Uma abordagem para geração de dados de teste para o teste de mutação utilizando técnicas baseadas em busca

Francisco Carlos Monteiro Souza

Tese de Doutorado do Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional (PPG-CCMC)

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura: _____

Francisco Carlos Monteiro Souza

Uma abordagem para geração de dados de teste para o teste de mutação utilizando técnicas baseadas em busca

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Doutor em Ciências – Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. Márcio Eduardo Delamaro

Coorientador: Prof. Dr. Michael Papadakis

USP – São Carlos
Agosto de 2017

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo(a) autor(a)

S634u Souza, Francisco Carlos Monteiro
Uma abordagem para geração de dados de teste para o teste de mutação utilizando técnicas baseadas em busca / Francisco Carlos Monteiro Souza; orientador Márcio Eduardo Delamaro; coorientador Michael Papadakis. - São Carlos - SP, 2017.
149 p.

Tese (Doutorado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2017.

1. Teste de Software. 2. Teste de Software baseado em Busca. 3. Teste de Mutação. 4. Geração de Dados de Teste. 5. Mutantes Equivalentes. I. Delamaro, Márcio Eduardo, orient. II. Papadakis, Michael, coorient. III. Título.

Francisco Carlos Monteiro Souza

**An approach for test data generation in mutation testing
using search-based techniques**

Doctoral dissertation submitted to the Instituto de
Ciências Matemáticas e de Computação – ICMC-
USP, in partial fulfillment of the requirements for the
degree of the Doctorate Program in Computer Science
and Computational Mathematics. *FINAL VERSION*

Concentration Area: Computer Science and
Computational Mathematics

Advisor: Prof. Dr. Márcio Eduardo Delamaro

Co-advisor: Prof. Dr. Michael Papadakis

USP – São Carlos

August 2017

*Este trabalho é dedicado a minha mãe Marlene e
ao meu pai Carlinhos que sempre apoiaram minhas decisões e
a memória do meu amado avô Dr. Raimundo Rocha que sem dúvida
era uma das pessoas que mais torcia por mim.*

AGRADECIMENTOS

Primeiramente agradeço a Deus pelo dom da vida, inteligência e força para seguir esse longo caminho em busca de sabedoria.

A minha mãe Marlene pela excelente educação, amor e por estar comigo em todas as horas, ao meu pai Carlinhos pelo amor, amizade e por me ensinar o sentido da honestidade e simplicidade de um verdadeiro homem, aos meus irmãos, que apesar da distância sempre enviaram vibrações positivas e aos meus familiares por todas palavras de apoio.

Um agradecimento especial para minha pequena família (ainda em construção). À minha noiva Alinne, que desde 2009 está ao meu lado compartilhando todos os momentos, seja de alegria ou de tristeza. Agradeço pelo amor incondicional, amizade, paciência, incentivo e por sempre apoiar minhas decisões, por mais malucas que pareçam. Ao nosso cãozinho Wally que nos últimos meses tem mostrado o verdadeiro significado de um amor puro.

Aos meus amigos de São Carlos pela apoio e parceria no ambiente de trabalho e por todos momentos de descontração, em especial a Livia, Steve e Wil que nunca pouparam esforços para ajudar, seja no trabalho ou em problemas pessoais. Aos meus amigos de Belém, em especial ao meu amigo André pela amizade verdadeira, por sempre está por perto e principalmente por torcer e comemorar minhas vitórias como se fossem suas.

Ao meu orientador Márcio Delamaro pela paciência e seus ensinamentos. Obrigado pelos momentos produtivos no trabalho e pela amizade fora da USP. Ao meu coorientador Mike Papadakis por todo auxílio necessário para concluir a pesquisa.

A todos os funcionários do ICMC, em especial à Ana Carolina Murata e ao Leonardo Miyake pelo apoio acadêmico.

Um agradecimento à todos professores do ICMC que tiveram papel importantíssimo no decorrer do doutorado e da pesquisa.

Ao CNPq e CAPES pelo apoio financeiro e à USP pela oportunidade.

Enfim, sou grato a todos aqueles que direta ou indiretamente contribuíram para meu desenvolvimento pessoal e profissional.

“I get by with a little help from my friends.”
(John Lennon)

RESUMO

SOUZA, F. C. M. **Uma abordagem para geração de dados de teste para o teste de mutação utilizando técnicas baseadas em busca.** 2017. 149 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2017.

O teste de mutação é um critério de teste poderoso para detectar falhas e medir a eficácia de um conjunto de dados de teste. No entanto, é uma técnica de teste computacionalmente cara. O alto custo provém principalmente do esforço para gerar dados de teste adequados para matar os mutantes e pela existência de mutantes equivalentes. Nesse contexto, o objetivo desta tese é apresentar uma abordagem chamada de *Reach, Infect and Propagation to Mutation Testing (RIP-MuT)* que visa gerar dados de teste e sugerir mutantes equivalentes. A abordagem é composta por dois módulos: (i) uma geração automatizada de dados de teste usando subida da encosta e um esquema de *fitness* de acordo com as condições de alcançabilidade, infecção e propagação (RIP); e (ii) um método para sugerir mutantes equivalentes com base na análise das condições RIP durante o processo de geração de dados de teste. Os experimentos foram conduzidos para avaliar a eficácia da abordagem RIP-MuT e um estudo comparativo com o algoritmo genético e testes aleatórios foi realizado. A abordagem RIP-MuT obteve um escore médio de mutação de 18,25 % maior que o AG e 35,93 % maior que o teste aleatório. O método proposto para detecção de mutantes equivalentes se mostrou viável para redução de custos relacionado a essa atividade, uma vez que obteve uma precisão de 75,05% na sugestão dos mutantes equivalentes. Portanto, os resultados indicam que a abordagem gera dados de teste adequados capazes de matar a maioria dos mutantes em programas C e, também auxilia a identificar mutantes equivalentes corretamente.

Palavras-chave: Teste de Software, Teste de Software baseado em Busca, Teste de Mutação, Geração de Dados de Teste, Mutantes Equivalentes.

ABSTRACT

SOUZA, F. C. M. **An approach for test data generation in mutation testing using search-based techniques**. 2017. 149 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2017.

Mutation Testing is a powerful test criterion to detect faults and measure the effectiveness of a test data set. However, it is a computationally expensive testing technique. The high cost comes mainly from the effort to generate adequate test data to kill the mutants and by the existence of equivalent mutants. In this thesis, an approach called *Reach, Infect and Propagation to Mutation Testing* (RIP-MuT) is presented to generate test data and to suggest equivalent mutants. The approach is composed of two modules: (i) an automated test data generation using hill climbing and a fitness scheme according to Reach, Infect, and Propagate (RIP) conditions; and (ii) a method to suggest equivalent mutants based on the analyses of RIP conditions during the process of test data generation. The experiments were conducted to evaluate the effectiveness of the RIP-MuT approach and a comparative study with a genetic algorithm and random testing. The RIP-MuT approach achieved a mean mutation score of 18.25% higher than the GA and 35.93% higher than random testing. The proposed method for detection of equivalent mutants demonstrate to be feasible for cost reduction in this activity since it obtained a precision of 75.05% on suggesting equivalent mutants. Therefore, the results indicate that the approach produces effective test data able to strongly kill the majority of mutants on C programs, and also it can assist in suggesting equivalent mutants correctly.

Keywords: Software Testing, Search-based Software Testing, Mutation Testing, Test Data Generation, Equivalent Mutants.

LISTA DE ILUSTRAÇÕES

Figura 1 – Modelo V descrevendo o paralelismo entre as atividades de desenvolvimento e teste de software	34
Figura 2 – Exemplo de GFC para o programa <i>identifier</i>	37
Figura 3 – Exemplo de mutante	40
Figura 4 – Exemplo de Mutante Equivalente	41
Figura 5 – Processo genérico do teste de mutação	42
Figura 6 – Algoritmos baseados em busca	50
Figura 7 – Processo genérico de geração de dados de teste baseada em busca	53
Figura 8 – Representação de um cromossomo	54
Figura 9 – Probabilidade dos indivíduos de acordo com o <i>fitness</i> utilizando o método de seleção roleta	56
Figura 10 – Representação de cruzamento com um ponto de corte	56
Figura 11 – Exemplo da operação de mutação	56
Figura 12 – <i>String</i> de busca das questões de pesquisa	67
Figura 13 – <i>String</i> de busca com os sinônimos invertidos	69
Figura 14 – Processo de seleção dos estudos	70
Figura 15 – Evidência dos estudos por país	71
Figura 16 – Evidência dos estudos de acordo com as suas respectivas técnicas	71
Figura 17 – RIP-MuT: uma abordagem para geração de dados de teste e sugestão de mutantes equivalentes	87
Figura 18 – Exemplo de alcançabilidade para função de <i>fitness</i> RD e infecção para função de <i>fitness</i> MD	88
Figura 19 – Exemplo da condição de propagação para o programa cal	89
Figura 20 – Programa Exemplo	92
Figura 21 – Medida <i>Approach Level</i>	94
Figura 22 – Exemplo da utilização da função de <i>fitness</i> ID ₂	98
Figura 23 – Processo da geração de dados de teste utilizando o esquema de <i>fitness</i> proposto	99
Figura 24 – Sugerindo um mutante como equivalente utilizando a premissa A ₂ para o programa cal	102
Figura 25 – Escore de Mutação de cada esquema de <i>fitness</i>	116
Figura 26 – Escore de mutação forte das funções de <i>fitness</i> RD & MD & ID, RD & MD e RD	118

Figura 27 – Número de mutantes mortos pelas funções de <i>fitness</i> para um determinado tempo	119
Figura 28 – Comparação entre FS, AG e Teste aleatório para mutantes L-R	121
Figura 29 – Comparação entre FS, AG e Teste aleatório para mutantes derivados dos 78 operadores	122
Figura 30 – Número de dados de teste para FS, AG e Teste aleatório para os mutantes L-R	126
Figura 31 – Sugestões verdadeiras para mutantes equivalentes e difíceis de serem mortos	127

LISTA DE ALGORITMOS

Algoritmo 1 – Algoritmo Genético	57
Algoritmo 2 – Otimização por enxame de partículas	60
Algoritmo 3 – Têmpera Simulada	61
Algoritmo 4 – Subida da Encosta	62

