wef@1**	**wef@3**	**wef@5**	**wef@10**
$\boldsymbol{PG}_{:T1}$: 2,23	AG_{T1}: 4,37	AG_{T1}: 6,00	AG_{T1}: 6,00	$\boldsymbol{PG}_{:T1}$: 7,77	AG_{T1}: 19,53	AG_{T1}: 28,53	AG_{T1}: 48,53
AG_{T1}: 2,10	H_1: 4	H_5: 5	H_5: 5	AG_{T1}: 7,90	H_1: 20	$\boldsymbol{PG}_{:T1}$: 31,60	H_5: 58
H_1: 2	H_4: 4	H_6: 5	H_6: 5	H_1: 8	H_4: 20	H_1: 32	H_6: 58
H_4: 2	H_5: 4	$\boldsymbol{PG}_{:T1}$: 4,33	$\boldsymbol{PG}_{:T1}$: 4,60	H_4: 8	H_{10}: 20	H_4: 32	$\boldsymbol{PG}_{:T1}$: 59,13
H_{10}: 2	H_6: 4	H_1: 4	H_1: 4	H_{10}: 8	H_{11}: 20	H_{10}: 32	H_1: 62
P3 – replace – (28 versions)							
acc@1	**acc@3**	**acc@5**	**acc@10**	**wef@1**	**wef@3**	**wef@5**	**wef@10**
AG_{T1}: 7,07	H_{15}: 11	H_5: 12	AG_{T1}: 20,00	AG_{T1}: 20,93	AG_{T1}: 58,07	AG_{T1}: 90,93	AG_{T1}: 143,30
$\boldsymbol{PG}_{:T1}$: 5,67	AG_{T1}: 10,63	H_{15}: 12	$\boldsymbol{PG}_{:T1}$: 18,57	$\boldsymbol{PG}_{:T1}$: 22,33	H_{15}: 60	H_{15}: 92	$\boldsymbol{PG}_{:T1}$: 152,00
H_1: 5	H_1: 10	AG_{T1}: 11,57	H_{33}: 17	H_1: 23	$\boldsymbol{PG}_{:T1}$: 60,47	$\boldsymbol{PG}_{:T1}$: 94,07	H_{33}: 156
H_4: 5	H_4: 10	$\boldsymbol{PG}_{:T1}$: 11,37	H_{15}: 15	H_4: 23	H_1: 61	H_5: 96	H_{15}: 167
H_{10}: 5	H_{10}: 10	H_6: 11	H_4: 14	H_{10}: 23	H_4: 61	H_1: 97	H_{32}: 171
P4 – schedule – (7 versions)							
acc@1	**acc@3**	**acc@5**	**acc@10**	**wef@1**	**wef@3**	**wef@5**	**wef@10**
H_5: 1	$\boldsymbol{PG}_{:T1}$: 3,73	$\boldsymbol{PG}_{:T1}$: 5,53	$\boldsymbol{PG}_{:T1}$: 5,93	H_5: 6	$\boldsymbol{PG}_{:T1}$: 14,63	$\boldsymbol{PG}_{:T1}$: 18,23	$\boldsymbol{PG}_{:T1}$: 24,13
H_6: 1	AG_{T1}: 3,10	H_5: 4	H_5: 4	H_6: 6	AG_{T1}: 15,23	H_5: 22	H_5: 37
AG_{T1}: 0,90	H_2: 2	AG_{T1}: 3,13	AG_{T1}: 3,13	AG_{T1}: 6,10	H_5: 16	AG_{T1}: 22,97	AG_{T1}: 42,30
$\boldsymbol{PG}_{:T1}$: 0,87	H_3: 2	H_6: 3	H_6: 3	$\boldsymbol{PG}_{:T1}$: 6,13	H_6: 16	H_6: 24	H_6: 44
H_1: 0	H_5: 2	H_1: 2	H_1: 2	H_1: 7	H_1: 19	H_2: 29	H_2: 54
P5 – schedule2 – (9 versions)							
acc@1	**acc@3**	**acc@5**	**acc@10**	**wef@1**	**wef@3**	**wef@5**	**wef@10**
$\boldsymbol{PG}_{:T1}$: 1,40	$\boldsymbol{PG}_{:T1}$: 1,73	$\boldsymbol{PG}_{:T1}$: 1,83	$\boldsymbol{PG}_{:T1}$: 3,17	$\boldsymbol{PG}_{:T1}$: 7,60	$\boldsymbol{PG}_{:T1}$: 22,20	$\boldsymbol{PG}_{:T1}$: 36,63	$\boldsymbol{PG}_{:T1}$: 68,33
AG_{T1}: 0,00	H_2: 1	H_2: 1	AG_{T1}: 1,00	AG_{T1}: 9,00	H_2: 25	H_2: 41	H_2: 81
H_1: 0	H_3: 1	H_3: 1	H_1: 1	H_1: 9	H_3: 25	H_3: 41	H_3: 81
H_2: 0	H_7: 1	H_7: 1	H_2: 1	H_2: 9	H_7: 25	H_7: 41	H_7: 81
H_3: 0	H_8: 1	H_8: 1	H_3: 1	H_3: 9	H_8: 25	H_8: 41	H_8: 81
P6 – tcas – (30 versions)							
acc@1	**acc@3**	**acc@5**	**acc@10**	**wef@1**	**wef@3**	**wef@5**	**wef@10**
$\boldsymbol{PG}_{:T1}$: 0,00	AG_{T1}: 5,00	H_5: 7	H_2: 12	$\boldsymbol{PG}_{:T1}$: 30,00	H_4: 80	H_{15}: 126	H_{15}: 223
AG_{T1}: 0,00	H_1: 5	H_{15}: 7	H_3: 12	AG_{T1}: 30,00	H_{11}: 80	AG_{T1}: 129,90	H_4: 227
H_1: 0	H_4: 5	H_2: 6	H_7: 12	H_1: 30	H_{13}: 80	H_4: 130	H_{11}: 227
H_2: 0	H_{10}: 5	H_3: 6	H_8: 12	H_2: 30	H_{15}: 80	H_5: 130	H_{13}: 227
H_3: 0	H_{11}: 5	H_7: 6	H_9: 12	H_3: 30	AG_{T1}: 80,03	H_{11}: 130	H_1: 229
P7 – tot_info – (23 versions)							
acc@1	**acc@3**	**acc@5**	**acc@10**	**wef@1**	**wef@3**	**wef@5**	**wef@10**
H_5: 2	AG_{T1}: 3,97	$\boldsymbol{PG}_{:T1}$: 6,60	$\boldsymbol{PG}_{:T1}$: 10,20	H_5: 17	H_5: 50	AG_{T1}: 77,03	$\boldsymbol{PG}_{:T1}$: 128,20
H_6: 2	$\boldsymbol{PG}_{:T1}$: 3,47	AG_{T1}: 6,00	AG_{T1}: 8,77	H_6: 17	H_6: 50	$\boldsymbol{PG}_{:T1}$: 77,57	AG_{T1}: 130,27
$\boldsymbol{PG}_{:T1}$: 1,17	H_5: 3	H_5: 5	H_5: 7	$\boldsymbol{PG}_{:T1}$: 17,83	AG_{T1}: 50,03	H_5: 79	H_6: 144
AG_{T1}: 1,00	H_6: 3	H_6: 5	H_6: 7	AG_{T1}: 18,00	$\boldsymbol{PG}_{:T1}$: 50,63	H_6: 79	H_5: 145
H_1: 0	H_1: 1	H_{33}: 5	H_{32}: 6	H_1: 19	H_1: 56	H_{33}: 87	H_{33}: 152

Novamente, estes resultados menos competitivos foram influenciados pelo conjunto pequeno de versões, conforme discutido anteriormente. Entretanto, as medições de *acc@1* e *wef@1* para AG_{T1} foram os melhores nestes programas, evidenciando que mesmo so-

frendo com baixa generalidade das soluções encontradas durante o treinamento em algumas execuções e baixo *expense* médio, o método foi capaz de colocar a linha defeituosa no topo da lista de linhas suspeitas.

Em relação aos resultados do conjunto das heurísticas que compõem o conjunto de terminais $T2$, AG_{T2} e PG_{T2}, observa-se que os valores de PG_{T2} (em negrito na Tabela 3.12) em P3 (*replace*), P4 (*schedule*), P5 (*schedule2*) e P7 (*tot_info*) estão presentes em todas as colunas da tabela, sempre entre as três primeiras posições, demonstrando melhor adaptação de desempenho em relação as outras técnicas. Os valores de PG_{T2} estão na primeira posição em 17 de 32 métricas de avaliação. Para os demais programas, os valores de PG_{T2} também são constantes entre as top-5 melhores técnicas para valores de *acc* e *wef*.

A análise das métricas de avaliação em programas individuais soma ao *rank* médio ao demonstrar que o método de composição de heurísticas baseado em Programação Genética é competitivo, embora sofra mais com a qualidade do treinamento, observada simplesmente pela quantidade de versões disponíveis no conjunto, em relação ao método de Wang *et al.* [51].

3.3.4 Análise Estatística

Seguindo as diretrizes de Arcuri *et al.* [5], o teste de *Wilcoxon* foi utilizado para avaliar a significância da diferença estatística entre as amostras, com nível de confiança de 95%. Complementarmente, foi utilizado o teste estatístico $\hat{A}_{12}$ de *Vargha e Delaney* para comparação de tamanho de efeito, quão superiores os resultados de uma amostra são em relação à outra.

Como não é possível garantir que os dados pertencem a uma distribuição normal, o teste de *Wilcoxon* é uma ferramenta de análise apropriada para comparação estatística entre os resultados dos algoritmos estocásticos AG_{T1} e PG_{T1} e AG_{T2} e PG_{T2} para a métrica de avaliação *Expense Médio* (conforme Tabelas 3.9 e 3.10). O teste de *Wilcoxon* calcula um *p-valor*, que sugere a existência de significância estatística da diferença de desempenho entre dois algoritmos quando este é inferior ao nível de significância (0,05 neste caso) [5].

O teste estatístico $\hat{A}_{12}$ de *Vargha* e *Delaney* foi empregado para avaliar o tamanho do efeito não paramétrico para a diferença entre os dois algoritmos [49]. Dada a medida de desempenho qualquer, $\hat{A}_{12}$ calcula a probabilidade de que o algoritmo A_1 obtenha p-valores maiores que o algoritmo A_2 [5]. Quando os dois algoritmos são equivalentes, então $\hat{A}_{12} = 0{,}5$.

Os testes foram realizados comparando-se 30 execuções de ambos algoritmos para cada conjunto de implantação. Tabelas 3.13 e 3.14 organizam os resultados de ambos

Tabela 3.12: *Cinco melhores técnicas em valores de precision (acc@n) e wasted effort (wef@n) dentre as heurísticas SBFL individuais que compõem o conjunto de terminais T2, PG_{T2} e AG_{T2}*

P1 – printtokens – (4 versions)							
acc@1	**acc@3**	**acc@5**	**acc@10**	**wef@1**	**wef@3**	**wef@5**	**wef@10**
H_7: 1	H_2: 3	H_2: 3	H_2: 4	H_7: 3	H_7: 7	H_7: 9	H_{16}: 10
$\boldsymbol{PG_{T2}}$: 0,67	H_5: 3	H_5: 3	H_5: 4	$\boldsymbol{PG_{T2}}$: 3,33	H_{33}: 8	H_{11}: 10	H_{24}: 10
AG_{T2}: 0,33	H_{11}: 3	H_7: 3	H_7: 4	AG_{T2}: 3,67	H_{29}: 8	H_{16}: 10	H_{29}: 10
H_1: 0	H_{12}: 3	H_{11}: 3	H_{12}: 4	H_1: 4	H_{24}: 8	H_{24}: 10	H_{33}: 10
H_2: 0	H_{16}: 3	H_{12}: 3	H_{13}: 4	H_2: 4	H_{16}: 8	H_{29}: 10	H_2: 11
P2 – printtokens2 – (10 versions)							
acc@1	**acc@3**	**acc@5**	**acc@10**	**wef@1**	**wef@3**	**wef@5**	**wef@10**
H_1: 2	H_{11}: 4	H_{16}: 6	H_{16}: 6	H_1: 8	H_{11}: 20	H_{16}: 28	H_{16}: 48
H_2: 2	H_{12}: 4	H_{29}: 6	H_{29}: 6	H_2: 8	H_{12}: 20	H_{29}: 28	H_{29}: 48
H_5: 2	H_{16}: 4	$\boldsymbol{PG_{T2}}$: 5,07	$\boldsymbol{PG_{T2}}$: 5,17	H_5: 8	H_{16}: 20	$\boldsymbol{PG_{T2}}$: 30,8	$\boldsymbol{PG_{T2}}$: 55,17
H_9: 2	H_{17}: 4	H_2: 4	H_1: 4	H_9: 8	H_{17}: 20	H_{11}: 32	H_{11}: 62
H_{10}: 2	H_{18}: 4	H_5: 4	H_2: 4	H_{10}: 8	H_{18}: 20	H_{12}: 32	H_{12}: 62
P3 – replace – (28 versions)							
acc@1	**acc@3**	**acc@5**	**acc@10**	**wef@1**	**wef@3**	**wef@5**	**wef@10**
H_{16}: 6	$\boldsymbol{PG_{T2}}$: 12,3	$\boldsymbol{PG_{T2}}$: 13,93	H_{29}: 21	H_{16}: 22	AG_{T2}: 84	$\boldsymbol{PG_{T2}}$: 84,93	H_{29}: 132
H_{29}: 6	H_{16}: 12	H_7: 13	$\boldsymbol{PG_{T2}}$: 20,2	H_{29}: 22	$\boldsymbol{PG_{T2}}$: 56,6	H_{16}: 87	H_{16}: 133
$\boldsymbol{PG_{T2}}$: 5,63	H_{29}: 12	H_{16}: 13	H_{16}: 20	$\boldsymbol{PG_{T2}}$: 22,37	H_1: 63	H_{29}: 87	$\boldsymbol{PG_{T2}}$: 133,77
H_1: 5	H_2: 10	H_{29}: 13	H_{24}: 20	H_1: 23	H_{10}: 63	H_2: 97	H_{24}: 149
H_2: 5	H_5: 10	H_2: 10	H_{17}: 18	H_2: 23	H_{11}: 61	H_5: 97	H_{17}: 155
P4 – schedule – (7 versions)							
acc@1	**acc@3**	**acc@5**	**acc@10**	**wef@1**	**wef@3**	**wef@5**	**wef@10**
$\boldsymbol{PG_{T2}}$: 0,33	H_5: 2	$\boldsymbol{PG_{T2}}$: 3,6	$\boldsymbol{PG_{T2}}$: 4,03	$\boldsymbol{PG_{T2}}$: 6,67	H_{11}: 17	H_{11}: 26	$\boldsymbol{PG_{T2}}$: 41,53
AG_{T2}: 0	H_{11}: 2	H_{11}: 3	AG_{T2}: 3,33	AG_{T2}: 7	H_5: 18	$\boldsymbol{PG_{T2}}$: 26,23	H_{11}: 46
H_1: 0	$\boldsymbol{PG_{T2}}$: 1,03	H_{16}: 3	H_5: 3	H_1: 7	$\boldsymbol{PG_{T2}}$: 18,7	H_5: 28	H_{16}: 48
H_2: 0	H_1: 1	H_{29}: 3	H_{11}: 3	H_2: 7	H_7: 19	H_{16}: 28	H_{29}: 48
H_3: 0	H_2: 1	AG_{T2}: 2	H_{13}: 3	H_3: 7	H_8: 19	H_{29}: 28	H_5: 49
P5 – schedule2 – (9 versions)							
acc@1	**acc@3**	**acc@5**	**acc@10**	**wef@1**	**wef@3**	**wef@5**	**wef@10**
$\boldsymbol{PG_{T2}}$: 0	$\boldsymbol{PG_{T2}}$: 0,5	$\boldsymbol{PG_{T2}}$: 0,83	H_3: 3	$\boldsymbol{PG_{T2}}$: 9	$\boldsymbol{PG_{T2}}$: 26,4	$\boldsymbol{PG_{T2}}$: 42,83	$\boldsymbol{PG_{T2}}$: 82,2
AG_{T2}: 0	AG_{T2}: 0	AG_{T2}: 0	H_{20}: 3	AG_{T2}: 9	AG_{T2}: 27	AG_{T2}: 45	H_3: 83
H_1: 0	H_1: 0	H_1: 0	AG_{T2}: 2	H_1: 9	H_1: 27	H_1: 45	H_{20}: 83
H_2: 0	H_2: 0	H_2: 0	$\boldsymbol{PG_{T2}}$: 1,3	H_2: 9	H_2: 27	H_2: 45	AG_{T2}: 86
H_3: 0	H_3: 0	H_3: 0	H_4: 1	H_3: 9	H_3: 27	H_3: 45	H_{16}: 86
P6 – tcas – (30 versions)							
acc@1	**acc@3**	**acc@5**	**acc@10**	**wef@1**	**wef@3**	**wef@5**	**wef@10**
H_7: 2	H_{16}: 7	H_{16}: 7	$\boldsymbol{PG_{T2}}$: 12,47	H_7: 28	H_7: 76	H_{16}: 122	H_{16}: 219
$\boldsymbol{PG_{T2}}$: 0	H_{29}: 7	H_{29}: 7	H_1: 11	$\boldsymbol{PG_{T2}}$: 30	H_{16}: 76	H_{29}: 122	H_{29}: 219
AG_{T2}: 0	AG_{T2}: 6	AG_{T2}: 6	H_2: 11	AG_{T2}: 30	H_{29}: 76	H_7: 124	H_7: 221
H_1: 0	H_7: 6	H_7: 6	H_4: 11	H_1: 30	H_{11}: 80	H_{11}: 130	H_{11}: 227
H_2: 0	H_1: 5	$\boldsymbol{PG_{T2}}$: 5,9	H_5: 11	H_2: 30	H_{17}: 80	H_{17}: 130	H_{17}: 227
P7 – tot_info – (23 versions)							
acc@1	**acc@3**	**acc@5**	**acc@10**	**wef@1**	**wef@3**	**wef@5**	**wef@10**
H_8: 1	$\boldsymbol{PG_{T2}}$: 1,63	H_{16}: 6	H_{16}: 10	H_8: 18	H_8: 54	$\boldsymbol{PG_{T2}}$: 83,7	H_{16}: 141
$\boldsymbol{PG_{T2}}$: 0,4	H_1: 1	H_{29}: 6	H_{29}: 10	$\boldsymbol{PG_{T2}}$: 18,6	$\boldsymbol{PG_{T2}}$: 54,47	H_{16}: 85	H_{29}: 141
AG_{T2}: 0	H_2: 1	$\boldsymbol{PG_{T2}}$: 5,3	$\boldsymbol{PG_{T2}}$: 8,9	AG_{T2}: 19	H_1: 56	H_{29}: 85	$\boldsymbol{PG_{T2}}$: 142,97
H_1: 0	H_5: 1	H_1: 5	H_7: 8	H_1: 19	H_2: 56	H_1: 87	H_7: 151
H_2: 0	H_8: 1	H_2: 5	H_1: 6	H_2: 19	H_5: 56	H_2: 87	H_{12}: 152

os testes para os métodos de composição configurados com os conjuntos de terminais $T1$ e $T2$ respectivamente. A coluna "*p-valor Wilcoxon*" representa os p-valores obtidos pelos

Table 3.13: *Teste estatístico de Wilcoxon e de $\hat{A}_{12}$ Vargha e Delaney com AG_{T1} (A_1) e PG_{T1} (A_2)*

Programa	Conjunto de Implantação	*p-valor* Wilcoxon	*V-D* $\hat{A}_{12}$	Magnitude
P1 – printtokens	1	**2,86** $\times$ $\mathbf{10^{-11}}$	0,97	▲
	2	**1,46** $\times$ $\mathbf{10^{-12}}$	0,98	▲
	3	**1,27** $\times$ $\mathbf{10^{-8}}$	0,12	▼
P2 – printtokens2	1	**8,37** $\times$ $\mathbf{10^{-5}}$	0,20	▼
	2	**3,17** $\times$ $\mathbf{10^{-2}}$	0,35	▽
	3	**4,07** $\times$ $\mathbf{10^{-8}}$	0,09	▼
P3 – replace	1	**2,54** $\times$ $\mathbf{10^{-8}}$	0,08	▼
	2	**1,56** $\times$ $\mathbf{10^{-4}}$	0,22	▼
	3	**2,31** $\times$ $\mathbf{10^{-2}}$	0,33	▽
P4 – schedule	1	**4,68** $\times$ $\mathbf{10^{-10}}$	0,93	▲
	2	**1,49** $\times$ $\mathbf{10^{-8}}$	0,90	▲
	3	**4,14** $\times$ $\mathbf{10^{-7}}$	0,87	▲
P5 – schedule2	1	**8,63** $\times$ $\mathbf{10^{-12}}$	1,00	▲
	2	0,67	0,47	◊
	3	**2,21** $\times$ $\mathbf{10^{-11}}$	1,00	▲
P6 – tcas	1	**1,04** $\times$ $\mathbf{10^{-11}}$	1,00	▲
	2	**6,49** $\times$ $\mathbf{10^{-12}}$	1,00	▲
	3	**2,10** $\times$ $\mathbf{10^{-6}}$	0,85	▲
P7 – tot_info	1	**4,71** $\times$ $\mathbf{10^{-11}}$	0,01	▼
	2	0,86	0,51	◊
	3	**1,09** $\times$ $\mathbf{10^{-5}}$	0,83	▲

testes de *Wilcoxon*, com valores destacados em negrito quando o nível de significância é acima do limite de 95% (p-valores menores que $0,05$). Assim, o teste de *Wilcoxon* indica que as diferenças entre PG_{T1} e AG_{T1} são estatisticamente significativas para todos, exceto dois, conjuntos de implantação em diferentes programas enquanto as diferenças entre PG_{T2} e AG_{T2} não são significativas em apenas cinco amostras em programas distintos.

Os resultados do teste estatístico $\hat{A}_{12}$ de *Vargha e Delaney* estão resumidos nas Tabela 3.13 e 3.14, em que o algoritmo A_1 corresponde ao AG_{T1} e AG_{T2} e o algoritmo A_2 corresponde ao PG_{T1} e PG_{T2} respectivamente. Assim, os valores de $\hat{A}_{12}$ correspondem a probabilidade de o método baseado em Programação Genética obter resultados melhores. A coluna "*V-D* $\hat{A}_{12}$" corresponde aos valores absolutos de $\hat{A}_{12}$, enquanto a coluna "Magnitude" representa uma referencia para magnitude da diferença de A_2 em relação ao A_1, em que "▲" indica que os resultados de A_2 são significantemente superiores, "▼" para significantemente inferiores, "△" para insignificantemente superiores, "▽" para insignificantemente inferiores e "◊" indica diferenças insignificantes.

Ao observar a Tabela 3.13, nota-se que não foi verificada diferença estatística em dois das 21 comparações. Nestes casos a magnitude aferida foi insignificante ("◊"), confirmando o teste de Wilcoxon. Estes resultados conferem maior confiança aos resultados da análise de *expense* médio. Observando o programa P1 (*printtokens*), houve magnitude significante em todas as três comparações, sendo valores próximos de um em duas, indi-

Tabela 3.14: *Teste estatístico de Wilcoxon e de $\hat{A}_{12}$ Vargha e Delaney com AG_{T2} (A_1) e PG_{T2} (A_2)*

Programa	Conjunto de Implantação	*p-valor* Wilcoxon	*V-D* $\hat{A}_{12}$	Magnitude
P1 – printtokens	1	0,23	0,58	◊
	2	$\mathbf{6{,}37 \times 10^{-9}}$	0,90	▲
	3	$\mathbf{1{,}62 \times 10^{-7}}$	0,89	▲
P2 – printtokens2	1	0,06	0,64	△
	2	$\mathbf{5{,}71 \times 10^{-13}}$	1,00	▲
	3	$\mathbf{1{,}88 \times 10^{-5}}$	0,8	▲
P3 – replace	1	$\mathbf{3{,}16 \times 10^{-12}}$	1,00	▲
	2	$\mathbf{4{,}02 \times 10^{-10}}$	0,93	▲
	3	$\mathbf{8{,}24 \times 10^{-7}}$	0,87	▲
P4 – schedule	1	0,06	0,37	▽
	2	$\mathbf{1{,}49 \times 10^{-11}}$	1,00	▲
	3	$\mathbf{1{,}31 \times 10^{-8}}$	0,10	▼
P5 – schedule2	1	$\mathbf{3{,}35 \times 10^{-11}}$	0,03	▼
	2	0,16	0,40	◊
	3	$\mathbf{2{,}05 \times 10^{-5}}$	0,80	▲
P6 – tcas	1	$\mathbf{9{,}57 \times 10^{-13}}$	1,00	▲
	2	$\mathbf{8{,}81 \times 10^{-12}}$	1,00	▲
	3	$\mathbf{1{,}73 \times 10^{-7}}$	0,87	▲
P7 – tot_info	1	$\mathbf{2{,}31 \times 10^{-8}}$	0,91	▲
	2	$\mathbf{4{,}60 \times 10^{-13}}$	1,00	▲
	3	0,18	0,60	◊

cando superioridade para PG_{T1}, e valor próximo de zero na outra, indicando superioridade para AG_{T1}. Portanto, apesar de valores médios menores, PG_{T1} foi significativamente superior em relação ao AG_{T1} em dois dos três conjuntos de implantação.

Da mesma forma, pode-se dizer que PG_{T1} superou AG nos programas P4 (*schedule*), P5 (*schedule2*) e P6 (*tcas*), com grande vantagem nos últimos dois para todos os conjuntos de implantação. Entretanto, AG_{T1} foi estatisticamente superior em P2 (*printtokens2*) e P3 (*replace*) e houve empate em P7 (*tot_inf*).

Em relação à comparação estatística dos métodos de composição de heurśiticas configurados com o conjunto de terminais $T1$ (Tabela 3.14), foram observadas diferenças estatisticamente significativas entre PG_{T2} e AG_{T2} em todos os programas, sendo os resultados da PG_{T2} significativamente superiores nos programas P2 (*printtokens2*), P3 (*replace*), P6 (*tcas*) e P7 (*tot_inf*), todos os programas para os quais a PG_{T2} obteve melhores resultados na proporção de linhas inspecionadas para localizar todos os defeitos. Além disso, PG_{T2} teve diferença estatística superior significativa em P1, que perdeu para AG_{T2} em valores médios. Nos programas em que as métricas de AG_{T2} tinham uma melhor média de proporção de linhas inspecionadas, a PG_{T2} tinha um valor superior estatisticamente significativo em pelo menos um conjunto de implantação.

3.3.5 Ameaças à Validade

As ameaças à *validade interna*, ou seja, ocorrência de resultados ao acaso, foram tratadas usando: métodos de referência e medidas de avaliação usadas em estudos anteriores; execuções múltiplas para reduzir a natureza estocástica dos algoritmos; e estruturas de software livre existentes que foram usadas em diversas aplicações.

Para lidar com a *validade externa*, isto é, a possibilidade de generalização dos resultados, um *benchmark* comum em muitos contextos relacionados à engenharia de software foi empregado; entretanto, avaliações em grande escala e em outras linguagens de programação são necessárias. Muitos trabalhos recentes utilizaram o conjunto de programas Siemens como *benchmark* no problema de localização de defeitos (tais como [30, 39, 43]), de modo a permitir comparações com pesquisas anteriores e posteriores. Isto permite a construção de uma base de evidências e incentiva experimentos repetíveis que promoverão avaliações baseadas na indústria.

Ameaças à *validade de construção* abordam o quão bem as medições estão realmente correlacionadas com o que elas afirmam fazer. Foram utilizadas métricas de avaliação absolutas e relativas, e suas medidas foram tomadas similarmente a estudos anteriores, para que estas possam comunicar realisticamente a capacidade de localização de defeitos.

Estes resultados indicam que o espaço de busca mais amplo disponibilizado ao método baseado em Programação Genética permite melhor adaptabilidade contra heurísticas independentes de programa e o espaço de busca limitado a equações lineares.

3.4 Considerações Finais

A localização de defeitos para muitos projetos é uma tarefa onerosa e demorada. Alguns trabalhos, tais como [26, 1, 51, 55], abordaram o problema e propuseram heurísticas para automatizar ou auxiliar o processo.

Neste capítulo, uma meta-heurística evolucionária, Programação Genética, foi aplicada para buscar composições não-lineares de heurísticas que derivam medidas de suspeita de elementos de código potencialmente defeituosos. A proposta utiliza uma abordagem orientada a projetos com o objetivo de encontrar fórmulas melhor adaptadas ao problema de localização de defeitos no âmbito de cada programa.

Foram utilizadas duas configurações distintas de conjuntos de heurísticas para a composição pela proposta. Os conjuntos são integrados por heurísticas individuais baseadas em cobertura de código (Tabela 3.2), métricas de associação (Tabela 3.1) e dados do espectro de cobertura.

Heurísticas de localização de defeitos consolidadas na literatura foram utilizadas para comparação, além do método de Wang *et al.* [51], baseado em Algoritmo Genético, que busca composições lineares de outras heurísticas. Assim, em média, o método proposto foi capaz de encontrar soluções mais complexas e melhor adaptadas aos programas treinados em relação ao método de composição linear e as heurísticas tradicionais.

Apesar do desempenho médio superior, o método proposto não foi o melhor para todos os programas. Devido ao reduzido número de versões destes programas, o algoritmo encontrou soluções extremamente especializadas ao conjunto de treinamento e que, consequentemente, não são suficientemente generalizáveis para a fase de implantação. Assim, constata-se que devido a característica de permitir soluções mais complexas, e portanto mais especializadas, o método baseado em Programação Genética é mais dependente da qualidade do conjunto de treinamento em relação ao metódo de Wang *et al.* [51].

A proposta resultou em publicações no 8^o Workshop de Engenharia de Software Baseada em Busca (*WESB 2017*) [8] e no *26th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning* (*ESANN 2018*).

O método proposto permite fórmulas mais complexas e melhor adaptação a um conjunto de versões de um mesmo programa: o treinamento baseado em projeto resulta em fórmulas customizadas ao conjunto de potenciais defeitos de um programa específico.

O Capítulo 4 expande o método apresentado para outras fontes de dados ao gerar novas heurísticas a partir de variáveis básicas do espectro de mutação utilizando a mesma metodologia de treinamento evolucionário baseado em projetos apresentada até aqui.

CAPÍTULO 4

Geração de Heurísticas Baseadas em Espectro de Mutação

Existem técnicas que expandem as heurísticas baseadas em espectro ao utilizar dados de outra natureza que adicionam informação à cobertura dos testes. Conforme apresentado no Capítulo 2, algumas pesquisas aplicaram dados da análise de mutantes ao problema de localização de defeitos [42, 35, 43], compondo assim as técnicas de localização de defeitos baseadas em espectro de mutação. As heurísticas de **Localização de Defeitos Baseada em Mutação** (**MBFL**, do Inglês: *Mutation-based Fault Localization*) são análogas às heurísticas SBFL (baseadas em espectro de cobertura) ao calcular a suspeita de cada elemento de código conter o defeito.

Dado um programa defeituoso, mutantes são gerados para cada elemento de código coberto por algum teste que revela algum defeito. Em seguida os programas mutantes são executados pelo conjunto de teste e os dados do espectro de mutação são coletados ao aferir quais testes mataram quais mutantes. De posse do espectro de mutação, uma heurística MBFL é utilizada para calcular um escore de suspeita para cada elemento de código do programa e organizar uma lista para guiar a investigação do código pelo defeito, do elemento mais suspeito para o de menor suspeita.

A suspeita de um elemento de código é obtida através da suspeita de seus mutantes, aqueles cujo ponto de mutação situa-se no elemento. Assim, a suspeita do elemento será o valor da maior suspeita de seus mutantes calculada pela heurística MBLF. Estas técnicas baseiam-se no princípio de que os testes que matam mutantes tem poder de diagnóstico, de modo que quanto mais um mutante afeta os testes negativos e menos afeta os testes positivos, maior será a suspeita do próprio comando.

No contexto de resultados competitivos obtidos na literatura com técnicas de localização de defeitos baseadas em espectro de mutação projetadas manualmente (por humanos) (*e.g.* [42] *inter alia*) e as conclusões promissoras da utilização de Programação Genética para geração de heurísticas baseadas em espectro de cobertura ([55] *inter alia*), este capítulo descreve o projeto e análise de uma abordagem evolucionária para construir heurísticas MBFL orientadas a programas de forma automática.