LISTA DE TABELAS

Tabela 1 – Cálculo da porcentagem do <i>fitness</i> (roleta)	55
Tabela 2 – Associação do SA com a Termodinâmica	61
Tabela 3 – Fontes de busca	67
Tabela 4 – Categorização das técnicas identificadas	78
Tabela 5 – Análise da qualidade dos estudos primários	79
Tabela 6 – Análise dos programas utilizados nos estudos primários	80
Tabela 7 – Análise do tempo e iterações para a geração dos dados de teste	81
Tabela 8 – Fórmulas para predicados lógicos e relacionais	95
Tabela 9 – Sujeitos do experimento	112
Tabela 10 – Operadores lógicos e relacionais	115
Tabela 11 – Número de mutantes mortos por cada esquema de <i>fitness</i>	116
Tabela 12 – Adequação dos dados de teste gerados para mutantes L-R e aplicados nos mutantes derivados de 78 operadores	117
Tabela 13 – Número de mutantes mortos por cada função de <i>fitness</i>	119
Tabela 14 – Resultados do teste de <i>Wilcoxon</i> sobre a diferença entre os escores de mutação obtidos pelo esquema e cada função de <i>fitness</i>	120
Tabela 15 – Número de Mutantes L-R mortos por FS, AG e teste aleatório	122
Tabela 16 – Adequação dos dados de teste gerados para mutantes L-R (FS, AG e Teste Aleatório) e aplicados nos mutantes derivados de 78 operadores	123
Tabela 17 – Resultados do teste <i>Mann-Whitney U</i> relacionados ao escore de mutação obtidos por <i>FS₃</i> , <i>AG</i> e <i>TA</i>	124
Tabela 18 – Tempo de geração para cada esquema de <i>fitness</i>	125
Tabela 19 – Resultados do teste <i>Mann-Whitney U</i> relacionados à quantidade de dados de teste gerados por <i>FS₃</i> , <i>AG</i> e <i>TA</i>	126
Tabela 20 – Sugestão de mutantes equivalentes baseada nas condições RIP	128
Tabela 21 – Sugestão de mutantes equivalentes utilizando as métricas de precisão e revo- cação	129

LISTA DE ABREVIATURAS E SIGLAS

<i>P</i>	precisão
<i>R</i>	revocação
<i>FS_{eq}</i>	<i>False Suggestion equivalent</i>
<i>FS_{hk}</i>	<i>False Suggestion hard-to-kill</i>
<i>TS_{eq}</i>	<i>True Suggestion equivalent</i>
<i>TS_{hk}</i>	<i>True Suggestion hard-to-kill</i>
<i>ACO</i>	<i>Ant Colony Optimization</i>
<i>AUSTIN</i>	<i>AUgmented Search-based TestINg</i>
<i>AVM</i>	<i>Alternating Variable Method</i>
<i>BA</i>	<i>Bacteriological Algorithms</i>
<i>CBT</i>	<i>Constraint-Based Testing</i>
<i>CSP</i>	<i>Constraint Satisfaction Problem</i>
<i>DSE</i>	<i>Dynamic Symbolic Execution</i>
<i>DTC</i>	<i>Distinguishing Test Cases</i>
<i>EBSE</i>	<i>Evidence-Based Software Engineering</i>
<i>ECFG</i>	<i>Enhanced Control Flow Graph</i>
<i>FOMs</i>	<i>First Order Mutants</i>
<i>FS</i>	<i>False Suggestion</i>
<i>GA</i>	<i>Genetic Algorithm</i>
<i>HC</i>	<i>Hill Climbing</i>
<i>HOMs</i>	<i>Higher Order Mutants</i>
<i>ID</i>	<i>Impact Distance</i>
<i>IIA</i>	<i>Immune Inspired Algorithm</i>
<i>IIRM</i>	<i>Improved Iterative Relaxation Method</i>
<i>MD</i>	<i>Mutation Distance</i>
<i>MS</i>	<i>Mutation Score</i>
<i>PICO</i>	<i>Population, Intervention, Comparison e Outcomes</i>
<i>PROTEUM</i>	<i>Program Testing Using Mutants</i>
<i>PSO</i>	<i>Particle Swarm Optimization</i>
<i>PUT</i>	<i>Program Under Test</i>
<i>RD</i>	<i>Reach Distance</i>

<i>RIP-MuT</i>	<i>Reach, Infect and Propagation to Mutation Testing</i>
<i>SA</i>	<i>Simulated Annealing</i>
<i>SBST</i>	<i>Search-Based Software Testing</i>
<i>SHOM</i>	<i>Strong Higher Order Mutation</i>
<i>SLR</i>	<i>Systematic Literature Review</i>
<i>SMS</i>	<i>Systematic Mapping Study</i>
<i>TS</i>	<i>True Suggestion</i>
AG	Algoritmo Genético
CE	Critérios de Exclusão
CI	Critérios de Inclusão
ES	Engenharia de Software
FS	esquema de <i>fitness</i>
GFC	Grafo de Fluxo de Controle
GQM	<i>Goal/Question/Metric</i>
L-R	operadores lógicos e relacionais
MS	Mapeamento Sistemático
QPs	Questões de Pesquisas
RIP	<i>Reachability, Infection e Propagation</i>
RSL	Revisão Sistemática da Literatura
TA	Teste Aleatório
VV&T	Verificação, Validação e Teste

SUMÁRIO

1	INTRODUÇÃO	25
1.1	Motivações	27
1.2	Objetivos	28
1.3	Estrutura da Tese	29
2	FUNDAMENTAÇÃO TEÓRICA	31
2.1	Considerações Iniciais	31
2.2	Conceitos e Terminologias	32
2.3	Técnicas de Teste	35
2.3.1	<i>Teste Funcional</i>	36
2.3.2	<i>Teste Estrutural</i>	37
2.3.3	<i>Teste de Mutação</i>	38
2.4	Geração de Dados de Teste	45
2.4.1	<i>Geração Aleatória</i>	47
2.4.2	<i>Geração com Execução Simbólica</i>	47
2.4.3	<i>Geração com Execução Dinâmica</i>	48
2.4.4	<i>Teste baseado em restrição</i>	48
2.5	Teste de Software Baseado em Busca	48
2.5.1	<i>Algoritmo Genético</i>	53
2.5.2	<i>Otimização por Enxame de Partículas</i>	57
2.5.3	<i>Têmpera Simulada</i>	59
2.5.4	<i>Subida da Encosta</i>	61
2.6	Considerações Finais	62
3	MAPEAMENTO SISTEMÁTICO	65
3.1	Considerações Iniciais	65
3.2	Planejamento do Mapeamento Sistmático	66
3.2.1	<i>Estratégia de Busca</i>	67
3.2.2	<i>Critérios para Seleção dos Estudos</i>	68
3.2.3	<i>Extração dos Dados</i>	68
3.3	Execução do Mapeamento Sistemático	69
3.4	Síntese e Análise dos Dados	70
3.4.1	<i>Discussão dos Estudos</i>	77

3.5	Ameaças à Validade	81
3.6	Considerações Finais	82
4	RIP-MUT: UMA ABORDAGEM PARA GERAÇÃO DE DADOS DE TESTE E SUGESTÃO DE MUTANTES EQUIVALENTES	85
4.1	Considerações Iniciais	85
4.2	Abordagem RIP-MuT	86
4.3	RIP-MuT: Geração de dados de teste	92
4.3.1	<i>Funções de fitness: RD, MD e ID</i>	93
4.4	RIP-MuT: Sugestão de mutantes equivalentes	100
4.5	Considerações Finais	102
5	AVALIAÇÃO EXPERIMENTAL	105
5.1	Considerações Iniciais	105
5.2	Planejamento	105
5.2.1	<i>Definição dos objetivos</i>	110
5.2.2	<i>Definição das variáveis</i>	111
5.2.3	<i>Seleção dos sujeitos</i>	111
5.3	Condução	111
5.4	Análise dos Dados	114
5.4.1	<i>Análise Descritiva</i>	114
5.5	Ameaças à Validade	130
5.6	Considerações Finais	131
6	CONSIDERAÇÕES FINAIS	133
6.1	Contribuições	135
6.2	Limitações	137
6.3	Trabalhos Futuros	138
	REFERÊNCIAS	139

INTRODUÇÃO

A Engenharia de Software (ES) é a criação e o uso de princípios sólidos da engenharia a fim de se obter um software que seja confiável, funcione de maneira eficiente e tenha baixo custo. Processos de ES fornecem atividades com o objetivo de estabelecer a construção de softwares com essas características. Tais atividades abrangem o planejamento, análise de requisitos, *projeto*, implementação, teste e manutenção ([PRESSMAN, 2010](#)).

Apesar de o processo de desenvolvimento de software envolver diversas atividades, ainda assim, erros no produto final podem ocorrer, uma vez que qualquer tarefa realizada por um humano é suscetível a erros. Nesse contexto, a atividade de teste é indispensável para a identificação e correção de possíveis defeitos que ainda possam existir no software. [Dijkstra \(1979\)](#) afirma que os testes podem mostrar a presença de defeitos, porém não sua ausência. É evidente que o custo da alteração do software cresce à medida que um defeito é detectado, logo, é importante a execução de casos de testes desde o início do projeto.

O teste de software é uma atividade fundamental para alcançar e avaliar a qualidade de um software. No entanto, o teste de software geralmente constitui mais da metade do custo do processo global. Assim, a automação parcial ou completa é a melhor maneira de reduzir esses custos e aumentar a eficácia dos testes ([MYERS; SANDLER, 2004](#)).

A atividade de teste se refere à produção e execução de um conjunto de dados de teste no qual o comportamento dos programas é analisado para detecção de falhas. Deste modo, esta atividade concentra-se na geração de entradas com base em critérios de teste específicos para exibir comportamentos distintos entre a saída do programa e os seus requisitos de software. Os critérios de teste fornecem um conjunto de requisitos para orientar a geração e a avaliação do conjunto de dados de teste. Existem diferentes tipos de técnicas de teste, como por exemplo, funcional, estrutural e baseada em defeitos. Dentre estas, o teste de mutação, um dos critérios baseados em defeitos, tem recebido um interesse crescente na literatura por proporcionar um nível consideravelmente elevado de confiança ([HARMAN; JIA; LANGDON, 2011](#)).

O teste de mutação é um critério baseado em defeitos que visa detectar falhas e medir a eficácia dos conjuntos de testes. Nesse critério, alterações são injetadas no código do programa em teste (do inglês, *Program Under Test (PUT)*) para obter novas versões, denominadas “mutantes”. Os defeitos são reproduzidos por meio de um conjunto de operadores de mutação, regras que criam versões defeituosas do programa original e que podem ser aplicadas em várias instruções do código fonte. Uma vez produzido um conjunto de mutantes, o objetivo é gerar um conjunto de dados de teste capaz de revelar as falhas, isto é, distinguir as saídas dos programas mutantes das do programa original. Se um mutante produz uma saída diferente em relação ao programa original para um determinado dado de teste, então, o mutante é definido como “morto”, o que significa que o defeito foi detectado. Por outro lado, se um dado de teste é incapaz de detectar o defeito inserido no mutante, ele é chamado de “vivo” (DEMILLO; LIPTON; SAYWARD, 1978).

No entanto, existe um tipo específico de mutante que produz a mesma saída que o programa original para todos os testes possíveis. Esse mutante é denominado como “equivalente”, isto é, não existe um dado de teste capaz de matá-lo. Para aqueles mutantes que permanecem vivos após a execução dos casos de teste, não é possível afirmar a priori se são equivalentes ou se o conjunto de dados de teste não é adequado para matá-los, sem antes examinar a composição interna do código-fonte dos programas. Por ser, em geral, um problema indecidível, identificar os mutantes equivalentes impossibilita automatizar completamente o teste de mutação.