Este trabalho aborda diretamente as lacunas delineadas em [56] e culminou na publicação de um artigo no *2018 IEEE Congress on Evolutionary Computation* (IEEE CEC 2018) [10]. Este capítulo analisa a eficácia de heurísticas baseadas em variáveis de mutação em comparação às baseadas em variáveis de cobertura para a abordagem evolucionária, especificamente o algoritmo de Programação Genética.

O texto está estruturado da seguinte maneira: A Seção 4.1 descreve a abordagem proposta; A Seção 4.2 apresenta o esquema de experimentação, parâmetros e os métodos de avaliação empregados; A Seção 4.3 apresenta e discute os resultados; e a Seção 4.4 conclui o capítulo.

4.1 Método Proposto: Aplicação de Programação Genética à Heurísticas MBFL

Retomando o contexto da aplicação do algoritmo Programação Genética ao problema de localização de defeitos, cada indivíduo representa a fórmula matemática de uma heurística para calcular suspeita de elementos de código. A população é um conjunto de soluções que se desenvolve conforme o algoritmo, visando alcançar fórmulas que sejam melhores em aferir o quão suspeito um elemento de programa é. Assim, o espaço de busca constitui-se do conjunto de todas as fórmulas válidas possíveis composta pelas funções e terminais definidas como parâmetros iniciais do algorítimo. Observa-se que equações não lineares são comumente obtidas neste processo.

Em [55] Programação Genética foi utilizada para sintetizar heurísticas SBFL a partir das quatro variáveis básicas de cobertura de código $\{c_{ef}, c_{ep}, c_{nf}$ e $c_{np}\}$ e a constante inteira 1 (um) e o conjunto de funções matemáticas composto pelas quatro operações básicas e raiz quadrada. O objetivo é a minimização da média da métrica *Expense* (a posição do comando defeituoso em relação ao número de comandos), em relação a n versões defeituosas, conforme representado na Equação 4-1. A aptidão do indivíduo é calculada a partir das lista de elementos de programa ordenadas por suspeitas, obtidas por sua heurística correspondente.

$$expense\ médio = \frac{\sum_{i=1}^{n} expense(i)}{n} \tag{4-1}$$

Na presente abordagem, o algoritmo de Programação Genética foi configurado com as quatro variáveis básicas de mutação $\{m_{kf}, m_{kp}, m_{nf}$ e $m_{np}\}$ e a constante 1 (análogo à configuração de [55]) como terminais, e o conjunto de funções definido como as quatro operações básicas da matemática (adição, divisão, subtração e divisão protegida,

isto é, retorna um quando há divisão por zero) e raiz quadrada protegida (a raiz quadrada do módulo de um número).

Já que um comando costuma possuir mais de um mutante, a suspeita máxima de seus mutantes é considerada a suspeita do elemento de código. Como as heurísticas MBFL apresentam desempenho ruim em defeitos que envolvem comandos que não podem sofrer mutação [45], as instruções sem mutantes recebem a suspeita mais baixa. Após o cálculo da suspeita, é construído uma lista de elementos ordenando-os em ordem não crescente de suspeita. Depois de criadas as classificações de suspeita, calcula-se a aptidão do indivíduo utilizando-se a Fórmula 4-1, similarmente à função usada por Yoo [55].

Similarmente ao método do Capítulo 3, a proposta é estruturada em duas etapas: treinamento e implantação. Os aspectos inovadores do método estão ligados à investigação conjunta de:

- especialização de heurísticas a certos contextos;
- aplicação de espectro de mutação para geração automática de heurísticas, ou seja, sinais (fontes de dados) além de cobertura de código;
- comparação da eficácia de espectro de cobertura e espectro de mutação no contexto de abordagens evolucionárias, isto é, a análise de heurísticas em termos de desempenho absoluto (*e.g. expense* médio) e relativo (*e.g. precision* e *wasted effort*);
- análise da qualidade do espectro de mutação e seu impacto na localização de defeitos.

4.2 Experimentos

Assim como no Capítulo 3, foi aplicada validação cruzada tripla *three-fold cross validation* para avaliar o método proposto, conforme o processo de experimentação representado na Figura 4.1. Inicialmente, os conjuntos de versões do mesmo programa foram divididos aleatoriamente em três subconjuntos cada. Em cada etapa da validação cruzada, dois subconjuntos compuseram o conjunto de treinamento, enquanto o restante foi utilizado para avaliar a heurística treinada, simulando a fase de implantação. O conjunto de versões do programa foi subdividido somente uma vez, sendo que a cada etapa da validação cruzada os subconjuntos foram permutados de modo que cada um pudesse ser utilizado como conjunto de treinamento durante uma etapa. Observa-se que a interseção dos conjuntos de treinamento e implantação para a mesma rodada da validação cruzada tripla é sempre vazia.

No esquema da Figura 4.1 k representa a etapa da validação cruzada tripla e i representa a rodada de experimentos. Em cada rodada o método de busca é executado para o conjunto de treinamento e ao final, a melhor solução encontrada é utilizada para

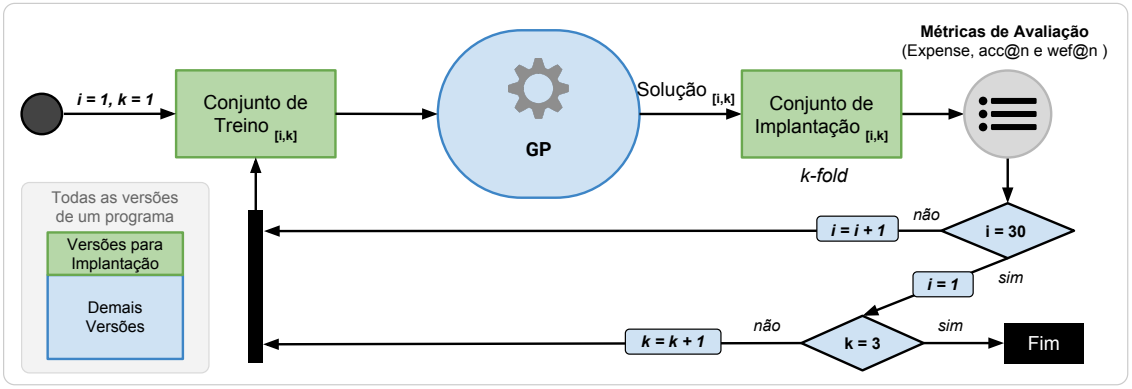

Figura 4.1: *Esquema de experimentação utilizado para geração de heurísticas baseadas em espectro de mutação.*

localizar os defeitos dos programas do conjunto de implantação e os resultados são coletados conforme as métricas de avaliação utilizadas. Ressalta-se que todos os resultados apresentados são coletados do conjunto de implantação, que não exerce influência sobre a busca. Devido ao caráter estocástico dos métodos, especificou-se como 30 o número de rodadas experimentais com o objetivo de reduzir os efeitos estocásticos inerentes ao processo. Assim, para cada etapa k, são executadas 30 rodadas do experimento.

O restante desta seção apresenta alguns detalhes importantes sobre a configuração dos experimentos conduzidos. A Subseção 4.2.1 elenca os programas utilizados e suas características; A Subseção 4.2.2 apresenta os parâmetros utilizados nas meta-heurísticas; A Subseção 4.2.3 descreve os métodos utilizados para comparação ao avaliar o método proposto; e a Subseção 4.2.4 apresenta e resume as métricas de avaliação empregadas.

4.2.1 Os *Benchmarks*

Os programas utilizados nos experimentos foram obtidos a partir dos *Benchmarks*: *Siemens Suite* [23], amplamente utilizado pela comunidade acadêmica, incluindo publicações recentes [42, 51, 32, 16, 56]; *Codeflaws* [48], uma coleção de programas da base de dados online *Codeforces*, uma plataforma de maratonas de programação; e o *Defeitos4J* [27], que é um conjunto de projetos robustos reais abertos com defeitos reais. Os benchmarks ajudam a construir experimentos replicáveis e possibilitam experimentos para posteriormente promover avaliações baseadas na indústria.

Os programas do *Siemens Suite* foram obtidos do repositório SIR (*Software-artifact Infrastructure Repository*) [12], que é um projeto criado para distribuir artefatos relacionados a software para experimentos com análise de programas, teste de software e educação. Os programas selecionados foram *print_tokens2*, *schedule* e *tcas*, em linguagem C. Os programas Siemens foram utilizados com um conjunto reduzido de casos de teste disponibilizados pelo SIR.

Do Benchmark *Codeflaws*, as submissões para o problema *475-A* foram selecionadas. Neste caso, cada submissão de solução para o problema é tratada como uma versão do mesmo programa. Cada submissão representa um programa na linguagem C enviada por um aluno real, portanto são programas reais com defeitos reais.

Em relação aos experimentos com código de defeito único (*single-bug*), como Siemens e Codeflaws, algumas versões não foram usadas porque: a falha estava em uma linha não instrumentada (por exemplo, erro de declaração variável); o conjunto de testes não revelou o defeito; ou o defeito não estava localizado em uma única linha de código, isto é, mais de um elemento era responsável pela falha.

O espectro para os programas foi gerado com as ferramentas *lcov*, *gcov* (uma ferramenta de cobertura de testes padrão do pacote GNU) e um programa próprio. Para a geração dos mutantes de programas C, foi utilizada a ferramenta *Proteum/IM 2,0* [34].

Defects4J é um framework que oferece uma base de dados e uma estrutura de projetos com defeitos reais na linguagem Java que visa a disponibilizar dados para estudos replicáveis em pesquisas em teste de software. Com o objetivo de avaliar como a disponibilidade de mutantes impacta na qualidade da localização de defeitos baseada em mutações, foram coletados dados de cobertura e mutação do programa *Math*, um projeto real multi-bug do Defects4J, usando o repositório criado e disponibilizado por Pearson *et al.* [45].

Detalhes do conjunto de programas utilizado nos experimentos são apresentados na Tabela 4.1.

Tabela 4.1: *Conjunto de programas usado nos experimentos*

Program	Inst. LOC	Faulty Versions	Test Set	Mutants
475A	38,5	8	35	3257,5
print_tokens2	200	10	265	4679,5
schedule	152	7	387	2240
tcas	65	26	354	2872
Math	2022,4	77	212,2	8950,7

Outra ferramenta utilizada no projeto, foi o framework para a linguagem Python *DEAP*, que permite a abstração dos operadores básicos necessários para a implementação de algoritmos evolutivos e programação paralela concorrente. Para a implementação da meta-heurística de Programação Genética, o DEAP foi usado com configurações básicas. Assim, apenas os parâmetros necessários foram definidos, usando as configurações padrão inalteradas sempre quando possível.

4.2.2 Parâmetros das Meta-heurísticas

Os parâmetros aplicados para o algoritmo de Programação Genética foram definidos empiricamente como: número de gerações: 1000; tamanho da população: 100 indivíduos; taxa de cruzamento: 0,85; taxa de mutação: 0,07; taxa de reprodução: 0,08; profundidade máxima de cruzamento: 20; profundidade inicial máxima: 10; profundidade inicial mínima: 2; método de seleção: roleta; operador de cruzamento: cruzamento de um ponto (troca de sub-árvores selecionadas aleatoriamente entre pais); operador de mutação: mutação uniforme (substituir uma sub-árvore no indivíduo por uma nova gerada aleatoriamente); seleção de próxima geração: Descendentes com elitismo (persistência do melhor indivíduo da população dos pais).

4.2.3 Avaliação do Método

O conjunto de heurísticas de comparação foi selecionado a partir de pesquisas recentes em SBFL e MBFL [45, 16] e estão dentre as mais populares no levantamento de Wong *et al.* [53]. Foram selecionadas as heurísticas SBFL Tarantula [26, 42], Ochiai [1, 16], OP^2 [40, 16], Barinel [2] e DStar [52] além de de suas homólogas em MBFL. Ainda, a abordagem baseada em PG de Yoo foi adaptada como uma versão análoga ao método proposto, no âmbito de SBFL. Deste modo tem-se uma referência evolucionária para comparação. Ao todo, 11 heurísticas de comparação foram selecionadas.

4.2.4 Métricas de Avaliação

Para avaliar o desempenho das heurísticas SBFL e MBFL em relação à quantidade absoluta e relativa de elementos de código investigados até encontrar os defeitos, foram aplicadas as seguintes métricas de avaliação, assim como na análise do Capítulo 3 (ver Subseção 3.2.4):

- ***Expense* Médio**: refere-se à média da posição relativa do comando defeituoso quando normalizada pelo número total de comandos no programa [55] em relação a todas as versões defeituosas.
- ***Precision*** (**acc**@n): o número de defeitos dentre as primeiras *n* posições da lista de elementos de código ordenada por suspeitas, obtida pela técnica avaliada. Valores maiores são melhores. Foram utilizados 1, 3 e 5 para *n*.
- ***Wasted Effort*** (**wef**@n): a quantidade de trabalho desperdiçado ao analisar cada elemento de código do programa que não é responsável pelo defeito antes de localizá-lo (valores menores são melhores). Também foram utilizados 1, 3 e 5 para *n*.

- ***Rank* Médio**: a posição média de cada técnica segundo seus resultados em cada programa para alguma outra métrica de avaliação. Para uma medida de avaliação, o *rank* médio de uma heurística H é a média de suas posições (primeiro, segundo, terceiro, etc.) calculadas para todos os programas. Se a medida para a heurística H é igual a outra, ambas tem a mesma posição relativa: o pior caso (*e.g.* se três heurísticas têm a segunda melhor medida de $wef@3$, então a todas é atribuída a quarta posição no *rank*).

4.3 Análise

As questões a serem respondidas (questões de pesquisa) estão ligadas: à *capacidade geral para localizar defeitos* e ao *número de elementos de programa investigados até encontrar os defeituosos*. Em outras palavras, as questões são, respectivamente,

- *Quão eficazes são as soluções obtidas por Programação Genética baseadas em variáveis de mutação em termos relativos?* **(QP1)**; e
- *Quão eficazes são as soluções obtidas por Programação Genética baseadas em variáveis de mutação em termos absolutos?* **(QP2)**.

Em adição, Pearson *et al.* recentemente constataram que heurísticas MBFL tem desempenho ruim em defeitos reais em relação a defeitos artificiais [45], mas tal pesquisa não considerou a qualidade dos dados de mutação utilizados. Assim, este capítulo também investiga *se o número de mutantes disponíveis é um fator relevante para o desempenho de heurísticas MBFL construídas evolucionariamente por Programação Genética*, ou seja:

- *Como a qualidade do espectro de mutação impacta sobre a capacidade de localizar defeitos?* **(QP3)**

Esta seção apresenta os resultados obtidos pelos experimentos e os analisa e compara em relação às heurísticas apresentadas na Subseção 4.2.3. A Subseção 4.3.1 analisa a eficácia das heurísticas MBFL treinadas pelo método proposto. A Subseção 4.3.2 analisa a qualidade do espectro de mutação e seu impacto ao método. A Subseção 4.3.3 apresenta uma análise estatística dos resultados. A Subseção 4.3.4 resume as ameaças a validade do trabalho.

4.3.1 Eficácia de Heurísticas MBFL Treinadas Evolucionariamente

Para responder às questões de pesquisa QP1 e QP2 (*quão eficazes são as soluções obtidas por Programação Genética baseadas em variáveis de mutação em termos relativos e absolutos*, respectivamente) foram analisados:

1. as medidas de avaliação para o conjunto de programas avaliados;
2. as classificações médias (*rank* médio) das heurísticas conforme todas as medidas de avaliação;
3. a comparação de técnicas de localização de defeitos em programas individuais.