De um modo geral, para distinguir o comportamento entre o programa original e o mutante, os dados de um teste devem satisfazer três condições conhecidas como Alcançabilidade, Infecção e Propagação (do inglês, *Reachability, Infection e Propagation – RIP conditions*).

O teste de mutação pode ser classificado em três categorias: i) Mutação Forte (*Strong Mutation*): refere-se ao processo original do teste de mutação em que o mutante é morto quando produz uma saída diferente do programa original; ii) Mutação Fraca (*Weak Mutation*): nesta categoria, em vez de verificar os resultados após a execução completa do programa, o estado do mutante é comparado imediatamente após o ponto de execução da instrução mutada; iii) Mutação Firme (*Firm Mutation*): o estado de comparação está localizado em estados intermediários, entre a execução do ponto de mutação e a saída final.

Apesar de testes de mutação serem amplamente considerados como muito eficazes e poderosos, o custo imposto por sua utilização é elevado. Normalmente, o custo provém da quantidade de mutantes produzidos, do número de dados de teste necessário para matar os mutantes e da dificuldade em identificar mutantes equivalentes. Entre essas questões, a geração de dados de teste é uma das mais críticas, intensivas em mão-de-obra e suscetíveis a erros, principalmente se for feita manualmente (DEMILLO; OFFUTT, 1991). Por estas razões, a automação da atividade de geração de dados de teste é um passo importante para reduzir os custos no teste de mutação, aumentar a confiabilidade dos testes e adicionalmente reduzir os esforços para identificar mutantes equivalentes.

1.1 Motivações

No cenário de desenvolvimento de software, as atividades de teste têm recebido muita atenção por parte dos desenvolvedores e dos testadores. No entanto, a automação dessas atividades ainda é um desafio, especialmente em sistemas de software que possuem diversas restrições e manipulam uma grande quantidade de dados.

Dentre as diversas pesquisas realizadas na área, o uso do teste de mutação vem crescendo nos últimos anos e despertando o interesse dos pesquisadores. Com isso, estudos recentes como [Fraser e Arcuri \(2014\)](#), [Papadakis, Delamaro e Traon \(2014\)](#), [Papadakis e Malevris \(2013\)](#), [Papadakis e Malevris \(2012\)](#), [Harman, Jia e Langdon \(2011\)](#) foram conduzidos para a avaliação desse critério em aplicações de diferentes contextos no processo de desenvolvimento de software.

Dado o atual estado da arte, é possível afirmar que existem diversos trabalhos nessa linha de pesquisa, entretanto, trata-se de um assunto de interesse e que pode ser amplamente explorado, pois ainda não atingiu um grau elevado de maturidade e aplicabilidade. Considerando o teste de mutação como um critério de teste bastante eficiente, a automatização da geração de dados de teste para satisfazer esse critério possibilita a obtenção de conjuntos de teste de alta qualidade. Ainda que conjuntos de dados de teste possam ser gerados e selecionados manualmente por um testador, essa prática é difícil e propensa a erros, o que indica a necessidade de automatizar essa etapa.

Para que a automação de teste seja bem sucedida, a geração de dados de teste deve produzir testes adequados ao teste de mutação, isto é, um conjunto de testes que seja capaz de matar todos os mutantes não-equivalentes gerados para um determinado programa e colateralmente reduzir os custos com a identificação de mutantes equivalentes. A identificação de mutantes equivalentes pode ser um problema desafiador nesse contexto, pois mesmo que a geração de dados seja realizada de forma automática, é preciso reconhecer que existem requisitos de teste que não podem ser satisfeitos. Os esforços necessários para a geração de dados de teste satisfazer requisitos infactíveis (identificar mutantes equivalentes) representa elevar o custo do processo. Portanto, combinado com técnicas para geração de dados de teste é necessário que sejam desenvolvidas estratégias na tentativa de identificar grande parte dos mutantes equivalentes, também de forma automatizada.

Embora a atividade de geração de dados de teste seja conhecida por ser complexa, diversas técnicas têm sido investigadas para automatizá-la. Algumas técnicas convencionais são limitadas para solucionar determinadas questões, pois cada programa possui um tipo específico de entrada e um conjunto de dados de testes adequado desconhecido. Para solucionar tais questões e complementar as técnicas convencionais surgiu o teste de software baseado em busca (do inglês, *Search-Based Software Testing (SBST)*), que consiste no uso de técnicas de busca e meta-heurísticas para automatizar a atividade de teste ([MCMINN, 2004](#)).

Com o surgimento do SBST, problemas relacionados ao teste de software puderam

ser otimizados utilizando algoritmos de busca e meta-heurísticas. A partir de um conjunto de evidências empíricas foi possível demonstrar que as técnicas utilizadas em SBST são robustas e bem sucedidas, podendo ser aplicadas em diversos domínios (MCMINN, 2004). Assim, o SBST consiste em reformular problemas de teste para problemas de busca, ou seja, identificar soluções ótimas ou próximas ao ótimo em um espaço de busca, guiado por uma função de avaliação que distingue as melhores dentre todas as soluções. Em função da sua robustez, o conceito de SBST vem sendo empregado em diferentes domínios de teste de software, como por exemplo, no contexto de teste de mutação (BOTTACI, 2001; ZHAN; CLARK, 2005; SRIDHARAN; NAMIN, 2010; DOMÍNGUEZ-JIMÉNEZ *et al.*, 2011). Desse modo, o SBST é visto como uma alternativa para tentar solucionar os problemas relacionados ao teste de mutação citados anteriormente.

1.2 Objetivos

O objetivo geral da presente tese é apresentar uma abordagem denominada de *Reach, Infect and Propagation to Mutation Testing (RIP-MuT)* que aplica técnicas de busca no teste de mutação para gerar dados de teste capazes de matar os mutantes não-equivalentes. O propósito da abordagem é combinar mutação fraca, firme e forte com técnicas de busca guiadas por uma função objetivo baseada nas condições RIP. Em adição, um método é desenvolvido para auxiliar na detecção de mutantes equivalentes durante o processo de geração dos dados de teste.

Para sistematizar a condução desse projeto e para que o objetivo geral seja atingido, alguns objetivos específicos foram definidos:

- investigar e selecionar diferentes critérios de teste para auxiliar no processo de geração de dados de teste;
- investigar e selecionar técnicas de busca e/ou meta-heurísticas presentes na literatura que possam ser aplicadas na abordagem proposta;
- desenvolver uma técnica definida juntamente com o teste de mutação para geração dos dados de teste;
- desenvolver a ferramenta AUSTIN-RIP para auxiliar a geração de dados de teste no teste de mutação;
- desenvolver um método para auxiliar na detecção de mutantes equivalentes no processo de geração dos dados de teste; e
- realizar uma avaliação experimental da abordagem.

Portanto, a abordagem busca auxiliar na atividade de geração de dados de teste, uma vez que na maioria dos casos a quantidade de dados gerados pode ser proibitiva para o projeto de teste, levando em consideração variáveis como tempo e custo para executá-los nos mutantes.

Além disso, um conjunto de dados de teste adequado possibilita a identificação dos mutantes equivalentes com mais rapidez, visto que a detecção manual envolve esforço e custo adicional ao teste de mutação.

1.3 Estrutura da Tese

Esta tese está organizada em seis capítulos. No primeiro capítulo, estão evidenciados a contextualização, a motivação e, os objetivos do trabalho.

No Capítulo 2, são detalhados os principais conceitos envolvendo teste de software, técnicas e critérios de teste. Além disso, é realizada uma contextualização sobre a geração de dados de teste e o teste de software baseado em busca, destacando os aspectos conceituais e singularidades das técnicas de busca com a finalidade de facilitar a compreensão com relação à tese. Por fim, é realizada uma contextualização sobre as técnicas baseadas em busca, em que seus conceitos e suas singularidades são detalhados.

No Capítulo 3, é descrito o mapeamento sistemático realizado com o objetivo de identificar técnicas existentes para geração de dados de teste no contexto de teste de mutação. Adicionalmente, no presente mapeamento também são apresentadas as principais observações e questões em aberto que nortearam a tese.

No Capítulo 4 é apresentada uma abordagem automatizada para geração de dados de teste. Nesse capítulo também é descrito o método que analisa as condições RIP para auxiliar a sugestão de mutantes equivalentes.

No Capítulo 5 é detalhado o planejamento, a condução e a análise dos dois experimentos que visam verificar a eficácia e o custo computacional da abordagem para geração dos dados de teste, a efetividade do método para sugerir mutantes equivalentes, bem como os resultados obtidos para ambas avaliações experimentais. E, por fim, no Capítulo 6, são descritas as conclusões da pesquisa, suas principais contribuições, limitações, bem como os trabalhos futuros que poderão ser conduzidos para continuar o presente trabalho.

FUNDAMENTAÇÃO TEÓRICA

2.1 Considerações Iniciais

O teste de software pode ser compreendido como um processo de execução de um programa (ou sistema) com a finalidade de revelar defeitos (MYERS; SANDLER, 2004) e estabelecer se um determinado programa está em conformidade com as especificações definidas para o seu desenvolvimento. Assim, um teste bem sucedido é aquele que gera um conjunto de casos de teste capaz de detectar defeitos no PUT visando o aumento da qualidade do produto.

Ao longo dos anos, diferentes técnicas de geração de dados de teste foram apresentadas, como por exemplo geração aleatória (HAMLET; TAYLOR, 1990), geração com execução simbólica (HOWDEN, 1977), entre outras. Com os conceitos do SBST e a utilização de técnicas de busca e meta-heurísticas, novas investigações nesse contexto começaram a surgir.

Em geral, a maioria das limitações que dificultam a automatização do teste de software podem ser tratadas como problemas de busca e otimização, que são caracterizados por alta complexidade e uma exorbitante quantidade de soluções. Nesse sentido, a geração de dados de teste pode ser considerada um típico problema de busca, pois deseja-se selecionar um subconjunto de dados de teste que seja adequado a um critério de teste a partir de um conjunto de teste com uma grande quantidade possibilidades.

Nesse contexto, o intuito desse capítulo é apresentar os conceitos fundamentais referentes a Teste de Software para elucidar o projeto de doutorado realizado. Na Seção 2.2 são sintetizados os principais conceitos e terminologias de teste de software. Na Seção 2.3, as técnicas e critérios de teste são apresentadas.

Além disso, neste capítulo, também são abordados os conceitos referentes à geração de dados de teste e algumas técnicas existentes para o contexto (Seção 2.4). E por fim, na Seção 2.5 são detalhados os aspectos conceituais de SBST, incluindo algumas técnicas de busca e meta-heurísticas que são amplamente utilizadas na literatura.