Destes três itens, os resultados obtidos de *expense* médio se aplicam a QP1 e os resultados de *precision* e *wasted effort* tratam QP2,

Os resultados são apresentados nas Tabelas 4.2, 4.3 e 4.4. No restante do texto, a técnica de construção evolucionária de heurísticas MBFL será referida como "*PG-mut*", enquanto técnica análoga de construção de heurísticas SBFL será citada como "*PG-cov*". Ainda, as heurísticas tradicionais Tarantula, Ochiai, OP^2, Barinel e DStar também são apresentadas utilizando-se os sufixos "*-cod*" e "*-mut*" em referencia à utilização de variáveis de cobertura e mutação em SBFL e MBFL, respectivamente (*e.g.* "*TAR-cov*" e "*TAR-mut*"). Conforme QP1 e QP2, o foco é a eficácia das soluções baseadas em variáveis de mutação obtidas do processo evolucionário do algoritmo de Programação Genética, isto é, o qual eficaz *PG-mut* é em relação às outras técnicas da literatura, especialmente em relação à *PG-cov*, sua contraparte evolucionária baseada em variáveis de cobertura.

Tabela 4.2: *Medidas de avaliação para o conjunto de programas avaliados*

Método	*Expense* médio	acc@n			wef@n		
		1	**3**	**5**	**1**	**3**	**5**
TAR-cov	47,14%	1	8	10	50	137	220
TAR-mut	43,98%	2	8	14	49	136	211
OCH-cov	46,48%	2	9	14	49	135	212
OCH-mut	53,60%	3	6	10	48	138	221
OP2-cov	40,27%	3	11	15	48	129	202
OP2-mut	59,86%	0	2	7	51	149	238
BAR-cov	48,37%	1	8	10	50	137	220
BAR-mut	43,70%	2	8	14	49	136	211
DST-cov	48,02%	2	8	10	49	136	219
DST-mut	46,78%	1	7	12	50	139	219
PG-cov	**30,00%**	0,33	4,63	10,17	50,67	145,63	232,40
PG-mut	36,38%	**6,47**	**15,40**	**20,54**	**44,53**	**118,90**	**181,14**

Avaliação do Conjunto de Programas Completo

A Tabela 4.2 apresenta uma comparação geral das medidas de avaliação utilizadas. Estão apresentados os valores de *expense* médio, bem como *acc* e *wef* para o conjunto completo de programas selecionados para avaliação. Os melhores valores de cada métrica de avaliação estão destacados em negrito. Por exemplo, a quarta coluna da Tabela 4.2 (valores de *acc@3*) demonstra que *PG-mut* colocou o elemento defeituoso dentre os três de maior suspeita (três primeiras posições da lista de elementos de código ordenada por suspeitas) em média 15,40 vezes, contra 11,00 do segundo melhor resultado (*OP2-cov*).

Tabela 4.3: *Rank médio de expense médio, acc@n e wef@n*

***Expense* médio**	**acc@1**	**acc@3**	**acc@5**	**wef@1**	**wef@3**	**wef@5**
PG-cov: 2,75	**PG-mut: 4,5**	**PG-mut: 1,5**	**PG-mut: 3**	**PG-mut: 4,5**	**PG-mut: 1,5**	**PG-mut: 1,75**
OP2-cov: 4	OP2-cov: 8,25	OP2-cov: 6,5	OP2-cov: 6	OP2-cov: 8,25	OP2-cov: 6,5	OP2-cov: 5,25
PG-mut: 4,5	**PG-cov: 9,5**	OCH-cov: 7,5	OCH-cov: 6,25	**PG-cov: 9,5**	OCH-cov: 7,5	OCH-cov: 6
OCH-cov: 4,75	OCH-mut: 9,5	TAR-cov: 8,5	**PG-cov: 7,5**	OCH-mut: 9,5	DST-cov: 8	DST-mut: 7,75
DST-cov: 5,75	OCH-cov: 9,75	BAR-cov: 8,5	TAR-cov: 8,75	OCH-cov: 9,75	TAR-cov: 8,5	TAR-mut: 8
TAR-cov: 6,5	DST-cov: 9,75	DST-cov: 8,5	BAR-cov: 8,75	DST-cov: 9,75	BAR-cov: 8,5	BAR-mut: 8
BAR-cov: 7,5	TAR-mut: 10	**PG-cov: 9,25**	DST-cov: 8,75	TAR-mut: 10	**PG-cov: 9,5**	DST-cov: 8,25
TAR-mut: 9	BAR-mut: 10	TAR-mut: 9,75	DST-mut: 8,75	BAR-mut: 10	TAR-mut: 9,75	OCH-mut: 8,5
BAR-mut: 9	TAR-cov: 10,5	BAR-mut: 9,75	TAR-mut: 9,25	TAR-cov: 10,5	BAR-mut: 9,75	TAR-cov: 8,75
DST-mut: 9,25	BAR-cov: 10,5	DST-mut: 10	BAR-mut: 9,25	BAR-cov: 10,5	OCH-mut: 10	BAR-cov: 8,75
OCH-mut: 9,75	DST-mut: 10,5	OCH-mut: 10,5	OCH-mut: 10	DST-mut: 10,5	DST-mut: 10,25	**PG-cov: 8,75**
OP2-mut: 10,5	OP2-mut: 12	OP2-mut: 12	OP2-mut: 10,75	OP2-mut: 12	OP2-mut: 12	OP2-mut: 10,25

Considerando que *PG-cov* e *PG-mut* são treinados para cada programa e seus resultados são calculados a partir de então, é esperado que superem os outros métodos. De fato, *PG-mut* superou os demais em todos os valores de *acc* e *wef*, enquanto foi o segundo melhor para o *expense* médio. Entretanto *PG-cov* foi superior na perspectiva relativa, pois obteve a melhor medida de *expense* médio. Especificamente para valores de *acc*, os resultados de *PG-mut* são muito melhores do que os das outras heurísticas, assim, foi a mais precisa na localização de defeitos sob a perspectiva absoluta.

Avaliação de *Rank* Médio

Para aferir o desempenho posicional relativo das técnicas, calculou-se a *rank* médio para o conjunto de todos os métodos avaliados em relação a cada métrica de avaliação. Para cada métrica (*expense* médio, $acc@1$, $acc@3$, *etc*), define-se o *rank* médio como a classificação média de uma técnica em relação aos programas avaliados. Se há empate entre medidas de dois métodos, todas são classificadas com a mesma posição no *rank*: o pior caso (*e.g.* se três técnicas estão empatados com a segunda melhor medida de $wef@5$, então a todas será atribuída a posição de classificação quatro em $wef@5$). Os *ranks* médios em todas as métricas de avaliação estão apresentados na Tabela 4.3, os valores de *PG-mut* e *PG-cov* estão destacados em negrito. Cada coluna é ordenada de modo que a técnica de melhor desempenho esteja no topo.

Observa-se que *PG-mut* apresentou medidas de *rank* médio superiores em quase todas as colunas, como mostrado na Tabela 4.3, isto é, foi a melhor técnica, dentre as avaliadas, para todas as medidas de *precision* e *wasted effort*. A exceção está em *expense* médio, apesar da terceira posição. Constantemente, *PG-cov* obteve o melhor desempenho relativo (*expense* médio). Este resultado é coerente com a avaliação acima mencionada de todos os programas como um todo.

Tabela 4.4: *Comparação dos métodos avaliados em programas individuais*

Prog	Technique	*Expense* médio	acc@n			wef@n		
			1	**3**	**5**	**1**	**3**	**5**
475-A	TAR-cov	61,21	0	0	1	8	24	39
	TAR-mut	86,07	0	0	1	8	24	38
	OCH-cov	69,00	0	0	1	8	24	39
	OCH-mut	82,12	0	0	1	8	24	38
	OP2-cov	69,00	0	0	1	8	24	39
	OP2-mut	82,12	0	0	1	8	24	38
	BAR-cov	69,10	0	0	1	8	24	39
	BAR-mut	84,09	0	0	1	8	24	38
	DST-cov	69,00	0	0	1	8	24	39
	DST-mut	82,12	0	0	1	8	24	38
	PG-cov	**55,51**	0	0	0	8	24	40
	PG-mut	75,73	0	**1,37**	**2,37**	8	**21,3**	**32,57**
print_tokens2	TAR-cov	36,21	1	3	3	9	23	37
	TAR-mut	45,35	0	0	0	10	30	50
	OCH-cov	31,68	**2**	**4**	4	**8**	**21**	33
	OCH-mut	69,40	0	0	0	10	30	50
	OP2-cov	**18,19**	**2**	**4**	4	**8**	**21**	33
	OP2-mut	73,35	0	0	0	10	30	50
	BAR-cov	36,21	1	3	3	9	23	37
	BAR-mut	45,35	0	0	0	10	30	50
	DST-cov	35,65	**2**	3	3	**8**	22	36
	DST-mut	59,25	0	0	2	10	30	47
	PG-cov	38,15	0	0,87	0,90	10	28,27	46,50
	PG-mut	33,94	1,67	3,83	**5,03**	8,33	21,27	**31,43**
schedule	TAR-cov	46,81	0	1	2	7	19	29
	TAR-mut	73,31	0	0	0	7	21	35
	OCH-cov	44,83	0	1	3	7	19	27
	OCH-mut	73,31	0	0	0	7	21	35
	OP2-cov	24,34	0	1	**4**	7	19	**25**
	OP2-mut	73,31	0	0	0	7	21	35
	BAR-cov	46,81	0	1	2	7	19	29
	BAR-mut	73,31	0	0	0	7	21	35
	DST-cov	46,43	0	1	2	7	19	29
	DST-mut	73,31	0	0	0	7	21	35
	PG-cov	**20,70**	0,33	0,77	2,43	6,67	19,37	28,90
	PG-mut	63,96	**1,53**	**1,73**	1,90	**5,47**	**16,10**	26,37
tcas	TAR-cov	47,10	0	4	4	26	71	115
	TAR-mut	22,60	2	8	**13**	24	61	**88**
	OCH-cov	45,68	0	4	6	26	71	113
	OCH-mut	33,43	3	6	9	23	63	98
	OP2-cov	44,20	1	6	6	25	65	105
	OP2-mut	44,20	0	2	6	26	74	115
	BAR-cov	47,10	0	4	4	26	71	115
	BAR-mut	22,66	2	8	**13**	24	61	**88**
	DST-cov	46,75	0	4	4	26	71	115
	DST-mut	23,96	1	7	9	25	64	99
	PG-cov	21,51	0	3	5	26	74	117
	PG-mut	**17,78**	**3,27**	**8,47**	11,17	**22,73**	**60,23**	90,77

Avaliação de Programas Individuais

Em uma visão mais detalhada dos experimentos, a Tabela 4.4 apresenta os resultados individuais para as métricas de avaliação (*expense* médio, *acc@n* e *wef@n*) de cada programa avaliado. A melhor técnica associada a cada medida de avaliação para cada programa está destacada em negrito. É notável que o desempenho do *PG-mut* é superior em cada medida relacionada ao programa *tcas*, que tem mais versões defeituosas (ou seja, melhor capacidade de treinamento). Nos outros programas *PG-mut* perde em *expense* médio, mas é o melhor em quase todos os valores de *acc* e *wef*.

Isso indica que *PG-mut* tem melhor eficácia, em relação a outros métodos, ao atribuir suspeitas maiores a elementos de código defeituosos. Ainda, conjectura-se que o baixo desempenho de *PG-mut* nos programas *475-A*, *print_tokens2* e *schedule* deve-se ao fato de que há poucas versões defeituosas disponíveis para os mesmos. Este comportamento ocorre similarmente para *PG-cov*, mas conforme observado na Tabela 4.4, a *PG-mut* é aparentemente mais sensível ao treinamento.

Em resumo, os resultados de *PG-mut* são consistentes quando avaliada no conjunto de programas completo, em programas individuais, bem como por *rank* médio: bem adaptados (segunda e terceira melhor) para a perspectiva relativa e o melhor técnica em todas as medidas de qualidade absoluta. Para a avaliação relativa, *PG-cov* apresentou bons resultados mostrando concordância com o trabalho anterior (*e.g.* [55, 56]). Portanto, heurísticas evoluídas por PG baseadas em espectros de mutação revelaram competitividade em relação à sua alternativa equivalente baseada em cobertura.

4.3.2 Qualidade do Espectro de Mutação

As heurísticas MBFL são construídas sobre a premissa de que uma alteração em um um comando altera o resultado do teste. Quanto mais frequentemente um comando afeta os testes negativos, e quanto menos ele afeta os testes positivos, mais suspeito o comando será considerado [45].

O número de mutantes depende geralmente de aspectos como o tipo de comando, os operadores de mutação, o processo de priorização de mutantes, dentre outros. Como os espectros de mutação incluem os mutantes gerados pelos comandos do programa, investigou-se *como a qualidade do espectro de mutação impacta sobre a capacidade de localizar defeitos (QP3)*; isto é, se o número de mutantes disponíveis é um fator relevante no desempenho das heurísticas MBFL construídas através do algoritmo de Programação Genética.

Pearson *et al.* descobriram recentemente que as heurísticas MBFL têm desempenho ruim em defeitos reais [45], mas eles não consideraram a disponibilidade de dados de mutação em seu estudo. Nos experimetos do presente trabalho, foi utilizado o programa *Math* (obtido do benchmark Defects4J), que é um dos programas que motivaram as conclusões de Pearson *et al.*.

Para responder ao PQ3 dois problemas foram abordados:

- **Defeitos de omissão.** Como a ideia principal por trás das heurísticas MBFL é atribuir suspeitas a mutantes gerados, com base na suposição de que casos de teste que matam mutantes carregam poder de diagnóstico para localização de defeitos [45], defeitos de omissão não oferecem dados de diagnóstico para as heurísticas MBFL. Além disso, o repositório Defects4J (descrito na Subseção 4.2.1) trata uma

defeito de omissão única (*e.g.* um comando ou método ausente) como um cenário de multi-defeito: considera cada local potencialmente relacionado à omissão como um novo defeito (*e.g.* a comando mais próximo dos possíveis pontos para a inserção do código ausente). Em outras palavras, pode haver mais de um local possível para inserir os comandos necessários para a correção, portanto, a cardinalidade entre o defeito de omissão e o ponto de correção é um-para-muitos.

- **Número médio mínimo de mutantes.** Foram selecionadas versões defeituosas do programa com base no *número médio mínimo de mutantes em comandos defeituosos* que representa um limite k no qual foi baseada a medida da qualidade dos espectros de mutação desta análise.

Na Tabela 4.5 a primeira linha mostra o número médio mínimo de mutantes k e as linhas seguintes apresentam o número de versões do programa *Math* com base no limite k imposto: as linhas 2 e 3 apresentam o número versões com e sem defeitos de omissão, respectivamente. Por exemplo, das 77 versões originais (Linha 2), 41 delas não têm defeitos de omissão (Linha 3). Há um número de versões (29 e 16 com e sem defeitos de omissão, respectivamente) cujos comandos defeituosos têm menos de dois mutantes em média. Potencialmente, isso indica uma fonte de dados de mutação ruim e um obstáculo ao aprendizado acerca de elementos do programa defeituosos.

Tabela 4.5: *Versões do programa Math em relação ao número médio mínimo de mutantes K em comandos defeituosos*

K	1	2	3	4	5	6	7	8	9	10
with FOOs	77	48	37	30	26	20	19	14	13	10
without FOOs	41	25	22	20	18	15	14	10	9	6

O objetivo não é encontrar o valor ideal de K, mas usá-lo para responder à questão de pesquisa. Foi definido $K = 3$ após alguns experimentos de calibração, principalmente devido a redução do número de versões disponíveis para o processo de treinamento e da validação cruzada. Desta forma, o número médio mínimo de mutantes foi utilizado em dois cenários distintos, de modo que a análise empírica compara três conjuntos de versões do programa *Math*: α é composto por todas as versões originais no repositório (77 versões); conjunto β, composto pelas versões cujo número médio mínimo de mutantes é limitado a 3 (37 versões); e γ exclui defeitos de omissão de β (22 versões). Portanto, consideraram-se os conjuntos β e γ como espectros de mutação enriquecidos em relação a α, pois eles têm um número médio mínimo de mutantes mais alto.

A Tabela 4.6 apresenta o comparativo dos resultados dos experimentos com *PG-cov* e *PG-mut* para os três conjuntos. Como os conjuntos têm números de versões distintos, utilizou-se a média das medidas de *precision* e *wasted effort* para a comparação (*Acc Méd.* e *Wef Méd.*, respectivamente).

Tabela 4.6: *Medidas de avaliação: Programa Math (Defects4j)*

Set	Técnica	Expense Médio	acc@1 Méd.	acc@3 Méd.	acc@5 Méd.	wef@1 Méd.	web@3 Méd.	web@5 Méd.
α	PG-cov	4,82%	0,26	0,53	0,65	0,77	2,08	3,25
	PG-mut	14,41%	0,23	0,43	0,55	0,81	2,27	3,59
β	PG-cov	4,50%	0,25	0,48	0,61	0,82	2,27	3,69
	PG-mut	8,76%	0,37	0,67	0,81	0,72	1,91	2,91
γ	PG-cov	4,86%	0,10	0,27	0,40	0,91	2,61	4,03
	PG-mut	13,17%	0,27	0,48	0,57	0,75	2,03	3,08

Os resultados do conjunto α revelam a superioridade da *PG-cov* em todas as medidas de avaliação. No entanto, em relação aos conjuntos com espectro de mutação enriquecido (Conjuntos β e γ), *PG-mut* apresentou os melhores resultados em todas as medidas de *precision*; a superioridade também é observada em todas as medidas de *wasted effort*. Para as medidas de *expense* médio, *PG-mut* apresentou maior sensibilidade ao limite de k (número médio mínimo de mutantes) em relação a *PG-cov*: 14,41%, 8,76% e 13,17% nos conjuntos α, β e γ, respectivamente, enquanto *PG-cov* revela estabilidade relativa (4,82%, 4,50% e 4,85 %). No contexto de defeitos de omissões, *PG-cov* expõe pontuações semelhantes entre as medidas para os conjuntos α e β (*e.g.* 4,82% e 4,50 % de *expense* médio e 0,26 e 0,25 na medida *acc@1*), diferente do *PG-mut* em todas as medidas de avaliação. Assim, o desempenho da heurística MBFL foi mais sensível à presença de defeitos de omissão do que as heurísticas baseadas em cobertura.

O *número médio de mutantes* de comandos defeituosos melhora o desempenho da heurística MBFL treinada por Programação Genética. Esta medida foi utilizada como um fator de qualidade dos espectros de mutação. Vale a pena notar que os experimentos da QP1 e QP2 usaram um benchmark robusto para espectros de mutação em relação ao da QP3, então os resultados obtidos são mais positivos para *PG-mut*. Além disso, a presença de defeitos de omissão afeta a eficácia. Estes pontos são bons acréscimos à área de pesquisa, mas certamente exigem mais trabalho antes que possam ser generalizados.

4.3.3 Análise Estatística

A análise estatística foi realizada segundo as diretrizes de Arcuri *et al.* [5] ao aplicar dois testes complementares: teste dos postos sinalizados de *Wilcoxon* para avaliar diferenças estatisticamente significativas com $\alpha = 0,05$, e o teste de magnitude de *Vargha e Delaney* (V-D) $\hat{A}_{12}$ para comparação de efeito com *PG-mut* como A_1 e *PG-cov* como A_2. Ou seja, V-D próximo de zero para *expense* médio indica vantagem de efeito para *PG-mut* e V-D próximo de 1 para as medidas de *precision* indica vantagem de efeito para *PG-mut*. A Tabela 4.7 apresenta os resultados de ambos os testes estatísticos para *expense* médio e *precision* em relação a *PG-mut* e *PG-cov*, com diferenças estatísticas expressivas observadas em todos os programas.

Na Tabela 4.7 todos os *p-valores* destacados em negrito indicam significância

estatística de pelo menos 5% e todos os valores *V-D* destacados indicam uma vantagem na magnitude. Se o valor *V-D* estiver marcado como *, então existe uma vantagem para o método *PG-cov*, caso contrário *PG-mut* tem magnitude superior. Para as medidas de *expense* médio, a significância estatística ($p\text{-valor} \leq 0{,}05$) é observada em quase todas as amostras, exceto no Conjunto 3 de *print_tokens2*. Por sua vez, o teste de *Vargha e Delaney* mostra vantagem de *PG-cov* apesar de ligeira superioridade do *PG-mut* no programa *tcas*.

Quanto à *precision* (valores de *acc* na Tabela 4.7), "–" indica que ambas as amostras são idênticas (todas amostras são zeros). Exceto pelo conjunto α do programa Math (como esperado), os testes de *Wilcoxon* e *Vargha e Delaney* mostram o melhor desempenho de *PG-mut* (significância estatística e magnetude de efeito) em relação a quase todas as amostras. Os testes com conjunto α do programa Math indicam vantagem estatística para *PG-cov*, mas por uma margem menor em relação aos outros conjuntos.

Tabela 4.7: *Teste de Wilcoxon e de Vargha e Delaney $\hat{A}_{12}$ para PG-mut (A_1) e PG-cov (A_2)*

		***Expense* médio**		**acc@1**		**acc@3**		**acc@5**	
Program	**T. Set**	**p-vaule**	**V-D**	**p-vaule**	**V-D**	**p-vaule**	**V-D**	**p-vaule**	**V-D**
475-A	1	**8,90E-13**	**1,00***	–	0,50	–	0,50	–	0,50
	2	**3,71E-3**	0,30	–	0,50	**2,10E-08**	**0,85**	**2,10e-08**	**0,85**
	3	**1,67E-07**	**0,86***	–	0,50	–	0,50	**1,67e-14**	**1,00**
print_tokens2	1	**3,43E-10**	**0,97***	**3,78E-10**	**0,90**	**1,67E-4**	0,74	**3,61e-09**	**0,90**
	2	**1,34E-12**	**0,00**	**5,64E-10**	**0,90**	**3,34E-11**	**0,93**	**3,34e-11**	**0,93**
	3	0,52	0,55	–	0,50	**2,37E-08**	**0,85**	**4,17e-12**	**0,98**
schedule	1	**1,40E-11**	**1,00***	**7,90E-4**	0,72	0,13	0,59	**7,62E-3**	0,36
	2	**8,35E-12**	**1,00***	**1,47E-09**	**0,88**	**1,97E-11**	**0,93**	**2,30E-4**	0,72
	3	**1,61E-06**	**0,86***	–	0,5	0,16	0,47	**6,11e-07**	**0,20***
tcas	1	**2,91E-3**	0,28	**1,55E-09**	**0,9**	**8,96E-13**	**1,00**	**8,97e-13**	**1,00**
	2	**2,31E-2**	0,66	**1,26E-12**	**0,98**	**1,90E-06**	**0,80**	**3,75e-05**	**0,77**
	3	**1,36E-11**	**0,01**	**6,46E-4**	0,67	**8,58E-10**	**0,92**	**2,59e-13**	**1,00**
Math (α)	1	**3,08E-08**	**0,92***	0,13	0,61	0,25	0,41	0,14	0,61
	2	**2,98E-11**	**1,00***	0,93	0,49	**3,64E-06**	**0,16***	**1,11E-3**	**0,26***
	3	**3,00E-11**	**1,00***	**4,76E-08**	**0,09***	**4,70E-10**	**0,04***	**8,68E-11**	**0,01***
Math (β)	1	**5,97E-07**	**0,88***	**3,09E-11**	**0,98**	**3,08E-11**	**0,99**	**2,44E-11**	**1,00**
	2	**2,98E-11**	**1,00***	**6,89E-05**	**0,79**	**2,06E-05**	**0,82**	**1,12E-2**	0,69
	3	**2,76E-11**	**1,00***	0,80	0,52	3,64E-3	0,71	**2,29E-06**	**0,85**
Math (γ)	1	**4,17E-10**	**0,97***	**1,62E-4**	**0,78**	**5,28E-4**	**0,75**	**7,40E-3**	0,69
	2	**2,02E-09**	**0,95***	**8,86E-05**	**0,77**	**1,51E-06**	**0,84**	**3,08E-08**	**0,74**
	3	**2,52E-08**	**0,92***	**1,16E-10**	**0,94**	**1,50E-10**	**0,97**	**9,20E-11**	**0,98**

4.3.4 Ameaças à Validade

As ameaças à *validade interna*, isto é, resultados por acaso, foram reduzidos utilizando-se: técnicas de comparação e medidas de avaliação aplicadas em estudos anteriores da literatura; execuções múltiplas para reduzir a natureza estocástica do algoritmo; e utilização de *frameworks* e softwares de código aberto existentes.

Com relação à *validade externa*, ou seja, se os resultados podem ser generalizados, avaliação em larga escala de defeitos e programas, outras linguagens de programação são necessárias e comparação com mais heurísticas de localização de defeitos. Entretanto, programas reais foram selecionados, bem como defeitos artificiais e reais de diferentes

benchmarks que foram utilizados em diversos contextos relacionados a experimentos de engenharia de software, o que reduziu o viés a certos tipos de projetos e promoveu os resultados para serem alinhados com os esforços da comunidade de pesquisa na área.

Ameaças à *validade de construção* tratam a conformidade do processo de mensuração de resultados. Foram utilizadas métricas de avaliação absolutas e relativas, e as medidas foram tomadas assim como em trabalhos já publicados da literatura, de modo que a capacidade de localização de defeitos seja registrada realisticamente .

4.4 Considerações Finais

Este capítulo relata uma abordagem de Programação Genética (PG) para o problema de localização de defeitos, juntamente com um conjunto de experimentos para avaliar as fórmulas encontradas com relação às heurísticas consolidadas da literatura.

Os aspectos inovadores são a investigação conjunta de: (i) especialização de heurísticas MBFL para determinados contextos; (ii) a aplicação de espectros de mutação a fórmulas desenvolvidas em PG, isto é, sinais além dos dados de cobertura do programa; (iii) uma comparação da eficácia dos espectros de cobertura e de mutação no contexto de abordagens evolutivas; e (iv) o impacto da qualidade dos espectros de mutação na eficácia da SBFL.

A proposta foi comparada a diversas heurísticas bem conhecidas da literatura juntamente com um método baseado em meta-heurística. A abordagem evolutiva baseada em espectro de mutação proposta apresentou melhores resultados de forma geral, embora uma provável maior sensibilidade aos dados de treinamento (*e.g.* número de programas para treinamento, defeitos de omissão, qualidade dos dados de mutação).

O desempenho das heurísticas MBFL obtidas pelo algoritmo de Programação Genética foi medido sob perspectivas relativas e absolutas (QP1 e QP2): o percentual e o número absoluto de elementos de código investigados para localizar defeitos, respectivamente (esta última se aproxima mais da medida percebida pelos engenheiros de software em suas atividades) . O *número médio mínimo de mutantes* foi utilizado para medir a qualidade dos espectros de mutação e seu impacto no processo de localização de defeitos (QP3).

As análises empírica e estatística dos resultados juntas mostram que a proposta é competitiva em ambas as perspectivas investigadas (relativa e absoluta) e que estes são consistentes uns com os outros na avaliação do conjunto de programas completo, dos programas individuais, bem como dos *rank* médio. Especificamente em termos absolutos, o método foi capaz de pontuar o elemento defeituoso com maior frequência em relação às demais heurísticas estudadas. Além disso, foi observado o impacto da métrica de *número médio mínimo de mutantes* no desempenho das heurísticas MBFL treinada.

Conclui-se que as heurísticas MBFL treinadas pelo algoritmo de Programação Genética representam uma adição válida ao campo de pesquisa de localização de defeitos, e que a qualidade das variáveis de mutação impacta diretamente na eficácia do processo de localização de defeitos.

CAPÍTULO 5

Conclusão

Durante desenvolvimento e manutenção de software, defeitos são introduzidos e corrigidos constantemente, o que tem impacto direto sobre a qualidade e o custo do mesmo. A localização de defeitos é parte central para garantir a qualidade do software e refere-se à atividade de identificação da localidade dos defeitos correspondentes às falhas que se tornaram conhecidas durante a etapa de teste de software. Devido à crescente complexidade dos projetos de software, localização de defeitos torna-se cada vez mais uma tarefa onerosa e delongada [50]. Portanto, alguns trabalhos abordaram o problema e propuseram heurísticas para automatizar ou guiar o processo de localização de defeitos.

O presente trabalho apresenta dois métodos para localização de defeitos baseada em busca. A meta-heurística Programação Genética foi aplicada para encontrar combinações lineares de heurísticas baseadas em espectro de cobertura e para gerar heurísticas baseadas em espectro de mutação. Ambos os métodos apresentados buscam soluções sob medida para projetos, obtidas através de uma fase de treino. O trabalho resultou em publicações no 8^o Workshop de Engenharia de Software Baseada em Busca (*WESB 2017*) [8], no *26th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning* (*ESANN 2018*) [9] e no *2018 IEEE Congress on Evolutionary Computation* (IEEE CEC 2018) [10].

O desempenho dos métodos em sua capacidade de atribuir ao elemento de código defeituoso a maior probabilidade de suspeita dentre todos os elementos do programa foi analisado sob duas perspectivas: absoluta e relativa. Os resultados foram investigados para o conjunto de todos os programas utilizados, individualmente por programa e pelo *rank* médio, uma medida de classificação dos resultados de todos métodos comparados em relação a cada métricas de avaliação.

A primeira abordagem apresentada, Capítulo 3, consiste em um método orientado a projetos de busca por equações não lineares para inferir a probabilidade de cada elemento do código ser defeituoso a partir do espectro de cobertura do programa. Foram utilizadas duas configurações de conjuntos de heurísticas para composição, integrados por heurísticas individuais padrão baseadas em dados de cobertura, métricas de associação [32] e variáveis de cobertura.

Durante a análise, o desempenho da abordagem foi contraposto às heurísticas individuais baseadas em cobertura e métricas de associação que integraram os conjuntos para composição além do método de composição linear baseada em Algoritmo Genético de Wang *et al.* [51]. O desempenho dos métodos foi medido sobre um conjunto de sete programas do *Simens Suite* [12], um conjunto de programas comumente aplicados em publicações de localização de defeitos.

O método proposto permitiu fórmulas mais complexas e melhor adaptadas aos programas para os quais foi treinado e de modo geral, ao observar o conjunto de programas completo, foi capaz de superar as heurísticas tradicionais e o método de composição linear de Wang *et al.* [51] em termos absolutos e relativos. Entretanto, a análise de *rank* médio e de programas individuais revelou que o método não foi capaz de superar todos os demais métodos para alguns programas. Isto é explicado pelo baixo número de versões apresentadas nestes programas, o que não permitiu ao algoritmo obter durante a fase de treino soluções suficientemente generalizáveis para a fase de implantação. Assim, constatou-se que o método baseado em Programação Genética é mais dependente da qualidade do conjunto de treino em relação à composição linear, visto que este algoritmo permite soluções mais complexas e que podem tornar-se altamente especializadas ao conjunto e treino.

Similarmente, a segunda abordagem apresentada, Capítulo 4, consiste em um método orientado a projetos para construção de heurísticas baseadas em espectro de mutação, em que são utilizando apenas os dados do espectro de mutação como variáveis. Para análise, as heurísticas obtidas pelo método foram comparadas a um conjunto de heurísticas baseadas em espectro de cobertura e de mutação além de um método de geração de heurísticas baseadas em espectro de cobertura de Yoo [55]. Além da análise de desempenho em perspectivas absoluta e relativa, foi avaliado o impacto da qualidade dos dados de mutação no método proposto.

O desempenho da proposta foi aferido sob as perspectivas relativa e absoluta pelo percentual e o número de elementos necessários investigar até que os defeitos sejam localizados com a utilização do método. Para a análise deste método, foram utilizados nos experimentos programas selecionados nos *benchmarks Siemens Suite* [23], *Codeflaws* [48] e *Defeitos4J* [27].

Para o conjunto completo de programas, a proposta obteve o segundo melhor resultado em termos relativos e os melhores resultados absolutos em relação a todas as comparações, embora com maior sensibilidade aos dados de treinamento (*e.g.* número de programas para treino, defeitos de omissão, qualidade dos dados de mutação). Para a análise em *rank* médio e programas individuais, o método apresentou resultados coerentes com os valores observados no conjunto completo de programas e foi competitivo. Assim, a análise mostrou que a proposta é competitiva em ambas as perspectivas investigadas

e se destacou em resultados absolutos, de modo que os elementos defeituosos foram posicionados mais vezes no topo da lista de elementos de código suspeitos.

Ainda, foi apresentada uma análise inédita do impacto da qualidade dos dados de mutação ao selecionar um subconjuntos de versões de um programa com um número médio mínimo de mutantes, sendo esta uma contribuição importante do trabalho. Foi constatada melhora do desempenho da técnica de construção de heurísticas baseadas em mutação neste subconjunto.