2.2 Conceitos e Terminologias

O processo de desenvolvimento de software é composto por um conjunto de atividades que podem envolver ambiguidades, suposições e comunicação humana imperfeita. Cada manutenção realizada em uma parte do software, cada nova funcionalidade, cada tentativa de corrigir um defeito, possibilita a introdução de novos defeitos. A cada defeito introduzido, o risco de que o software não poderá corresponder à sua especificação aumenta (JENKINS, 2008).

Nesse contexto, para que tais erros não perdurem e consequentemente o software atinja um grau de qualidade aceitável, atividades conhecidas como Verificação, Validação e Teste (VV&T) têm sido utilizadas, a fim de garantir que o software seja desenvolvido em conformidade com o que foi especificado. Dentre as técnicas de VV&T, o teste de software é considerado fundamental no contexto de ES. Portanto, é evidente a importância do teste de software de maneira planejada e sistemática para reduzir ao máximo a incidência de defeitos no produto (DELAMARO; MALDONADO; JINO, 2007; BERTOLINO, 2007).

Teste de software é composto por uma série de atividades desenvolvidas para fazer com que o software realize as tarefas que foram projetadas previamente. Um produto de software deve ser previsível e consistente, não oferecendo nenhuma ação inesperada para os usuários. O teste ajuda a garantir que o software atenda seus requisitos, mas não é garantia de qualidade. A principal finalidade do teste é a identificação de defeitos em sistemas de software. Um defeito pode ser introduzido em qualquer fase do desenvolvimento ou manutenção do sistema, podendo os erros ser de enganos, omissões ou mesmo intenções equivocadas por parte dos desenvolvedores (MCGREGOR; SYKES, 2001).

Com intuito de padronização e melhor compreensão do teste de software o documento IEEE (1990) conceitua e distingue alguns termos que ainda podem ser interpretados erroneamente por parte dos pesquisadores.

- **defeito (*fault*):** passo, processo ou definição de dados incorretos, como uma instrução ou um comando escrito de maneira incorreta;
- **engano (*mistake*):** ação humana que produz um resultado errôneo, como uma ação incorreta realizada pelo programador;
- **erro (*error*):** diferença entre o valor obtido e o valor esperado, ou seja, qualquer estado intermediário que apresente resultados incorretos ou que não funcione corretamente durante a execução do programa;
- **falha (*failure*):** produção de uma saída incorreta do programa em relação à especificação, ou seja, uma falha ocorre quando um programa não se comporta conforme foi planejado, manifestando um erro;
- **dado de teste:** consiste em uma de entrada para a execução do programa em teste;

- **caso de teste:** consiste em um par composto por um valor de entrada e a respectiva saída esperada do programa; e
- **conjunto de teste ou conjunto de casos de teste:** formado pelos casos de teste efetivamente usados na atividade de teste.

Durante o processo de desenvolvimento de software, é necessário que sejam consideradas algumas etapas do teste de software. Embora a maioria das técnicas de teste possam ser executadas somente após uma parte do software ser implementado, os testes podem ser elaborados e executados durante todas as fases de desenvolvimento. O teste de software envolve quatro etapas: (i) planejamento de testes; (ii) projeto de casos de teste; (iii) execução; e (iv) análise.

Segundo [Ammann e Offutt \(2008\)](#), os testes podem ser derivados de requisitos e especificações, artefatos de *design*, ou do código-fonte e cada nível de teste corresponde a uma atividade de desenvolvimento do software. Os níveis de teste proveem um direcionamento sobre o foco do teste e os tipos de problemas a serem identificados. Decorrente do nível no qual o teste será realizado ele pode ser chamado de:

- **teste de unidade:** visa a identificar defeitos de lógica e de implementação por meio de testes em cada unidade do sistema. Esses testes são realizados separadamente, sem a necessidade de dispor do sistema totalmente finalizado;
- **teste de módulo:** consiste em avaliar o software em relação ao projeto detalhado;
- **teste de integração:** visa a verificar se a interação entre as unidades, individualmente testadas no teste de unidade, funciona de maneira adequada e não revela defeitos;
- **teste de sistema:** é realizado quando as partes do sistema estão integradas. Essa fase visa a verificar se as funcionalidades estão em conformidade com o que foi especificado no documento de requisitos; e
- **teste de aceitação:** é realizado com os usuários finais e consiste em avaliar o sistema em relação aos requisitos, ou seja, visa a verificar se seu comportamento está de acordo com o que foi solicitado.

Na Figura 1 é ilustrado um modelo clássico de como as fases de testes se relacionam em decorrência das atividades envolvidas no ciclo de vida do software. Esse modelo enfatiza a importância do projeto de teste iniciar simultaneamente com o ciclo de desenvolvimento e que o teste não deve ser considerado uma atividade opcional e sim, como uma parte essencial para o desenvolvimento do software.

A fase de análise de requisitos é responsável em compreender as necessidades do cliente, o teste de aceitação, por sua vez, é projetado para determinar se o software atende a essas

Figura 1 – Modelo V descrevendo o paralelismo entre as atividades de desenvolvimento e teste de software

Fonte: Adaptada de [Ammann e Offutt \(2008\)](#).

necessidades do cliente conforme foi especificado. O teste de aceitação deve envolver todos os indivíduos que tenham concretos conhecimentos sobre o domínio do software, tais como, objetivos, requisitos, ambiente, etc.

A fase de projeto arquitetural determina a função de cada componente e conector e de como eles interagem entre si. Nesta fase também deve ser considerado como o sistema funcionará internamente para que os requisitos do cliente possam ser atendidos. O teste do sistema é projetado para avaliar se as unidades individuais e se o sistema funcionam corretamente quando todas as unidades estão trabalhando em conjunto visando geralmente procurar problemas de projeto e especificação.

A fase de projeto de subsistemas no desenvolvimento de software especifica a estrutura e o comportamento de subsistemas, além de como eles estão relacionados para realizar determinadas funções e as funções globais do sistema. Os testes de integração são projetados para verificar se as interfaces entre os módulos de um determinado subsistema possui suposições consistentes e se comunicam corretamente.

A fase de projeto detalhado define a estrutura e o comportamento de módulos individuais. O teste de módulo é projetado para verificar módulos separadamente, mas também, como as unidades e componentes interagem entre si.

Implementação é a fase do desenvolvimento de software que os códigos são produzidos. O teste unitário é definido para avaliar as unidades produzidas pela fase de implementação.

Com base nas definições, a atividade de teste é um processo que executa um determinado programa com objetivo de encontrar defeitos, por isso, é necessária a obtenção de bons casos de teste que possuam maior probabilidade de identificá-los. Nesse contexto, critérios de teste foram propostos com o objetivo de auxiliar de maneira sistemática na seleção dos melhores dados de teste dentre um conjunto de entradas geralmente muito grande ou mesmo infinito. Além disso,

critérios de teste podem ser utilizados para auxiliar na avaliação da adequação desses dados de teste (DELAMARO; MALDONADO; JINO, 2007).

2.3 Técnicas de Teste

De acordo com Pressman (2010), qualquer produto de software pode ser testado principalmente por meio do teste funcional e teste estrutural. No teste funcional, o software é avaliado apenas com base em sua especificação, isto é, o testador não tem acesso às estruturas internas ou implementação do software. Enquanto que o teste estrutural consiste na utilização da estrutura interna do programa para derivar os casos de teste. Além das técnicas de teste funcional e estrutural, existe o teste baseado em defeitos, que consiste na utilização do conhecimento de erros típicos cometidos no processo de desenvolvimento de software (DELAMARO *et al.*, 2007).

Os critérios de teste podem ser classificados em três técnicas. Todavia, nenhuma delas é suficiente para garantir a qualidade da atividade de teste. O ideal é que as técnicas de teste sejam complementares, considerando a forma como são utilizadas, de modo que as vantagens de cada uma sejam melhor exploradas. Nesse sentido, é importante destacar a existência de algumas limitações teóricas referentes à definição de critérios de teste, principalmente daqueles critérios que utilizam a estrutura do programa para derivar os requisitos. Em geral, vale ressaltar as seguintes limitações (DELAMARO; MALDONADO; JINO, 2007):

- é indecidível determinar se uma sequência de comandos (um caminho) é executável, ou seja, não existe um procedimento que indique se existe ou não algum dado de teste que faça com que um caminho do programa seja executado;
- é indecidível se duas rotinas ou dois caminhos computam a mesma função.

Adicionalmente, existem critérios que não são fundamentados nessas três técnicas (funcional, estrutural e baseada em erros), pois possuem características distintas em relação às que foram mencionadas, como é o caso do teste aleatório e do teste exaustivo.

O teste aleatório consiste em gerar o maior número possível de casos de teste de forma aleatória, de tal maneira que eles ajudem na detecção de defeitos ou alcancem uma determinada cobertura, como por exemplo até que todos os comandos do programa em teste sejam executados. Esse teste é considerado um tipo de amostragem aleatória simples, pois o conjunto de dados de teste é gerado arbitrariamente a partir de todo domínio de entrada do programa em teste (ARCURI; IQBAL; BRIAND, 2012).

O teste exaustivo consiste no teste de todos os possíveis cenários exaustivamente (todas permutações e combinações de testes existentes). Entretanto, isso se torna inviável em função da grande quantidade de combinações de casos de testes existentes. De acordo com Pezze e Young (2008), mesmo se fossem ignorados alguns detalhes de implementação, como o tamanho da

memória para armazenar um programa e seus dados, o teste exaustivo não pode ser completado em um tempo hábil.

Apesar do teste aleatório ser utilizado em diversas aplicações de teste, não é considerado uma técnica eficiente quando aplicada em sistemas complexos. O teste exaustivo, independente do tamanho e da complexidade do software, é em geral impraticável. Nesse contexto, nas próximas seções as técnicas de teste funcional, estrutural e baseada em defeitos são detalhadas, concentrando-se principalmente na técnica baseada em defeitos que é o foco principal deste trabalho.

2.3.1 *Teste Funcional*

O teste funcional trata o sistema como uma caixa preta, ou seja, não há acesso aos detalhes de código. Assim, os conjuntos de testes são elaborados apenas com base nos requisitos do software e nas especificações das funcionalidades. Esse teste ignora os mecanismos internos do programa e foca apenas nas saídas geradas em resposta às entradas utilizadas e às condições de execução (IEEE, 1990).

Vale ressaltar também que o teste funcional é utilizado para verificar se as funcionalidades do software são operacionais, se as entradas são adequadamente aceitas, se as saídas são produzidas corretamente, e verificar se a integridade das informações externas (por exemplo, uma base de dados) está sendo mantida.

Considerando o emprego de técnicas de teste funcional, os testadores examinam e, as exigências da especificação do cliente para construir os casos de teste, visando a garantir que o programa seja executado conforme foi especificado. Segundo Pressman (2010), o teste funcional tenta principalmente identificar erros das seguintes categorias: (1) funções incorretas ou ausentes; (2) erros de interface; (3) erros nas estruturas de dados ou acesso a base de dados externa; (4) erros de comportamento ou de desempenho; e (5) inicialização e encerramento de erros.