Assim, conclui-se que os métodos propostos são competitivos e tem boa capacidade de adaptação ao conjunto de versões de um mesmo programa em relação às heurísticas genéticas padrão. Assim, as propostas representam contribuições válidas para o campo de pesquisa de localização de defeitos e engenharia de software baseada em busca, o que é evidenciado pelas publicações obtidas.

Como trabalhos futuros, estão previstos estudos acerca do desempenho e da adaptabilidade dos métodos em projetos maiores, em diferentes tipos de defeitos e em programas multi-defeito. Ainda, avaliar qual o melhor conjunto de terminais para Programação Genética considerando quais heurísticas estão mais frequentemente nas melhores soluções e quais são as heurísticas que geram melhores resultados quando aplicadas individualmente. Outros desdobramentos são uma investigação em maior escala sobre a qualidade dos espectros de mutação e um estudo com hibridização de dados de cobertura e mutação.

Referências Bibliográficas

[1] ABREU, R.; ZOETEWEIJ, P.; GEMUND, A. J. C. V. **An evaluation of similarity coefficients for software fault localization**. In: *Proceedings of the 12th Pacific Rim International Symposium on Dependable Computing*, PRDC '06, p. 39–46, Washington, DC, USA, 2006.

[2] ABREU, R.; ZOETEWEIJ, P.; GEMUND, A. J. C. V. **Spectrum-based multiple fault localization**. In: *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, ASE '09, p. 88–99, Washington, DC, USA, 2009. IEEE Computer Society.

[3] ABREU, R.; ZOETEWEIJ, P.; GOLSTEIJN, R.; VAN GEMUND, A. J. C. **A practical evaluation of spectrum-based fault localization**. *J. Syst. Softw.*, 82(11):1780–1792, November 2009.

[4] ANDREWS, J.; BRIAND, L.; LABICHE, Y. **Is mutation an appropriate tool for testing experiments? [software testing]**. In: *Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on*, p. 402–411, May 2005.

[5] ARCURI, A.; BRIAND, L. **A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering**. *Softw. Test. Verif. Reliab.*, 24(3):219–250, May 2014.

[6] B. LE, T.-D.; LO, D.; LE GOUES, C.; GRUNSKE, L. **A learning-to-rank based fault localization approach using likely invariants**. In: *Proceedings of the 25th International Symposium on Software Testing and Analysis*, ISSTA 2016, p. 177–188, New York, NY, USA, 2016. ACM.

[7] CAMPOS, J.; ABREU, R.; FRASER, G.; D'AMORIM, M. **Entropy-based test generation for improved fault localization**. In: *2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013 - Proceedings*, p. 257–267, 2013.

[8] DE FREITAS, D. M.; LEITAO-JUNIOR, P.; CAMILO-JUNIOR, C.; DANTAS, A.; HARRISON, R. **Genetic programming-based composition of fault localization heuristics**. In: *CBSoft 2017 - WESB*, Fortaleza, CE, Brazil, Sept 2017.

[9] DE FREITAS, D. M.; LEITAO-JUNIOR, P.; CAMILO-JUNIOR, C.; HARRISON, R. **Evolutionary composition of customised fault localisation heuristics**. In: *ESANN 2018 - 26th European Symposium on Artificial Neural Networks*, Bruges, Belgium, April 2018.

[10] DE FREITAS, D. M.; LEITAO-JUNIOR, P.; CAMILO-JUNIOR, C.; HARRISON, R. **Mutation-based evolutionary fault localisation**. In: *2018 IEEE Congress on Evolutionary Computation (CEC)*, Rio de Janeiro, RJ, Brazil, July 2018.

[11] DEMILLO, R.; LIPTON, R.; SAYWARD, F. **Hints on test data selection: Help for the practicing programmer**. *Computer*, 11(4):34–41, April 1978.

[12] DO, H.; ELBAUM, S.; ROTHERMEL, G. **Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact**. *Empirical Softw. Engg.*, 10(4):405–435, Oct. 2005.

[13] FISHER, R. A. **The Genetical Theory of Natural Selection**. Oxford University Press, 2000.

[14] GENG, J.; LI, Z.; ZHAO, R.; GUO, J. **Search based test suite minimization for fault detection and localization: A co-driven method**. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9962 LNCS:34–48, 2016.

[15] GLOVER, F. **Future paths for integer programming and links to artificial intelligence**. *Comput. Oper. Res.*, 13(5):533–549, May 1986.

[16] GONG, P.; ZHAO, R.; LI, Z. **Faster mutation-based fault localization with a novel mutation execution strategy**. In: *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, p. 1–10, April 2015.

[17] HAILPERN, B.; SANTHANAM, P. **Software debugging, testing, and verification**. *IBM Systems Journal*, 41(1):4–12, 2002.

[18] HARMAN, M.; JONES, B. F. **Search-based software engineering**. *Information and Software Technology*, 43(14):833 – 839, 2001.

[19] HARMAN, M.; MCMINN, P.; DE SOUZA, J. T.; YOO, S. **Empirical software engineering and verification**. In: Meyer, B.; Nordio, M., editors, *Empirical Software Engineering and Verification*, chapter Search Based Software Engineering: Techniques, Taxonomy, Tutorial, p. 1–59. Springer-Verlag, Berlin, Heidelberg, 2012.

[20] HOLLAND, J. H. **Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence**. MIT Press, Cambridge, MA, USA, 1992.

[21] HUANG, L.; AI, J. **Automatic software fault localization based on artificial bee colony**. *Journal of Systems Engineering and Electronics*, 26(6):1325–1332, 2015.

[22] HUI, Z. **Fault localization method generated by regression test cases on the basis of genetic immune algorithm**. In: *Proceedings - International Computer Software and Applications Conference*, volume 2, p. 46–51, 2016.

[23] HUTCHINS, M.; FOSTER, H.; GORADIA, T.; OSTRAND, T. **Experiments of the effectiveness of dataflow- and controlflow-based test adequacy criteria**. In: *Proceedings of the 16th International Conference on Software Engineering*, ICSE '94, p. 191–200, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[24] IEEE. **ISO/IEC/IEEE 29119-1:2013(E), Software and systems engineering - Software testing - Part 1: Concepts and definitions**. IEEE, 2013.

[25] JONES, J. A.; HARROLD, M. J.; STASKO, J. **Visualization of test information to assist fault localization**. In: *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*, p. 467–477, May 2002.

[26] JONES, J. A.; HARROLD, M. J.; STASKO, J. T. **Visualization for fault localization**. In: *in Proceedings of ICSE 2001 Workshop on Software Visualization*, p. 71–75, Toronto, ON, Canada, 2001.

[27] JUST, R.; JALALI, D.; ERNST, M. D. **Defects4J: A database of existing faults to enable controlled testing studies for Java programs**. In: *Proc. of the 2014 International Symposium on Software Testing and Analysis*, ISSTA 2014, p. 437–440, New York, NY, USA, 2014. ACM.

[28] KANG, D.; SOHN, J.; YOO, S. **Empirical evaluation of conditional operators in gp based fault localization**. In: *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, p. 1295–1302, New York, NY, USA, 2017. ACM.

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

[30] LE, T.-D. B.; THUNG, F.; LO, D. **Theory and practice, do they match? a case with spectrum-based fault localization**. In: *Proceedings of the 2013 IEEE International Conference on Software Maintenance*, ICSM '13, p. 380–383, Washington, DC, USA, 2013. IEEE Computer Society.

[31] LI, X.; WONG, W.; GAO, R.; HU, L.; HOSONO, S. **Genetic algorithm-based test generation for software product line with the integration of fault localization techniques**. *Empirical Software Engineering*, p. 1–51, 2017.

[32] LUCIA.; LO, D.; JIANG, L.; BUDI, A. **Comprehensive evaluation of association measures for fault localization**. In: *2010 IEEE Intl. Conference on Software Maintenance*, p. 1–10, Timișoara, Romania, Sept 2010.

[33] LUKE, S. **Essentials of Metaheuristics**. Lulu, second edition, 2013. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

[34] MALDONADO, J. C.; DELAMARO, M. E.; FABBRI, S. C. P. F.; SIMÃO, A. D. S.; SUGETA, T.; VINCENZI, A. M. R.; MASIERO, P. C. **Proteum: A family of tools to support specification and program testing based on mutation**. In: Wong, W. E., editor, *Mutation Testing for the New Century*, p. 113–116. Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[35] MOON, S.; KIM, Y.; KIM, M.; YOO, S. **Ask the mutants: Mutating faulty programs for fault localization**. In: *Proceedings of the 2014 IEEE International Conference on Software Testing, Verification, and Validation*, ICST '14, p. 153–162, Washington, DC, USA, 2014.

[36] MYERS, G. **Software Reliability: Principles and Practices**. A Wiley Interscience publication. Wiley, 1976.

[37] MYERS, G. J.; SANDLER, C. **The Art of Software Testing**. John Wiley & Sons, Inc., USA, 2004.

[38] NAISH, L.; NEELOFAR.; RAMAMOHANARAO, K. **Multiple bug spectral fault localization using genetic programming**. In: *2015 24th Australasian Software Engineering Conference*, p. 11–17, Sept 2015.

[39] NAISH, L.; LEE, H. J.; RAMAMOHANARAO, K. **A model for spectra-based software diagnosis**. *ACM Trans. Softw. Eng. Methodol.*, 20(3):11:1–11:32, Aug. 2011.

[40] NAISH, L.; LEE, H. J.; RAMAMOHANARAO, K. **A model for spectra-based software diagnosis**. *ACM Trans. Softw. Eng. Methodol.*, 20(3):11:1–11:32, Aug. 2011.

[41] NEELOFAR, N.; NAISH, L.; RAMAMOHANARAO, K. **Spectral-based fault localization using hyperbolic function**. *Software - Practice and Experience*, 2017. cited By 0; Article in Press.

[42] PAPADAKIS, M.; TRAON, Y. L. **Using mutants to locate "unknown"faults**. In: *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, p. 691–700, April 2012.

[43] PAPADAKIS, M.; LE TRAON, Y. **Metallaxis-FL: Mutation-based fault localization**. *Softw. Test. Verif. Reliab.*, 25(5-7):605–628, Aug. 2015.

[44] PARSA, S.; PORSHOKOOH, H.; TEYMOURI, S.; VAHIDI-ASL, M. **A heuristic test data generation approach for program fault localization**. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7002 LNAI(PART 1):236–243, 2011.

[45] PEARSON, S.; CAMPOS, J.; JUST, R.; FRASER, G.; ABREU, R.; ERNST, M. D.; PANG, D.; KELLER, B. **Evaluating and improving fault localization**. In: *Proceedings of the 39th International Conference on Software Engineering*, ICSE '17, p. 609–620, Piscataway, NJ, USA, 2017.

[46] ROTHLAUF, F. **Design of Modern Heuristics: Principles and Application**. Springer Publishing Company, Incorporated, 1st edition, 2011.

[47] TALBI, E.-G. **Metaheuristics: From Design to Implementation**. Wiley, 2009.

[48] TAN, S. H.; YI, J. Y.; YULIS.; MECHTAEV, S.; ROYCHOUDHURY, A. **Codeflaws: A Programming Competition Benchmark for Evaluating Automated Program Repair Tools**.

[49] VARGHA, A.; DELANEY, H. D. **A critique and improvement of the cl common language effect size statistics of mcgraw and wong**. *Journal of Educational and Behavioral Statistics*, 25(2):101–132, 2000.

[50] VESSEY, I. **Expertise in debugging computer programs: An analysis of the content of verbal protocols**. *IEEE Trans. Syst. Man Cybern.*, 16(5):621–637, Sept. 1986.

[51] WANG, S.; LO, D.; JIANG, L.; LUCIA.; LAU, H. C. **Search-based fault localization**. In: *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, ASE '11, p. 556–559, Washington, DC, USA, 2011.

[52] WONG, W. E.; DEBROY, V.; GAO, R.; LI, Y. **The dstar method for effective software fault localization**. *IEEE Transactions on Reliability*, 63(1):290–308, March 2014.

[53] WONG, W. E.; GAO, R.; LI, Y.; ABREU, R.; WOTAWA, F. **A survey on software fault localization**. *IEEE Transactions on Software Engineering*, 42(8):707–740, Aug 2016.

[54] XIE, X.; KUO, F.-C.; CHEN, T. Y.; YOO, S.; HARMAN, M. **Provably optimal and human-competitive results in sbse for spectrum based fault localisation**. In: *Proc. of the 5th International Symposium on Search Based Software Engineering - Volume 8084*, SSBSE 2013, p. 224–238, New York, NY, USA, 2013. Springer-Verlag New York, Inc.

[55] YOO, S. **Evolving human competitive spectra-based fault localisation techniques**. In: *Proceedings of the 4th International Conference on Search Based Software Engineering*, SSBSE'12, p. 244–258, Berlin, Heidelberg, 2012. Springer-Verlag.

[56] YOO, S.; XIE, X.; KUO, F.-C.; CHEN, T. Y.; HARMAN, M. **Human competitiveness of genetic programming in spectrum-based fault localisation: Theoretical and empirical analysis**. *ACM Trans. Softw. Eng. Methodol.*, 26(1):4:1–4:30, June 2017.

APÊNDICE A

Dados da Métrica *Rank* Médio

Este apêndice apresenta as tabelas completas de todos os resultados da métrica de avaliação *rank* médio em relação às medidas de *expense* médio, precisão ($acc@m$) e esforço perdido ($wef@n$) das análises da Seção 3.3.2 do Capítulo 3. As Tabelas A.1 e A.2 apresentam os valores de *rank* médio de cada programa para o conjunto completo das heurísticas que compõem o conjunto de terminais $T1$ e $T2$ respectivamente e os métodos de composição configurados com o mesmos. A Tabela A.3 apresenta o *rank* médio total para todas as heurísticas apresentadas no Capítulo 3 (presentes nos conjuntos de terminais $T1$ e $T2$) e as duas variações de ambas as técnicas de composição (GP_{T1}, GA_{T1}, GP_{T2} e GA_{T2}).