Existem diversos critérios baseados na técnica de teste funcional dentre eles, pode-se destacar o **Particionamento em classes de equivalência** e **Análise de valor limite**. Ainda que possuam algumas limitações, é importante o uso de critérios funcionais para explorar determinados tipos de defeitos. A utilização do teste funcional provê alguns pontos fortes, dentre os quais destacam-se:

- **reutilização de casos de teste:** por ser um critério baseado nas especificações, e portanto independente da implementação, mesmo que haja alteração no código os casos de teste podem permanecer os mesmos;
- **independência de paradigma:** uma vez que o código fonte não é necessário, esses critérios podem ser aplicados em qualquer programa, seja ele procedimental ou orientado a objetos;

- **qualidade de conjuntos de teste:** os critérios funcionais resultam um conjunto de teste com boa representação de todo domínio de entrada das variáveis em teste; e
- **tempo reduzido:** a derivação dos casos de teste pode ser realizada paralelamente à implementação, reduzindo o tempo do projeto.

2.3.2 Teste Estrutural

O teste estrutural, também conhecido como caixa branca, consiste na utilização da estrutura interna do programa para derivar os casos de teste. Muitos desses critérios utilizam uma representação de programa chamada Grafo de Fluxo de Controle (GFC). O GFC consiste em uma notação para representar o fluxo de execução de um determinado programa.

A representação de um dado programa P como um GFC consiste em estabelecer uma correspondência entre vértices (nós) e blocos de comandos e em indicar possíveis fluxos de controle entre esses blocos por meio das arestas direcionadas (arcos). Assim, assume-se que um GFC é um grafo orientado, com um único nó de entrada, em que cada nó representa um bloco indivisível de comandos e cada aresta representa um possível desvio de um bloco para outro (MYERS; SANDLER, 2004).

Para ilustrar o GFC é utilizado o programa *identifier* (Figura 2). O programa *identifier* determina se um dado identificador é válido ou não. Um identificador válido inicia com uma letra e contém apenas letras ou dígitos. Além disso possui no mínimo um e no máximo seis caracteres de tamanho.

Figura 2 – Exemplo de GFC para o programa *identifier*

Fonte: Adaptada de Barbosa *et al.* (2007).

Formalmente, um bloco básico é uma sequência máxima de comandos do programa de tal forma que, se qualquer instrução do bloco for executada, todas as declarações no bloco são executadas. Um bloco básico tem apenas um ponto de entrada e um ponto de saída (AMMANN; OFFUTT, 2008). Os critérios baseados na técnica de teste estrutural são classificados com base no fluxo de dados, no fluxo de controles e na complexidade.

Os critérios de fluxos de dados são baseados nas informações de fluxo de dados (definições e usos das variáveis). Esses critérios combinam definições de variáveis com seus usos, garantindo que cada valor computado seja de fato usado. Assim, selecionando, dentre muitos caminhos de execução, um conjunto com mais chance de propagar um resultado de uma computação incorreta até o ponto de um defeito observável (PEZZE; YOUNG, 2008). Dessa categoria, pode-se destacar os critérios: todas-definições, todos-usos e todos-*du*-caminhos.

Os critérios baseados em fluxo de controle utilizam somente informações referentes ao fluxo de controle da execução do programa, como blocos de comandos e desvios, tal como o critério de cobertura de ramos (do inglês, *Branch testing* ou *Branch Coverage*).

A cobertura de ramos é um critério de teste estrutural que visa garantir que cada um dos ramos possíveis de cada ponto de decisão seja executado pelo menos uma vez e, assim, garantir que todos os comandos alcançáveis sejam executados. Dentro dos GFCs os nós correspondentes às instruções de decisão (por exemplo, uma instrução *if* ou *while*) são referidos como nós de ramificação ou *branching nodes*.

O ramo executado quando a condição no nó de ramificação é verdadeira é definido como ramo verdadeiro (*true branch*). Ao contrário, o ramo executado quando a condição é falsa é definido como ramo falso (*false branch*). E, por fim, a condição que determina se um ramo é tomado é referida como um predicado de ramo (*branch predicate*).

Assim, o objetivo é encontrar um conjunto de dados de teste que exercite todas as saídas verdadeiras e falsas de todas as decisões. Um ramo é um caminho potencial para a execução de um nó de ramificação, porém, existem ramos que são impossíveis de se percorrer, pois não existe nenhuma entrada que seja capaz de avaliar o predicado que este ramo é executado.

A aplicação dos critérios de teste estrutural são independentes da especificação e podem ser iniciados mesmo durante a implementação do software. O teste estrutural é uma técnica para teste a nível de código que identifica defeitos nos caminhos lógicos de um programa, incluindo ramos, condições e instruções. Apesar da tarefa de exercer todos os caminhos válidos do programa ser inviável, os critérios baseados nessa técnica estabelecem de forma sistemática os requisitos a serem testados para alcançar uma boa cobertura.

2.3.3 Teste de Mutação

Teste de mutação é um critério de teste baseado em defeitos, que tem como principal finalidade medir a adequação de um conjunto de teste. Para isso, o teste de mutação utiliza um

conjunto de programas com erros introduzidos em sua estrutura, representados por pequenas alterações sintáticas no programa em teste (DEMILLO; LIPTON; SAYWARD, 1978; DEMILLO; SAYWARD; LIPTON, 1979).

De acordo com Jia e Harman (2011), os primeiros relatos sobre o teste de mutação foram feitos em 1971, em um periódico estudantil feito por Lipton (1971) e no final dos anos 70 em trabalhos de Hamlet (1977) e Demillo, Lipton e Sayward (1978). Desde os primeiros relatos, o teste de mutação tem sido alvo de muitas pesquisas em função da sua eficiência em identificar defeitos e problemas relacionados com o alto custo de sua aplicação.

Segundo Offutt (1992), a definição do critério de mutação é baseada em duas hipóteses: (i) hipótese do programador competente (do inglês, *competent programmer hypothesis*), assume que programas desenvolvidos por programadores competentes estão corretos ou próximo do correto; e (ii) efeito de acoplamento (do inglês, *coupling effect*), assume que um conjunto de dados de teste capaz de detectar erros simples em um programa, também será capaz de detectar erros mais complexos.

Na etapa inicial do teste de mutação, o testador deve executar um conjunto de casos de testes $T = \{t_1, t_2, \dots, t_n\}$ em um programa P para verificar se o resultado obtido é igual ao resultado esperado. Caso eles sejam diferentes, então o programa possui um defeito e deve ser corrigido para dar continuidade ao teste.

A partir de um programa P , é gerado um conjunto de programas defeituosos denominados de mutantes $\{M_1, M_2, M_3, \dots, M_k\}$. Esses mutantes são gerados com auxílio de operadores de mutação que são definidos de acordo com as características da linguagem de programação em que os programas estão sendo implementados. Um operador de mutação é uma regra que determina alterações sintáticas a serem aplicadas em uma instrução (I) em P , criando uma instrução mutada I' .

Em seguida, o mesmo conjunto de teste T aplicado em P é executado no conjunto de mutantes, podendo apresentar dois tipos de resultados: (i) Se a saída de um M_i for diferente de P para qualquer dado de teste; ou (ii) Se a saída de um M_i for igual a de P (DEMILLO; LIPTON; SAYWARD, 1978).

Para o resultado (i), o mutante é dito “morto” pois o erro produzido em M_i a partir de P foi detectado pelo dado de teste. No resultado (ii), o mutante M_i é dito “vivo” pois nenhum caso de teste foi capaz de detectar o erro que o originou. É possível que M_i e P sejam programas equivalentes, ou seja, M_i e P apresentam sempre o mesmo resultado para todos os elementos do domínio de entrada e portanto M_i não pode ser morto.

Na Figura 3 é exibido um exemplo de uma mutação no operador relacional do programa original P . A mutação ocorre na alteração do operador “<” para “>”.

Os operadores de mutação são criados com a finalidade de induzir mudanças sintáticas simples com base nos erros comuns cometidos durante o processo de desenvolvimento, por

Figura 3 – Exemplo de mutante

Fonte: Elaborada pelo autor.

exemplo, alterar o nome de uma variável, trocar um operador por outro, etc. Em geral não existe uma maneira de definir os operadores de mutação para uma dada linguagem, os operadores são determinados com base na experiência do uso da linguagem, assim como nos erros mais comuns que podem ser cometidos pelos programadores durante a sua utilização (OFFUTT *et al.*, 1996).

A qualidade e quantidade dos operadores de mutação são fatores que possuem forte influência no sucesso do teste de mutação. Normalmente, a quantidade de mutantes gerados é alta e nem sempre com boa qualidade. Um mutante de boa qualidade reflete em um erro difícil de ser identificado, ao passo que um mutante de má qualidade é aquele que possui erros simples que podem ser facilmente identificados por qualquer caso de teste.

Para realizar a aplicação do teste de mutação o uso de ferramentas é indispensável, visto que existem diversas etapas como *i)* geração de mutantes; *ii)* geração de dados de teste; *iii)* identificação e marcação de mutantes equivalentes; e *iv)* oráculos e análise de escores, que requerem esforços e podem ser propensas a erros. Realizar essas etapas manualmente para um programa complexo pode ser inviável, em função do tempo e do tamanho do programa a ser testado. Portanto, a disponibilidade de ferramentas proporciona maior qualidade ao teste de mutação e contribui para seu uso em ambientes industriais.

Em suma, os mutantes são confrontados com o mesmo conjunto de teste T que foi executado em P . A finalidade é obter dados de teste que sejam capazes de matar todos os mutantes (não-equivalentes) e detectar os mutantes equivalentes. Desse modo, é possível obter um conjunto T que seja adequado para verificar se P está livre de defeitos ou possui defeitos pouco prováveis de ocorrerem (DELAMARO *et al.*, 2007).

Mutantes equivalentes sempre irão produzir a mesma saída do programa original independente do teste aplicado. O problema de mutantes equivalentes apresenta desvantagens referentes ao uso do teste de mutação, pois a árdua tarefa de identificação e eliminação dos equivalentes é realizada manualmente pelo testador, o que a torna muito custosa e demorada.

Para um dado programa P , diz-se que M_i é um mutante equivalente, se ele é sintaticamente diferente de P , porém com o mesmo comportamento. Na Figura 4 é ilustrado um exemplo de mutante equivalente gerado com a mudança do operador “==” do programa P para o operador “>=”. Se a instrução não alterar o valor de i , o programa P e o mutante M_i geram a mesma saída.

Figura 4 – Exemplo de Mutante Equivalente

Programa Original	Mutante Equivalente
<pre>int i = 0; while (...) { i ++; if (i == 10) { break; } }</pre>	<pre>int i = 0; while (...) { i ++; if (i >= 10) { break; } }</pre>

Fonte: Elaborada pelo autor.