Tabela A.1: *Rank médio para as heurísticas que compõem o conjunto de terminais $T1$ e os métodos de composição a ele relacionados (GP_{T1} e GA_{T1})*

***Expense* Médio**	**acc@1**	**acc@3**	**acc@5**	**acc@10**	**wef@1**	**wef@3**	**wef@5**	**wef@10**
GA_{T1}: 1,43	GP_{T1}: 4,57	GA_{T1}: 6,57	GP_{T1}: 4,43	GP_{T1}: 2,57	GP_{T1}: 4,57	GP_{T1}: 5,14	GA_{T1}: 5,14	GA_{T1}: 4,86
GP_{T1}: 3,43	GA_{T1}: 10,00	GP_{T1}: 6,57	AM_5: 6,29	GA_{T1}: 7,43	GA_{T1}: 10,00	GA_{T1}: 5,86	GP_{T1}: 5,14	GP_{T1}: 5,14
AM_{10}: 5,14	AM_5: 12,71	AM_{15}: 11,00	GA_{T1}: 7,00	AM_5: 9,57	AM_5: 12,71	AM_{15}: 9,43	AM_5: 8,29	H_{33}: 8,00
AM_{15}: 5,86	AM_6: 12,86	AM_1: 12,00	AM_6: 9,43	AM_6: 10,00	AM_6: 12,86	AM_4: 10,43	AM_{15}: 9,14	AM_{15}: 8,14
H_{33}: 6,29	AM_1: 17,43	AM_4: 12,00	AM_{15}: 10,71	H_{33}: 10,57	AM_1: 17,43	AM_{11}: 10,43	H_{33}: 10,57	AM_5: 8,29
AM_1: 7,00	AM_4: 17,43	AM_{10}: 12,00	H_{33}: 13,29	AM_{15}: 12,29	AM_4: 17,43	AM_5: 10,43	AM_4: 11,00	AM_4: 9,86
AM_{11}: 7,43	AM_{10}: 17,43	AM_{11}: 12,00	AM_1: 14,43	H_{32}: 13,57	AM_{10}: 17,43	AM_{13}: 11,00	AM_{11}: 11,00	AM_{11}: 9,86
AM_4: 8,00	AM_{11}: 17,43	AM_{13}: 12,00	AM_4: 14,43	AM_4: 13,86	AM_{11}: 17,43	AM_{10}: 11,29	AM_{13}: 11,43	AM_6: 10,00
AM_{13}: 8,57	AM_{13}: 17,43	H_{33}: 12,00	AM_{10}: 14,43	AM_{11}: 13,86	AM_{13}: 17,43	H_{33}: 11,29	AM_6: 11,43	AM_{13}: 10,43
AM_5: 9,71	AM_{14}: 17,43	AM_5: 12,14	AM_{11}: 14,43	AM_{13}: 14,14	AM_{14}: 17,43	AM_1: 11,86	AM_{10}: 11,71	AM_{10}: 10,86
AM_{14}: 10,00	AM_{15}: 17,43	AM_6: 12,29	AM_{13}: 14,43	AM_1: 14,57	AM_{15}: 17,43	AM_6: 12,00	AM_1: 12,14	AM_1: 11,29
AM_6: 11,00	H_{32}: 17,43	AM_{14}: 13,57	H_{32}: 14,57	AM_{10}: 14,57	H_{32}: 17,43	AM_{14}: 13,57	H_{32}: 13,43	H_{32}: 12,00
H_{32}: 11,29	H_{33}: 17,43	H_{32}: 13,57	AM_{14}: 15,43	AM_{14}: 15,86	H_{33}: 17,43	H_{32}: 13,57	AM_{14}: 14,29	AM_{14}: 13,86
AM_{12}: 15,71	AM_2: 19,86	AM_2: 15,86	AM_2: 15,71	AM_2: 16,14	AM_2: 19,86	AM_2: 16,86	AM_2: 15,71	AM_2: 15,14
AM_2: 16,14	AM_3: 19,86	AM_3: 15,86	AM_3: 15,71	AM_3: 16,14	AM_3: 19,86	AM_3: 16,86	AM_3: 15,71	AM_3: 15,14
AM_3: 16,14	AM_7: 19,86	AM_{12}: 15,86	AM_8: 15,71	AM_{12}: 16,14	AM_7: 19,86	AM_{12}: 16,86	AM_{12}: 15,71	AM_{12}: 15,14
AM_{16}: 17,29	AM_8: 19,86	AM_7: 17,71	AM_9: 15,71	AM_7: 16,43	AM_8: 19,86	AM_7: 17,71	AM_8: 17,00	AM_7: 16,86
AM_7: 17,86	AM_9: 19,86	AM_8: 17,71	AM_{12}: 15,71	AM_8: 16,86	AM_9: 19,86	AM_8: 17,71	AM_9: 17,00	AM_8: 17,00
AM_9: 18,43	AM_{12}: 19,86	AM_9: 17,71	AM_7: 16,29	AM_9: 16,86	AM_{12}: 19,86	AM_9: 17,71	AM_7: 17,57	AM_9: 17,00
AM_8: 18,71	AM_{16}: 20,00	AM_{16}: 20,00	AM_{16}: 20,00	AM_{16}: 20,00	AM_{16}: 20,00	AM_{16}: 20,00	AM_{16}: 20,00	AM_{16}: 20,00

Tabela A.2: *Rank médio para as heurísticas que compõem o conjunto de terminais $T2$ e os métodos de composição a ele relacionados (GP_{T2} e GA_{T2})*

***Expense* Médio**	**acc@1**	**acc@3**	**acc@5**	**acc@10**	**wef@1**	**wef@3**	**wef@5**	**wef@10**
H_{29}: 2,57	GP_{T2}: 15,57	GP_{T2}: 9,57	GP_{T2}: 5,86	GP_{T2}: 5,29	GP_{T2}: 15,57	GP_{T2}: 7,86	H_{16}: 8,14	H_{29}: 3,00
GP_{T2}: 5,86	H_7: 25,14	H_{16}: 16,43	H_{16}: 9,00	H_{29}: 9,14	H_7: 25,14	H_{11}: 14,14	H_{29}: 8,14	H_{16}: 3,14
H_{16}: 6,00	H_{16}: 30,00	H_{29}: 16,43	H_{29}: 9,00	H_{16}: 9,57	H_{16}: 30,00	H_{16}: 14,14	GP_{T2}: 8,14	H_{24}: 8,43
H_{13}: 7,29	H_{29}: 30,00	H_{11}: 17,57	H_7: 18,57	H_7: 16,57	H_{29}: 30,00	H_{29}: 14,14	H_{11}: 14,14	GP_{T2}: 8,43
H_{33}: 9,14	H_8: 31,00	H_5: 20,57	H_{12}: 19,86	H_{13}: 16,71	H_8: 31,00	H_{24}: 15,86	H_{24}: 14,29	H_{33}: 9,29
H_{24}: 9,86	GA_{T2}: 31,29	H_{12}: 20,86	H_{17}: 19,86	H_{24}: 18,14	GA_{T2}: 31,29	H_{17}: 16,71	H_{17}: 15,14	H_{17}: 9,57
H_{18}: 9,86	H_1: 33,29	H_{17}: 20,86	H_{18}: 19,86	H_{17}: 18,43	H_1: 33,29	H_{18}: 16,71	H_{18}: 15,14	H_{18}: 9,57
H_{12}: 10,00	H_2: 33,29	H_{18}: 20,86	H_{24}: 19,86	H_{18}: 18,43	H_2: 33,29	H_{33}: 18,14	H_{33}: 16,57	H_{12}: 10,14
H_{17}: 10,00	H_5: 33,29	H_{24}: 20,86	H_{33}: 19,86	H_{12}: 18,86	H_5: 33,29	H_{12}: 19,00	H_{12}: 17,43	H_{13}: 12,57
H_7: 11,71	H_9: 33,29	H_{33}: 20,86	H_{11}: 20,14	H_{33}: 18,86	H_9: 33,29	H_5: 20,43	H_7: 18,00	H_7: 12,71
GA_{T2}: 12,43	H_{10}: 33,29	H_2: 23,86	H_5: 21,43	H_{11}: 19,71	H_{10}: 33,29	H_7: 20,71	H_5: 19,29	H_{11}: 12,86
H_2: 13,57	H_{11}: 33,29	H_7: 25,29	H_{13}: 21,43	H_{34}: 19,86	H_{11}: 33,29	H_2: 23,71	H_2: 20,86	H_{34}: 16,14
H_4: 15,43	H_{12}: 33,29	H_{34}: 25,71	H_2: 21,86	H_5: 21,14	H_{12}: 33,29	H_{34}: 23,86	H_{13}: 20,86	H_5: 17,57
H_{14}: 16,57	H_{13}: 33,29	H_1: 27,29	H_{34}: 21,86	H_2: 22,29	H_{13}: 33,29	H_{13}: 25,29	H_{34}: 21,00	H_2: 18,57
H_{26}: 20,00	H_{14}: 33,29	H_{10}: 27,29	H_{14}: 24,71	GA_{T2}: 22,29	H_{14}: 33,29	H_1: 27,00	H_{14}: 24,14	H_{26}: 22,29
H_{27}: 20,00	H_{15}: 33,29	H_{13}: 27,29	H_{26}: 24,71	H_{14}: 24,86	H_{15}: 33,29	H_{10}: 27,00	H_{26}: 24,14	H_{27}: 22,29
H_{28}: 20,00	H_{17}: 33,29	H_{14}: 27,29	H_{27}: 24,71	H_{26}: 24,86	H_{17}: 33,29	H_{14}: 27,00	H_{27}: 24,14	H_{28}: 22,29
H_{30}: 20,00	H_{18}: 33,29	H_{26}: 27,29	H_{28}: 24,71	H_{27}: 24,86	H_{18}: 33,29	H_{26}: 27,00	H_{28}: 24,14	H_{30}: 22,29
H_{31}: 20,00	H_{24}: 33,29	H_{27}: 27,29	H_{30}: 24,71	H_{28}: 24,86	H_{24}: 33,29	H_{27}: 27,00	H_{30}: 24,14	H_{31}: 22,29
H_{11}: 20,14	H_{25}: 33,29	H_{28}: 27,29	H_{31}: 24,71	H_{30}: 24,86	H_{25}: 33,29	H_{28}: 27,00	H_{31}: 24,14	H_{14}: 22,86
H_{34}: 20,57	H_{26}: 33,29	H_{30}: 27,29	H_1: 26,29	H_{31}: 24,86	H_{26}: 33,29	H_{30}: 27,00	H_1: 25,57	GA_{T2}: 23,14
H_5: 20,71	H_{27}: 33,29	H_{31}: 27,29	H_{10}: 26,57	H_1: 25,29	H_{27}: 33,29	H_{31}: 27,00	H_{10}: 25,86	H_1: 23,86
H_1: 22,14	H_{28}: 33,29	H_{32}: 27,29	H_{32}: 26,57	H_{10}: 25,29	H_{28}: 33,29	H_{32}: 27,00	H_{32}: 25,86	H_{10}: 24,43
H_{32}: 23,29	H_{30}: 33,29	H_9: 27,71	GA_{T2}: 26,71	H_{32}: 25,29	H_{30}: 33,29	H_9: 27,57	H_9: 26,29	H_{32}: 24,43
H_{10}: 23,71	H_{31}: 33,29	H_{15}: 30,00	H_9: 26,86	H_9: 28,00	H_{31}: 33,29	H_8: 27,71	GA_{T2}: 29,43	H_9: 26,86
H_{15}: 25,71	H_{32}: 33,29	H_{25}: 30,00	H_{15}: 30,57	H_4: 28,14	H_{32}: 33,29	H_{15}: 30,14	H_{15}: 30,14	H_4: 29,14
H_{25}: 27,00	H_{33}: 33,29	GA_{T2}: 30,14	H_{25}: 30,57	H_3: 29,71	H_{33}: 33,29	H_{25}: 30,14	H_{25}: 30,14	H_{15}: 29,71
H_9: 27,57	H_{34}: 33,29	H_{19}: 33,29	H_4: 33,29	H_{20}: 29,71	H_{34}: 33,29	GA_{T2}: 32,57	H_8: 32,00	H_3: 29,86
H_{20}: 27,71	H_{19}: 34,00	H_{21}: 33,29	H_{19}: 33,43	H_{15}: 30,14	H_{19}: 34,00	H_{19}: 33,29	H_4: 33,14	H_{20}: 29,86
H_3: 28,57	H_{21}: 34,00	H_{22}: 33,29	H_{21}: 33,43	H_{25}: 31,14	H_{21}: 34,00	H_{21}: 33,29	H_{19}: 33,14	H_{25}: 30,43
H_{19}: 30,57	H_{22}: 34,00	H_{23}: 33,29	H_{22}: 33,43	H_{19}: 33,57	H_{22}: 34,00	H_{22}: 33,29	H_{21}: 33,14	H_{19}: 33,00
H_{21}: 30,57	H_{23}: 34,00	H_8: 33,57	H_{23}: 33,43	H_{21}: 33,57	H_{23}: 34,00	H_{23}: 33,29	H_{22}: 33,14	H_{21}: 33,00
H_{22}: 30,57	H_3: 36,00	H_4: 34,43	H_8: 34,00	H_{22}: 33,57	H_3: 36,00	H_4: 34,43	H_{23}: 33,14	H_{22}: 33,00
H_{23}: 30,57	H_4: 36,00	H_3: 36,00	H_3: 34,86	H_{23}: 33,57	H_4: 36,00	H_3: 36,00	H_3: 35,14	H_{23}: 33,00
H_6: 31,71	H_6: 36,00	H_6: 36,00	H_{20}: 34,86	H_8: 34,57	H_6: 36,00	H_6: 36,00	H_{20}: 35,14	H_8: 34,00
H_8: 35,86	H_{20}: 36,00	H_{20}: 36,00	H_6: 36,00	H_6: 35,43	H_{20}: 36,00	H_{20}: 36,00	H_6: 36,00	H_6: 35,43

Tabela A.3: *Rank médio para todas as heurísticas apresentadas (presentes nos conjuntos de terminais $T1$ e $T2$) e as duas variações de ambas as técnicas de composição (GP_{T1}, GA_{T1}, GP_{T2} e GA_{T2})*