O teste de mutação fornece uma medida objetiva do grau de adequação dos conjuntos de teste analisados, denominada escore de mutação (do inglês, *Mutation Score (MS)*), que varia entre 0% e 100%. O escore de mutação pode ser utilizado para medir a eficácia de um conjunto de teste em termos da sua capacidade para detectar falhas. Calcula-se o escore de mutação da seguinte maneira (Função 2.1) (DELAMARO *et al.*, 2007):

$$ms(P,T) = \frac{DM(P,T)}{M(P) - EM(P)} \quad (2.1)$$

em que,

DM (P,T): número de mutantes mortos pelo conjunto de teste T;

M(P): número total de mutantes gerados;

EM(P): número de mutantes gerados equivalentes a P.

Quando o valor do escore de mutação for igual a 100%, significa que o conjunto de teste *T* é adequado para testar o programa *P*. Assim, a confiança de que o programa em teste está praticamente livre de defeitos é consideravelmente alta.

Na Figura 5 é apresentado o processo tradicional do teste de mutação. Nesse caso, a análise dos mutantes equivalentes é realizada manualmente e o processo ocorre até que todos os mutantes estejam mortos, ou seja, até que o escore de mutação seja 100%. Porém, eliminar todos os mutantes equivalentes manualmente e alcançar um alto escore de mutação pode ser muito caro para o processo de teste.

Normalmente, para distinguir o comportamento entre *P* e *M*, os dados de um teste devem satisfazer três premissas conhecidas, como condições Alcançabilidade, Infecção e Propagação (do inglês, *Reachability, Infection e Propagation (RIP)*), que se referem às condições de Alcançabilidade, Necessidade e Suficiência proposto por DeMillo e Offutt (1991). A primeira condição, alcançabilidade, afirma que um teste deve atingir a instrução mutada para um mutante ser morto. A segunda condição, necessidade, requer que o estado de execução do programa mutante seja diferente do programa original após a execução da instrução mutada. A terceira condição, a

Figura 5 – Processo genérico do teste de mutação

Fonte: Adaptada de [Offutt e Untch \(2000\)](#).

suficiência, requer que o estado incorreto seja propagado para uma saída do programa. A partir disso, as condições RIP são descritas como:

- **Alcançabilidade:** $t \in T$ deve ser capaz de alcançar o ponto de mutação. Se I e I' não forem executados por t , então é garantido que t não mata M .
- **Infecção:** $t \in T$ deve ser capaz de causar diferentes estados internos em P e M imediatamente após a execução de I e I' . Assim, para t para matar M , é necessário que I e I' sejam atingidos e o estado após a sua execução deve ser diferente.
- **Propagação:** $t \in T$ deve ser capaz de fazer com que o estado final do M seja diferente de P , isto é, o estado infectado deve se propagar até um ponto em P no qual possa ser observado. Assim, o estado diferente causado pela satisfação da condição de necessidade deve propagar-se através da execução do programa para produzir uma saída diferente.

A geração de dados de teste é uma atividade importante para alcançar uma boa eficácia e eficiência na fase de teste de software. No entanto, encontrar dados de teste adequados de acordo com um critério específico pode ser muito trabalhoso. Aplicada ao teste de mutação, as técnicas de geração de dados de teste exigem esforço substancial para gerar um conjunto de dados de teste adequado e um esforço adicional para executar o conjunto de dados de teste contra o conjunto de

mutantes. Motivados por essa última questão pesquisadores propuseram variações do teste de mutação chamadas de *Weak* (HOWDEN, 1982) e *Firm mutation* (WOODWARD; K., 1988).

De acordo com teste de mutação fraca, um T que satisfaz as condições de alcançabilidade e necessidade também satisfaz o critério de teste de mutação fraca. Mutação fraca compara o estado da instrução mutada I' com o original I imediatamente após a sua execução para decidir se o mutante foi fracamente morto. Assim, o mutante fracamente morto é um termo empregado quando I e I' produzem estados diferentes e, caso contrário, estão fracamente vivos. É importante ressaltar que um mutante pode ser considerado fracamente morto e, ainda P e M podem produzir as mesmas saídas (HOWDEN, 1982).

O teste de mutação firme é uma abordagem intermediária entre a mutação forte (teste de mutação tradicional) e a mutação fraca, uma vez que o ponto de comparação localizado em qualquer lugar após o I' e o fim de P . Assim, em mutação firme pode ser considerada uma parte do código de P . Em vez de executar todo o programa como teste de mutação forte ou considerando apenas o estado após I' como mutação fraca. Para que um mutante seja considerado morto, o usuário deve definir condições ao expressar o que é necessário para que os dados de um teste sejam adequados (WOODWARD; K., 1988).

Estudos têm mostrado que o teste de mutação é um critério bastante efetivo em revelar defeitos, levando à construção de conjuntos de teste com alta qualidade. No entanto, existem pontos que precisam ser tratados para que o critério possa ser usado de forma prática, principalmente se for considerado o custo de sua aplicação.

O primeiro problema na utilização do critério de teste de mutação está relacionado à execução dos mutantes pois, normalmente, a quantidade de mutantes gerada em um programa simples pode ser alta, conseqüentemente exigindo um tempo de execução elevado. Na tentativa de reduzir o custo inerente a esse processo, pesquisadores propuseram técnicas como: mutação seletiva (do inglês, *selective mutation*) (OFFUTT; ROTHERMEL; ZAPF, 1993; OFFUTT *et al.*, 1996); amostragem mutante (do inglês, *mutant sampling*) (MATHUR; WONG, 1994; MATHUR; WONG, 1995); operadores essenciais (BARBOSA, 1998); e agrupamento mutante (do inglês, *mutant clustering*) (JI *et al.*, 2009b).

De acordo com Jia e Harman (2011) as técnicas para redução de mutantes consistem em selecionar um subconjunto de mutantes M' de um conjunto de mutantes M , de modo que o escore de mutação do conjunto M' seja semelhante ao do conjunto M ou em definir um subconjunto de operadores de mutação para a redução de mutantes gerados. Entre as técnicas existentes, pode-se destacar:

- **mutação seletiva:** essa técnica consiste em gerar os mutantes por meio de um subconjunto de operadores de mutação, de modo que não se tenha perda significativa na eficácia do teste de mutação (OFFUTT *et al.*, 1996).

- **amostragem mutante:** consiste na seleção de um subconjunto de mutantes aleatoriamente de um conjunto de mutantes gerados por meio de todos operadores tradicionais (MATHUR; WONG, 1995).
- **operadores essenciais:** os mutantes são gerados por meio de um subconjunto do total de operadores de mutação para a linguagem de programação C (BARBOSA, 1998).
- **agrupamento mutante:** nessa técnica é selecionado um subconjunto de mutantes utilizando algoritmos de *clustering* (JI *et al.*, 2009b).

Nesta perspectiva, a quantidade de mutantes está associada ao número de operadores de mutação, uma vez que muitos mutantes são gerados. Este problema pode ser solucionado determinando um subconjunto de operadores de mutação que possui a mesma eficácia que um conjunto composto por muitos operadores de mutação.

Somando-se a isso, o fato da existência de mutantes equivalentes pode requerer a intervenção do testador para que sejam identificados. De um modo geral, identificar mutantes equivalentes é um problema indecível, como foi comprovado no estudo de Budd e Angluin (1982). Dessa forma, se todos os mutantes equivalentes não forem identificados, é impossível o escore de mutação alcançar 100% o que reduz a confiança de que o conjunto de teste seja eficaz.

Em relação à linha de esforços para identificação de mutantes equivalentes, foram investigados por meio de uma revisão sistemática conduzida por Orzeszyna (2011) alguns métodos para tentar solucionar esse problema. O autor identificou 22 estudos que foram classificados considerando os seguintes aspectos:

- **técnicas para detecção de mutantes equivalentes:** essa categoria considera os estudos que visam tratar o problema de mutantes equivalentes por meio de técnicas ou estratégias para detectá-los no conjunto de todos os mutantes. Nessa categoria foram identificados 8 estudos (BALDWIN; SAYWARD, 1979; OFFUTT; CRAFT, 1994; OFFUTT; PAN, 1996; OFFUTT; J., 1997; HIERONS; HARMAN; DANICIC, 1999; ELLIMS; INCE; PETRE, 2007; MARTIN; XIE, 2007; BOUSQUET; DELAUNAY, 2008).
- **técnicas para evitar a geração de mutantes equivalentes:** essa categoria refere-se ao uso de técnicas que evitam a geração de mutantes equivalentes, por meio de estratégias focadas nos tipos de mutantes e como eles são gerados, como o teste de mutação de ordem superior¹. Foram identificados 10 estudos para essa categoria (ELFURJANI; BOTTACI, 1999; HARMAN; BRYAN, 2001; ADAMOPOULOS; HARMAN; HIERONS, 2004; OFFUTT; MA; KWON, 2006; JIA; HARMAN, 2009; KAMINSKI; MASON; AMMANN, 2009; JI *et al.*, 2009a; OFFUTT, 1992; KINTIS; PAPADAKIS; MALEVRIS, 2010; PAPADAKIS; MALEVRIS, 2010b).

¹ No teste de mutação de ordem superior os mutantes são criados pela inserção de dois ou mais erros (JIA; HARMAN, 2009).

- **técnicas para sugestão de mutantes equivalentes:** essa categoria está relacionada às técnicas que realizam uma análise no comportamento dos mutantes para indicar ao testador a existência de um possível mutante equivalente. Em geral, essas técnicas avaliam o impacto do comportamento interno dos mutantes para predizer se eles são equivalentes ou não-equivalentes. Foram incluídos 4 estudos nessa categoria ([VINCENZI et al., 2002](#); [GRUN; SCHULER; ZELLER, 2009](#); [SCHULER; DALLMEIER; ZELLER, 2009](#); [SCHULER; ZELLER, 2010](#)).

De acordo com ([ORZESZYNA, 2011](#)), mesmo após anos de pesquisas o método mais eficaz identificado ([OFFUTT; PAN, 1996](#); [OFFUTT; J., 1997](#)) detecta apenas 47,63% de mutantes equivalentes (primeira categoria). Os métodos presentes na categoria para evitar a geração de equivalentes, alcançaram até 86.8% de redução com apenas 1,75% de perda de eficácia no teste de mutação (segunda categoria) e métodos para sugestão de mutantes equivalentes baseados em medidas de impacto, podem alcançar até 75% de precisão (terceira categoria).

Nesse contexto, esse é um problema de grande interesse teórico, pois solucioná-lo, é um dos maiores desafios do teste de mutação para que esse critério seja utilizado de fato em ambientes reais. Além disso, encontrar dados de teste relevantes para um determinado critério pode ser uma tarefa muito trabalhosa, pois impõe um obstáculo para a automatização do critério ([OFFUTT; UNTCH, 2000](#)). Assim, para o uso eficaz do teste de mutação é necessário que sejam gerados dados de testes capazes de eliminar o maior número de mutantes possíveis, somado com um método que seja capaz de identificar os mutantes equivalentes automaticamente.