***Expense* Médio**	**acc@1**	**acc@3**	**acc@5**	**acc@10**	**wef@1**	**wef@3**	**wef@5**	**wef@10**
GA_{T1}: 2,57	GP_{T1}: 9,86	GP_{T1}: 15,29	GP_{T1}: 7,86	GP_{T1}: 5,29	GP_{T1}: 9,86	GP_{T1}: 10,71	GP_{T1}: 11,00	H_{29}: 4,86
H_{29}: 4,86	GP_{T2}: 23,57	GP_{T2}: 16,29	GP_{T2}: 10,29	GP_{T2}: 7,43	GP_{T2}: 23,57	GP_{T2}: 13,57	H_{16}: 12,43	H_{16}: 5,00
GP_{T1}: 7,71	GA_{T1}: 24,71	GA_{T1}: 18,00	H_{16}: 14,14	H_{29}: 14,29	GA_{T1}: 24,71	GA_{T1}: 13,71	H_{29}: 12,43	GA_{T1}: 11,14
H_{16}: 9,71	AM_5: 34,43	H_{16}: 24,71	H_{29}: 14,14	H_{16}: 14,86	AM_5: 34,43	H_{16}: 21,71	GA_{T1}: 12,71	GP_{T1}: 11,14
GP_{T2}: 9,71	AM_6: 34,57	H_{29}: 24,71	GA_{T1}: 15,57	GA_{T1}: 16,86	AM_6: 34,57	H_{29}: 21,71	GP_{T2}: 13,29	GP_{T2}: 13,86
AM_{10}: 10,14	H_7: 37,00	H_{11}: 27,00	AM_5: 16,71	H_{13}: 24,00	H_7: 37,00	H_{11}: 22,14	AM_5: 17,43	H_{24}: 15,00
H_{13}: 12,00	H_{16}: 44,29	AM_5: 28,14	AM_6: 23,57	AM_5: 24,43	H_{16}: 44,29	AM_{15}: 23,14	AM_{15}: 22,14	H_{33}: 16,29
H_{18}: 14,00	H_{29}: 44,29	AM_6: 28,57	AM_{15}: 26,43	H_7: 25,29	H_{29}: 44,29	H_{24}: 25,14	H_{11}: 22,71	H_{17}: 16,29
H_{17}: 14,14	GA_{T2}: 46,86	AM_{15}: 29,29	H_7: 28,86	AM_6: 25,71	GA_{T2}: 46,86	AM_4: 25,14	H_{24}: 23,14	H_{18}: 16,29
H_{12}: 14,29	H_8: 47,00	H_5: 30,29	H_{11}: 31,57	H_{24}: 27,00	H_8: 47,00	AM_{11}: 25,14	AM_6: 24,14	H_{12}: 17,14
H_{33}: 14,43	H_1: 48,71	H_{12}: 31,29	H_{12}: 31,86	H_{17}: 27,43	H_1: 48,71	H_{17}: 26,57	H_{17}: 24,43	H_7: 19,00
H_{24}: 14,57	H_2: 48,71	H_{17}: 31,29	H_{17}: 31,86	H_{18}: 27,43	H_2: 48,71	H_{18}: 26,57	H_{18}: 24,43	H_{13}: 19,00
AM_{15}: 15,29	H_5: 48,71	H_{18}: 31,29	H_{18}: 31,86	H_{12}: 27,86	H_5: 48,71	AM_{13}: 26,57	AM_4: 25,57	H_{11}: 19,57
H_7: 17,43	H_9: 48,71	H_{24}: 31,29	H_{24}: 31,86	H_{33}: 27,86	H_9: 48,71	AM_5: 27,43	AM_{11}: 25,57	AM_{15}: 19,57
AM_1: 18,14	H_{10}: 48,71	H_{33}: 31,29	H_{33}: 31,86	H_{34}: 29,00	H_{10}: 48,71	H_{33}: 28,00	H_{33}: 25,86	AM_5: 21,57
AM_{11}: 18,71	H_{11}: 48,71	AM_1: 31,29	H_{13}: 33,57	H_{11}: 29,00	H_{11}: 48,71	AM_{10}: 28,00	AM_{13}: 26,86	AM_4: 23,14
H_2: 19,14	H_{12}: 48,71	AM_4: 31,29	H_2: 34,00	H_5: 30,43	H_{12}: 48,71	H_{12}: 29,43	H_7: 27,00	AM_{11}: 23,14
AM_4: 19,43	H_{13}: 48,71	AM_{10}: 31,29	H_{34}: 34,00	AM_{15}: 31,57	H_{13}: 48,71	AM_1: 29,43	H_{12}: 27,14	H_{34}: 23,71
GA_{T2}: 20,00	H_{14}: 48,71	AM_{11}: 31,29	H_5: 34,43	H_2: 31,86	H_{14}: 48,71	H_5: 30,14	AM_{10}: 28,29	AM_{13}: 23,86
AM_{13}: 21,00	H_{15}: 48,71	AM_{13}: 31,29	AM_1: 34,43	GA_{T2}: 32,86	H_{15}: 48,71	H_7: 30,86	H_5: 29,14	AM_6: 24,14
H_4: 23,14	H_{17}: 48,71	H_2: 34,57	AM_4: 34,43	AM_4: 34,57	H_{17}: 48,71	AM_6: 31,29	AM_1: 29,57	AM_{10}: 24,43
AM_5: 24,00	H_{18}: 48,71	H_7: 36,71	AM_{10}: 34,43	AM_{11}: 34,57	H_{18}: 48,71	H_2: 34,86	H_2: 31,43	AM_1: 24,86
H_{14}: 24,29	H_{24}: 48,71	H_{34}: 37,00	AM_{11}: 34,43	AM_{13}: 34,86	H_{24}: 48,71	H_{34}: 35,14	H_{34}: 31,57	H_5: 25,00
AM_{14}: 25,14	H_{25}: 48,71	AM_2: 38,14	AM_{13}: 34,43	AM_1: 35,29	H_{25}: 48,71	H_{13}: 36,71	H_{13}: 31,71	H_2: 26,71
AM_6: 27,57	H_{26}: 48,71	AM_3: 38,14	AM_2: 36,57	AM_{10}: 35,29	H_{26}: 48,71	H_1: 38,57	H_{14}: 35,57	H_{26}: 32,14
H_{26}: 28,29	H_{27}: 48,71	AM_{12}: 38,14	AM_3: 36,57	H_{14}: 36,29	H_{27}: 48,71	H_{10}: 38,57	H_{26}: 35,57	H_{27}: 32,14
H_{27}: 28,29	H_{28}: 48,71	H_1: 38,86	AM_8: 36,57	H_{26}: 36,29	H_{28}: 48,71	H_{14}: 38,57	H_{27}: 35,57	H_{28}: 32,14
H_{28}: 28,29	H_{30}: 48,71	H_{10}: 38,86	AM_9: 36,57	H_{27}: 36,29	H_{30}: 48,71	H_{26}: 38,57	H_{28}: 35,57	H_{30}: 32,14
H_{30}: 28,29	H_{31}: 48,71	H_{13}: 38,86	AM_{12}: 36,57	H_{28}: 36,29	H_{31}: 48,71	H_{27}: 38,57	H_{30}: 35,57	H_{31}: 32,14
H_{31}: 28,29	H_{32}: 48,71	H_{14}: 38,86	H_{14}: 37,29	H_{30}: 36,29	H_{32}: 48,71	H_{28}: 38,57	H_{31}: 35,57	H_{14}: 32,71
H_5: 30,00	H_{33}: 48,71	H_{26}: 38,86	H_{26}: 37,29	H_{31}: 36,29	H_{33}: 48,71	H_{30}: 38,57	H_1: 37,00	H_1: 33,86
H_{34}: 30,00	H_{34}: 48,71	H_{27}: 38,86	H_{27}: 37,29	H_1: 36,86	H_{34}: 48,71	H_{31}: 38,57	H_{10}: 37,29	H_{10}: 34,43
H_{11}: 30,14	AM_1: 48,71	H_{28}: 38,86	H_{28}: 37,29	H_{10}: 36,86	AM_1: 48,71	H_{32}: 38,57	H_{32}: 37,29	H_{32}: 34,43
H_1: 31,29	AM_4: 48,71	H_{30}: 38,86	H_{30}: 37,29	H_{32}: 36,86	AM_4: 48,71	AM_{14}: 38,57	H_9: 37,71	GA_{T2}: 34,57
H_{32}: 32,57	AM_{10}: 48,71	H_{31}: 38,86	H_{31}: 37,29	AM_2: 38,57	AM_{10}: 48,71	H_9: 39,43	AM_{14}: 38,00	AM_{14}: 37,14
H_{10}: 33,14	AM_{11}: 48,71	H_{32}: 38,86	H_1: 38,86	AM_3: 38,57	AM_{11}: 48,71	AM_2: 40,57	AM_2: 38,43	AM_2: 37,14
H_{15}: 38,29	AM_{13}: 48,71	AM_{14}: 38,86	H_{10}: 39,14	AM_{12}: 38,57	AM_{13}: 48,71	AM_3: 40,57	AM_3: 38,43	AM_3: 37,14
H_9: 39,14	AM_{14}: 48,71	H_9: 39,71	H_{32}: 39,14	H_9: 39,86	AM_{14}: 48,71	AM_{12}: 40,57	AM_{12}: 38,43	AM_{12}: 37,14
H_{25}: 39,57	AM_{15}: 48,71	H_{15}: 43,71	H_9: 39,43	AM_{14}: 41,00	AM_{15}: 48,71	H_8: 43,00	AM_8: 40,86	H_9: 38,71
H_{20}: 40,14	H_{19}: 49,71	H_{25}: 43,71	AM_{14}: 39,86	AM_8: 41,14	H_{19}: 49,71	H_{15}: 43,86	AM_9: 40,86	AM_8: 40,43
H_3: 42,29	H_{21}: 49,71	GA_{T2}: 44,57	AM_7: 40,29	AM_9: 41,14	H_{21}: 49,71	H_{25}: 43,86	H_{15}: 44,00	AM_9: 40,43
AM_{12}: 43,00	H_{22}: 49,71	AM_7: 44,57	GA_{T2}: 41,86	AM_7: 41,57	H_{22}: 49,71	AM_7: 44,57	H_{25}: 44,00	AM_7: 42,86
AM_2: 43,57	H_{23}: 49,71	AM_8: 44,57	H_{15}: 44,43	H_4: 43,43	H_{23}: 49,71	AM_8: 44,57	AM_7: 44,43	H_{15}: 43,43
AM_3: 43,57	AM_2: 53,00	AM_9: 44,57	H_{25}: 44,43	H_{15}: 44,29	AM_2: 53,00	AM_9: 44,57	GA_{T2}: 44,71	H_{25}: 44,14
H_{19}: 43,57	AM_3: 53,00	H_{19}: 48,29	H_{19}: 48,43	H_3: 44,57	AM_3: 53,00	GA_{T2}: 48,00	H_{19}: 48,14	H_4: 44,29
H_{21}: 43,57	AM_7: 53,00	H_{21}: 48,29	H_{21}: 48,43	H_{20}: 44,57	AM_7: 53,00	H_{19}: 48,29	H_{21}: 48,14	H_3: 45,57
H_{22}: 43,57	AM_8: 53,00	H_{22}: 48,29	H_{22}: 48,43	H_{25}: 45,29	AM_8: 53,00	H_{21}: 48,29	H_{22}: 48,14	H_{20}: 45,57
H_{23}: 43,57	AM_9: 53,00	H_{23}: 48,29	H_{23}: 48,43	H_{19}: 49,71	AM_9: 53,00	H_{22}: 48,29	H_{23}: 48,14	H_{19}: 48,14
H_6: 44,71	AM_{12}: 53,00	H_8: 50,00	H_8: 50,71	H_{21}: 49,71	AM_{12}: 53,00	H_{23}: 48,29	H_8: 48,57	H_{21}: 48,14
AM_{16}: 46,86	H_3: 54,00	H_4: 51,00	H_4: 50,86	H_{22}: 49,71	H_3: 54,00	H_4: 51,14	H_4: 49,57	H_{22}: 48,14
AM_9: 47,00	H_4: 54,00	H_3: 54,00	H_3: 52,71	H_{23}: 49,71	H_4: 54,00	H_3: 54,00	H_3: 53,00	H_{23}: 48,14
AM_8: 47,29	H_6: 54,00	H_6: 54,00	H_{20}: 52,71	H_8: 51,71	H_6: 54,00	H_6: 54,00	H_{20}: 53,00	H_8: 51,14
AM_7: 48,14	H_{20}: 54,00	H_{20}: 54,00	H_6: 54,00	H_6: 52,29	H_{20}: 54,00	H_{20}: 54,00	H_6: 54,00	H_6: 52,43
H_8: 53,71	AM_{16}: 54,00	AM_{16}: 54,00	AM_{16}: 54,00	AM_{16}: 53,86	AM_{16}: 54,00	AM_{16}: 54,00	AM_{16}: 54,00	AM_{16}: 53,71

APÊNDICE B

Dados da Métrica *Expense Médio*

Este apêndice apresenta as tabelas completas para a métrica de avaliação *expense* médio por programa, conforme a análise da Subseção 3.3.3 do Capítulo 3. As Tabelas B.1 e B.2 apresentam os valores de *expense* médio de cada programa para o conjunto completo das heurísticas que compõem o conjunto de terminais $T1$ e $T2$ respectivamente e os métodos de composição configurados com o mesmos. Os dados estão ordenados pela coluna *Total*.

Tabela B.1: *Proporção média de código investigado para encontrar todos os defeitos (Expense Médio) para as heurísticas que compõem o conjunto de terminais $T1$ e os métodos de composição a ele relacionados (GP_{T1} e GA_{T1})*

Método	P1 (4)	P2 (10)	P3 (28)	P4 (7)	P5 (9)	P6 (30)	P7 (19)	Total (107)
$\boldsymbol{GP}_{T1}$	9,44	21,84	10,38	5,77	30,69	21,77	12,69	16,43
$\boldsymbol{GA}_{T1}$	3,21	11,09	8,45	9,13	51,23	40,34	12,44	21,79
$\boldsymbol{AM}_{10}$	4,23	18,42	12,16	10,54	55,21	44,82	23,04	27,05
$\boldsymbol{AM}_{15}$	4,11	21,79	11,78	10,54	61,43	45,95	25,07	28,46
$\boldsymbol{AM}_{1}$	4,36	22,08	11,82	10,54	60,66	46,26	26,67	28,82
$\boldsymbol{AM}_{11}$	4,11	23,04	11,57	10,64	62,13	46,15	26,89	28,97
$\boldsymbol{AM}_{4}$	4,11	23,09	11,57	10,64	62,13	46,77	26,89	29,15
$\boldsymbol{AM}_{13}$	4,23	24,95	12,33	22,57	60,48	47,08	26,71	30,22
$\boldsymbol{H}_{33}$	3,46	26,87	10,78	59,03	54,09	44,26	22,35	30,25
$\boldsymbol{AM}_{14}$	7,18	25,19	14,78	11,58	60,48	46,36	28,65	30,42
$\boldsymbol{AM}_{5}$	5,77	26,27	12,99	34,95	62,70	48,97	18,51	30,65
$\boldsymbol{AM}_{6}$	5,77	27,83	13,65	35,89	69,01	49,79	22,52	32,50
$\boldsymbol{H}_{32}$	9,36	34,30	13,65	60,06	59,09	45,95	28,62	33,99
$\boldsymbol{AM}_{12}$	34,41	49,78	43,49	39,10	68,38	75,85	44,75	54,84
$\boldsymbol{AM}_{2}$	34,41	49,78	43,49	39,10	70,64	75,85	44,75	55,03
$\boldsymbol{AM}_{3}$	34,41	49,78	43,49	39,10	70,64	75,85	44,75	55,03
$\boldsymbol{AM}_{9}$	41,73	54,34	44,23	40,23	71,15	77,38	47,86	57,02
$\boldsymbol{AM}_{8}$	41,73	54,34	44,28	40,23	71,15	77,38	48,16	57,09
$\boldsymbol{AM}_{7}$	39,55	61,69	38,55	87,12	71,06	76,41	49,45	59,21
$\boldsymbol{AM}_{16}$	96,02	42,34	73,21	34,90	70,70	84,10	63,77	69,84

Tabela B.2: *Proporção média de código investigado para encontrar todos os defeitos (Expense Médio) para as heurísticas que compõem o conjunto de terminais $T2$ e os métodos de composição a ele relacionados (GP_{T2} e GA_{T2})*

Método	P1 (4)	P2 (10)	P3 (28)	P4 (7)	P5 (9)	P6 (30)	P7 (19)	Total(107)
GP_{T2}	20,43%	20,23%	9,06%	22,23%	40,05%	26,16%	19,09%	20,57%
H_{29}	3,46%	11,91%	8,14%	26,15%	46,40%	41,74%	16,36%	23,59%
H_{16}	3,46%	18,71%	9,19%	58,37%	46,66%	41,74%	16,53%	26,67%
GA_{T2}	9,81%	25,36%	13,48%	19,37%	33,65%	48,81%	25,10%	28,50%
H_{13}	4,11%	24,13%	10,58%	26,43%	57,12%	45,33%	24,51%	28,77%
H_{12}	3,59%	23,73%	10,14%	26,81%	59,36%	45,69%	25,12%	29,02%
H_{18}	3,59%	24,34%	10,05%	26,81%	59,71%	45,64%	24,90%	29,03%
H_{17}	3,59%	24,34%	10,05%	26,81%	59,71%	45,64%	25,07%	29,06%
H_{24}	3,46%	24,59%	9,89%	26,90%	60,74%	45,64%	25,12%	29,14%
H_{2}	3,59%	25,69%	11,23%	27,84%	60,57%	45,79%	27,84%	30,17%
H_{33}	3,46%	26,87%	10,78%	59,03%	54,09%	44,26%	22,35%	30,25%
H_{7}	3,72%	25,59%	11,52%	42,47%	59,09%	45,33%	24,26%	30,31%
H_{4}	9,88%	22,93%	13,71%	27,37%	54,87%	46,51%	27,19%	30,37%
H_{14}	7,82%	27,25%	12,57%	27,84%	59,27%	45,95%	28,36%	30,85%
H_{26}	6,16%	33,34%	13,27%	60,06%	59,09%	45,85%	25,94%	33,18%
H_{27}	6,16%	33,34%	13,27%	60,06%	59,09%	45,85%	25,94%	33,18%
H_{28}	6,16%	33,34%	13,27%	60,06%	59,09%	45,85%	25,94%	33,18%
H_{30}	6,16%	33,34%	13,27%	60,06%	59,09%	45,85%	25,94%	33,18%
H_{31}	6,16%	33,34%	13,27%	60,06%	59,09%	45,85%	25,94%	33,18%
H_{1}	9,23%	34,20	13,64%	60,06%	59,09%	45,95%	27,97%	33,86%
H_{32}	9,36%	34,30	13,65%	60,06%	59,09%	45,95%	28,62%	33,99%
H_{10}	9,36%	34,30	13,65%	60,06%	59,27%	45,95%	28,88%	34,05%
H_{11}	4,62%	29,39%	13,54%	35,70	66,76%	51,03%	33,49%	34,67%
H_{34}	3,59%	33,30	12,45%	60,06%	67,22%	46,92%	35,96%	35,63%
H_{5}	3,59%	31,05%	13,54%	36,64%	68,32%	52,67%	37,55%	36,16%
H_{9}	20,28%	37,01%	17,85%	60,16%	68,77%	48,62%	40,88%	39,50
H_{25}	27,87%	33,47%	28,79%	30,67%	74,04%	63,08%	59,01%	48,10
H_{15}	28,00%	32,52%	29,97%	30,67%	74,04%	63,18%	58,71%	48,30
H_{20}	48,39%	53,06%	42,25%	61,29%	34,09%	61,28%	44,84%	49,84%
H_{19}	36,59%	35,47%	36,14%	31,14%	74,04%	64,10%	58,97%	50,85%
H_{21}	36,59%	35,47%	36,14%	31,14%	74,04%	64,10%	58,97%	50,85%
H_{22}	36,59%	35,47%	36,14%	31,14%	74,04%	64,10%	58,97%	50,85%
H_{23}	36,59%	35,47%	36,14%	31,14%	74,04%	64,10%	58,97%	50,85%
H_{6}	38,52%	38,22%	38,54%	31,32%	74,04%	64,41%	62,81%	52,58%
H_{3}	53,52%	69,36%	52,14%	94,35%	34,09%	61,28%	46,17%	56,55%
H_{8}	100,00	97,25%	97,83%	76,58%	92,80	100,00	94,78%	96,11%