2.4 Geração de Dados de Teste

A atividade de geração de dados automática é fundamental para o teste de software, pois os dados de teste são um dos fatores-chave para determinar a qualidade do teste durante sua execução. Assim, a automatização é uma tarefa desejável, porém não trivial, em função da complexidade de determinados domínios, justificando o interesse crescente de diversos pesquisadores no uso de métodos de otimização para a geração automática dos dados de teste ([KOREL, 1990](#)).

Essa complexidade está relacionada com diversas limitações inerentes à atividade de teste que impossibilitam automatizar completamente essa etapa, tais como ([VERGILIO; MALDONADO; JINO, 2007](#)):

- **correção coincidente:** ocorre quando um programa em teste possui um defeito que é identificado por um dado de teste, um estado de erro é produzido, mas coincidentemente um resultado correto é obtido. Essa limitação pode impedir que os dados gerados identifiquem todos os defeitos de um programas pois um defeito pode mascarar o outro e a saída obtida

pode corresponder à esperada;

- **caminho ausente:** corresponde a uma determinada funcionalidade requerida para o programa, mas que por engano não foi implementada, isto é, o caminho correspondente não existe no programa. Neste caso, a falta da implementação de uma funcionalidade no programa faz com que o requisito correspondente nunca seja alcançado por um dado de teste. Portanto, é impossível para uma estratégia gerar dados de teste que percorram um caminho ausente;
- **caminhos não executáveis:** um caminho é não executável caso não exista um conjunto de valores para as variáveis de entrada do programa, parâmetros e variáveis globais que causem a sua execução. Uma das causas dos caminhos não executáveis são códigos mortos, ou seja, aquelas instruções que nunca podem ser executadas. Isso causa caminhos não executáveis e portanto, não existe nenhum dado de teste que consiga alcançar essa instrução; e
- **mutantes equivalentes:** como já foi mencionado anteriormente, não existe nenhum dado de teste capaz de identificar o comportamento distinto entre o programa original e o mutante. Assim, os mutantes equivalentes demonstram uma falsa fraqueza nos conjuntos de teste e na estratégia para geração automática de dados de teste.

Um caso de teste é composto por um dado de teste que consiste em uma entrada necessária para executar o código de teste e sua saída esperada. Assim, a geração de dados de teste é uma atividade focada na busca de valores de entrada que atendam a critérios de teste específicos. Estes valores de entrada podem assumir diferentes formas, tais como parâmetros de uma função, parâmetros para uma sequência de chamadas de método, dentre outras.

Formalmente, $I = \{x_1, x_2, \dots, x_n\}$ sendo um vetor de entradas para as variáveis de P , tal que $x_i \in Dx_i$, onde Dx_i é o conjunto com todos possíveis valores que podem ser atribuídos para x_i . E para conjuntos de testes pode se considerar $S \rightarrow R$, onde S é o conjunto com todas possíveis entradas e R o conjunto de todas saídas. Assim, uma única entrada x_i em S é referida como um dado de teste (KOREL, 1990).

No teste de mutação, a geração de dados de teste consiste em identificar entradas de teste que maximizam o número de mutantes mortos. A questão fundamental é identificar um pequeno conjunto de dados de teste, de modo que, quando executado sobre o conjunto de mutantes seja obtido o maior número de mutantes mortos. Matar todos os mutantes não-equivalentes significa que um conjunto de teste de alta qualidade foi obtido e mutantes vivos indicam a ineficiência de um conjunto de teste. Neste último caso, o processo consiste em encontrar novos dados de teste

e melhorar o conjunto de teste até que todos os mutantes não equivalentes tenham sido mortos (OFFUTT, 1990).

Encontrar um conjunto de dados de teste adequado ainda é uma atividade muito difícil, em primeiro lugar devido às estruturas de dados dinâmicas e ao tamanho do domínio de entrada de um programa (muitas vezes, é enorme ou mesmo infinito) e depois, o esforço necessário para uma exigência de teste. Deste modo, a automatização da geração de dados de teste é fundamental para tornar os critérios de teste mais eficientes e mais baratos.

Apesar das limitações existentes, no contexto geral, algumas categorias de técnicas podem ser utilizadas para gerar dados de teste, a fim de satisfazer diversos critérios, bem como a proposição de novos métodos/técnicas para automatizar a geração de dados de teste.

Dentre as técnicas existentes, destacam-se: *geração aleatória* (DURAN; NTAPOS, 1984; HAMLET; TAYLOR, 1990); *geração com execução simbólica* (BOYER; ELSPAS; LEVITT, 1975; CLARKE, 1976; HOWDEN, 1977; RAMAMOORTHY; HO; CHEN, 1976); *geração com execução dinâmica* (KOREL, 1990); *geração utilizando técnicas de busca e meta-heurísticas* (AYARI; BOUKTIF; ANTONIOL, 2007; MICHAEL; MCGRAW; SCHATZ, 2001; WEGENER; BARESEL; STHAMER, 2001) e *geração de dados sensíveis a defeitos* (WHITE; COHEN, 1980). Nessa última categoria dentre as técnicas existentes destaca-se o teste baseado em restrição (DEMILLO; OFFUTT, 1991), cujo objetivo é auxiliar a geração de dados de teste adequados ao teste de mutação.

As seções a seguir visam apresentar algumas técnicas para geração de dados de teste citadas anteriormente.

2.4.1 Geração Aleatória

Essa técnica consiste em gerar dados de teste aleatoriamente até que uma entrada útil seja encontrada ou um critério seja satisfeito. A geração aleatória pode ser facilmente implementada e é bastante utilizada para fins de comparação com outras técnicas e até mesmo para teste em programas mais simples.

No entanto, essa abordagem não é adequada para o uso em programas ou critérios de adequação complexos, pois uma entrada teria que satisfazer requisitos muito específicos. Em tal caso, o número de entradas adequadas pode ser muito pequeno em comparação ao número total de entradas, de modo que a probabilidade que uma entrada adequada seja selecionada pode ser muito baixa (MICHAEL; MCGRAW; SCHATZ, 2001).

2.4.2 Geração com Execução Simbólica

Geração de dados baseada em execução simbólica consiste em atribuir valores simbólicos às variáveis, a fim de obter uma caracterização matemática sobre a função que o programa

desempenha. Dessa forma, a atividade de geração de dados de teste é convertida em um problema de resolução de expressão algébrica.

Segundo Vergilio, Maldonado e Jino (2007), normalmente a geração com execução simbólica é utilizada em conjunto com a técnica estrutural baseada em caminhos, com a finalidade de auxiliar a geração automática de dados de teste para um determinado caminho ou conjunto de caminhos. A aplicação dessa técnica é bastante custosa e, adicionalmente, existem alguns problemas relacionados às estruturas de laços indefinidas, referências indiretas para uma variável (como acesso em vetor), referências a ponteiros e chamadas de métodos.

2.4.3 Geração com Execução Dinâmica

Geração com Execução Dinâmica (do inglês, *Dynamic Symbolic Execution (DSE)*) é uma abordagem alternativa à geração de dados de teste simbólica e foi proposta por Korel (1990). Essa técnica é uma abordagem fundamentada na execução real de um programa em teste, na análise de fluxo de dados dinâmica e nos métodos de minimização de funções. A análise de fluxo de dados dinâmica é responsável em definir quais variáveis de entrada estão relacionadas com o comportamento incorreto do programa.

A técnica seleciona o conjunto de valores de entrada para executar um caminho específico no programa. De acordo com Korel (1990), esse objetivo pode ser representado com uma sequência de subobjetivos, em que eles podem ser solucionados por meio de métodos de minimização de funções.

2.4.4 Teste baseado em restrição

O teste baseado em restrição (do inglês, *Constraint-Based Testing (CBT)*) proposto por DeMillo e Offutt (1991), visa gerar dados de teste para o teste de mutação utilizando restrições algébricas. O CBT se baseia na ideia de que se um caso de teste é capaz de matar um mutante, então o sistema de restrição é verdadeiro. Caso contrário, significa que não existem casos de teste capazes de matar o mutante. Para que um caso de teste seja capaz de matar um mutante três condições devem ser satisfeitas: 1) Alcançabilidade, 2) Necessidade e, 3) Suficiência.

Dentre as condições mencionadas, a condição de suficiência pode ser considerada a ideia principal do CBT, pois, satisfazê-la indica que os casos de teste revelaram um comportamento diferente entre os programas originais e mutantes.

2.5 Teste de Software Baseado em Busca

Técnicas baseadas em busca foram utilizadas pela primeira vez por Miller e Spooner (1976) no domínio de geração de dados de teste. No entanto, o termo SBST surgiu somente anos depois, por McMinn (2004), em função da intensificação das pesquisas envolvendo o uso de

técnicas de busca e meta-heurísticas para geração de dados e outras atividades no âmbito de teste de software.

O SBST é o processo de automatização de uma atividade de teste empregando algumas técnicas, como buscas locais e globais, meta-heurísticas e algoritmos evolutivos. Tais técnicas oferecem soluções ótimas ou aproximadas para problemas complexos a um custo computacional razoável quando comparadas às técnicas convencionais, como técnicas aleatórias.

Para a geração de dados de teste no contexto geral, diversos estudos têm empregado técnicas de busca e meta-heurísticas. A maioria desses estudos utilizam Algoritmo Genético (AG) (do inglês, *Genetic Algorithm (GA)*) com função de avaliação (ou função objetivo) baseada na cobertura do dado de teste ou de um elemento específico. A tendência é que o uso de métodos baseados em busca aumente progressivamente, principalmente, em função do amplo uso com sucesso em várias áreas da Ciência da Computação. Contudo, a maioria dessas técnicas é sensível à modelagem do problema e, caso elas sejam utilizadas erroneamente, além de influenciar nos resultados obtidos, pode-se não obter soluções satisfatórias ou próximas do ótimo desejado, e também comprometer o custo computacional da abordagem.

De acordo com [Rayward-Smith et al. \(1996\)](#) e [Zbigniew e Fogel \(2004\)](#), “heurística”, no contexto de um algoritmo de otimização, é uma função que utiliza as informações coletadas durante a execução do algoritmo para auxiliar na decisão de quais soluções candidatas devem ser testadas ou produzidas para os passos seguintes. Já uma meta-heurística é um método ou algoritmo que combina funções-objetivo com heurísticas. Segundo [Weise \(2009\)](#), essa combinação é muitas vezes realizada por meio de estatísticas obtidas das amostras de um espaço de busca ou baseadas em algum fenômeno natural ou processo físico. Um exemplo é a analogia da técnica AG com o comportamento natural e evolutivo das espécies descrito na teoria de Charles Darwin², ou a metáfora de um processo térmico utilizado em metalurgias com a técnica Têmpera Simulada que será detalhada nas próximas seções.

Nesse contexto, as meta-heurísticas são algoritmos que visam a resolução de problemas de otimização objetivando encontrar soluções ótimas ou aproximadas ao ótimo. Normalmente, tais técnicas são utilizadas em problemas que não podem ser resolvidos por algoritmos convencionais ou que possuam grande complexidade. Diversos desses problemas são problemas de busca e otimização, nos quais não é possível encontrar uma solução ótima que seja alcançada em tempo razoável a um custo computacional aceitável.

As técnicas ou algoritmos de otimização são divididos basicamente em duas classes: determinísticos e probabilísticos (Figura 6). Os algoritmos determinísticos irão produzir o mesmo resultado, sempre que uma mesma entrada do problema for usada pelo programa. Já os algoritmos probabilísticos (também chamados de não-determinísticos), a partir de uma mesma

² Charles Darwin foi um famoso naturalista britânico que propôs a teoria da evolução das espécies. Esta teoria culminou no que é, agora, considerado o paradigma central para explicação de diversos fenômenos na biologia ([WYHE, 2002](#)).

entrada do problema, podem gerar soluções distintas, pois em uma execução consideram eventos ou operações aleatórias (WEISE, 2009).

Da classe dos algoritmos baseados em busca, é importante ressaltar a computação evolucionária, que abrange todos os algoritmos baseados em um conjunto de soluções candidatas que são iterativamente melhoradas e evoluídas usando informações do problema. Dentre esses algoritmos, o AG é um dos mais utilizados para problemas de otimização.

Figura 6 – Algoritmos baseados em busca

Fonte: Adaptada de Weise (2009).

O processo de otimização é um mecanismo pelo qual se objetiva encontrar o valor máximo ou mínimo de uma função. Esse mecanismo é usado em áreas como matemática, física, química, economia e engenharia, nas quais o objetivo é maximizar a eficiência ou encontrar os possíveis melhores elementos de um conjunto de dados seguindo alguns critérios estabelecidos. Esses critérios são funções matemáticas associadas ao problema e são denominados de “função objetivo”. A otimização pode se referir tanto à minimização ou à maximização. A maximização de uma função f é equivalente à minimização do oposto dessa mesma função, $-f$ (BLONDIN, 2009).

Matematicamente, a tarefa de minimização pode ser determinada como:

Dado $f: \mathcal{R}^n \rightarrow \mathcal{R}$

Buscar $x' \in \mathcal{R}^n$ tal que $f(x') \leq f(x), \forall x \in \mathcal{R}^n$

Obviamente a maximização será o inverso da notação acima:

Buscar $x' \in \mathcal{R}^n$ tal que $f(x') \geq f(x), \forall x \in \mathcal{R}^n$

f é a função objetivo, e $\mathcal{R}^n$ é referido como o espaço de busca. O valor de n significa o número de dimensões do espaço de busca, ou seja, quantos parâmetros desejam-se otimizar. Cada elemento de $\mathcal{R}^n$ é denominado de solução candidata, sendo x' , a solução ótima.

Para a melhor compreensão das técnicas que são detalhadas a seguir, alguns termos são apresentados:

- **espaço de busca:** conjunto de possibilidades ou soluções candidatas para solucionar um determinado problema;
- **máximos e mínimos locais:** são boas soluções candidatas dentro do espaço de busca, porém não são as melhores soluções para o problema;
- **função objetivo:** trata-se de uma função especial que guia a busca para o objetivo e avalia a qualidade das soluções candidatas. Ela é determinada de acordo com o problema. A função objetivo (também chamada de função de *fitness*) é uma peça fundamental para as técnicas de busca, visto que a convergência e o sucesso da técnica é altamente dependente dessa função;
- **geração/iteração:** o momento em que a técnica completa a execução de um ciclo, é denominado de geração ou iteração; e
- **critério de parada:** consiste em uma estratégia para o término da execução de uma técnica, sendo, em geral, um determinado número de gerações/iterações, critério de convergência ou um determinado tempo.

Normalmente, os problemas complexos tendem a ser incompreensíveis devido à natureza dos dados, ao grande tamanho do espaço de busca e consequentemente ao custo para encontrar uma solução. Em tais problemas, o objetivo é encontrar a melhor solução dentre todas as possíveis, isto é, minimizar ou maximizar a função objetivo para alcançar o “ótimo global”.

Um ótimo global é um ponto ótimo em relação a todo o domínio de soluções D_s , enquanto um ótimo local é um ótimo em apenas um subconjunto de D_s . Normalmente, podem existir mais de uma solução ótima global em um problema de geração de dados de teste baseado em busca. Por exemplo, dado problema onde o objetivo é cobrir o ramo verdadeiro do predicado $var > 5$. Qualquer entrada para var que seja maior que 5 irá cobrir o ramo como verdadeiro e, portanto, pode ser qualificado como uma solução globalmente ideal.

Atualmente diferentes técnicas baseadas em busca e meta-heurísticas têm sido aplicadas em problemas de teste, como testes baseados em especificações, testes estruturais e de mutação. A Figura 7 mostra o processo genérico de geração de dados de teste baseada em busca. O funcionamento do processo se inicia a partir da aplicação de uma técnica de busca no domínio de entrada do programa em teste. A técnica de busca é responsável em produzir os dados de teste, uma vez que o conjunto inicial de teste é gerado, ele deve ser executado no programa visando verificar a cobertura obtida.

A função de *fitness* deve ser definida de acordo com as características do problema, ou seja, o objetivo é representado por uma característica particular capaz de avaliar as soluções candidatas, em termos da sua eficácia e adequação para resolver o problema em questão. Deve ser observado que, para aplicar efetivamente uma abordagem baseada em busca, é necessário ter uma boa função de *fitness*, mas isso nem sempre é uma tarefa trivial (MCMINN, 2011). No contexto de teste de software, as funções de *fitness* são encarregadas de avaliar os conjuntos de testes. Em geral, informações produzidas com a execução do programa, tais como, cobertura, caminhos executados, entre outras podem ser úteis para avaliar a adequação de um conjunto de dados de teste.

Para o problema de geração de dados de teste, normalmente, o espaço de busca costuma ser muito grande, mesmo para um programa pequeno. O espaço de busca consiste em todos os valores de entrada possíveis para executar o programa em teste. Considerando um programa com duas variáveis de entrada do tipo inteiro (a e b), o espaço de busca representa cada combinação possível de inteiros para a e b no domínio de entrada. Note que para um programa podem existir diferentes soluções ótimas, isto é, existem diferentes entradas que satisfazem o mesmo critério de teste específico. Portanto, encontrar um conjunto de dados de teste em um grande espaço de busca nem sempre é garantido sem uma boa função de *fitness*.

Dependendo do programa e do objetivo do teste, diferentes estratégias para as funções de *fitness* podem levar a busca para diferentes soluções (DAHALL *et al.*, 2008). As etapas do processo são iterativas e se alternam principalmente entre: 1) gerar e melhorar os dados de teste; 2) executar os dados de teste no programa; e 3) avaliar os dados de teste.

A primeira dificuldade para aplicar uma técnica de busca encontra-se em três elementos principais: *i*) como representar o espaço de busca de um problema, *ii*) representação das soluções candidatas; e *iii*) como definir a função de *fitness*. Na geração de dados de teste podemos definir estes elementos como:

- **problema:** Identificar um subconjunto de dados de entrada dentro um conjunto com todas as possíveis entradas para um determinado programa.
- **espaço de busca:** Todas as possíveis entradas que podem ser executadas no programa.
- **solução candidata:** valor de uma variável, parâmetros de uma função, métodos e outros.

- **função de *fitness***: maximizar a adequação de um conjunto de dados de teste para um critério de teste ou requisitos específicos a serem satisfeitos.

Figura 7 – Processo genérico de geração de dados de teste baseada em busca

Fonte: Elaborada pelo autor.

Adicionalmente, existem na literatura diversas técnicas de busca e meta-heurísticas que podem ser utilizadas em múltiplos problemas de otimização. Tais técnicas variam desde mais simples e baseadas em busca local como: Busca Gulosa, Busca A* e Subida da Encosta (do inglês, *Hill Climbing (HC)*); até mais robustas baseadas em processos naturais, como Algoritmo Genético, Otimização por Enxame de Partículas (do inglês, *Particle Swarm Optimization (PSO)*), Têmpera Simulada (do inglês, *Simulated Annealing (SA)*) e Otimização por Colônias de Formigas (do inglês, *Ant Colony Optimization (ACO)*). Nas próximas seções, algumas dessas técnicas são detalhadas.

2.5.1 Algoritmo Genético

O AG foi proposto por [Holland \(1975\)](#) na Universidade de Michigan. A ideia inicial da pesquisa era se fundamentar em processos de evolução das espécies para formalizar um algoritmo e aplicar conceitos da evolução (conhecidos como operadores genéticos, tais como, genes, cromossomos, seleção, cruzamento e mutação) para resolver problemas que possuem um amplo espaço de busca.

Em AGs a evolução é um dos princípios fundamentais para sua execução, permitindo que uma população de indivíduos (soluções) se modifique até representar a melhor solução

encontrada. Os indivíduos com as melhores características genéticas têm maiores chances de sobrevivência e de produzirem filhos (novas soluções) cada vez mais aptos, enquanto indivíduos menos aptos tendem a ser eliminados da população pela seleção natural. Em outras palavras, um AG é um modelo computacional que imita os mecanismos da evolução natural para resolver problemas de otimização.

Na década de 80, foram implementados os primeiros AGs em aplicações industriais. Goldberg, um dos alunos de Holland, foi o primeiro a desenvolver o AG e obteve excelentes resultados, permitindo o uso desse algoritmo em diversos problemas de otimização e, de difícil solução.

As soluções candidatas para um problema são codificadas por meio de uma sequência de caracteres chamada “cromossomo”. Cada cromossomo (comumente chamado de indivíduo) é formado por genes, sendo que cada gene possui uma informação sobre a resolução do problema em questão. Um conjunto de cromossomos ou indivíduos constitui uma população (Figura 8).

Figura 8 – Representação de um cromossomo

Cromossomo	0	1	0	1	0	1	1
------------	---	---	---	---	---	---	---

Fonte: Elaborada pelo autor.

A codificação binária é a mais comum e amplamente utilizada devido a sua facilidade de implementação e manipulação. Essa forma é a mais comum também por ter sido utilizada com maior ênfase pelo autor da técnica, em função da sua relativa simplicidade.

O desempenho do AG é bastante sensível aos seus parâmetros de configuração como, por exemplo, o tamanho da população. Esse parâmetro está diretamente associado ao problema que se pretende resolver. Se for definida uma população com muitos indivíduos, a aplicação do AG poderá ter um elevado custo computacional. Por outro lado, se for muito pequena ocorrerá uma redução nas possibilidades de respostas para o problema, ou seja, não haverá variedades de soluções candidatas (MITCHELL, 1998).

Na vida real, as espécies mais adaptadas ao meio ambiente são as que têm mais chances de sobreviver e passar seu código genético adiante. Para representar esse comportamento da natureza no AG, os indivíduos são submetidos a uma função de avaliação, também chamada *aptidão* ou *fitness*, a fim de avaliá-los. A função é modelada de acordo com o problema, portanto cada problema terá uma função específica que determinará quais indivíduos da população são mais aptos. Aqueles com melhor valor de *fitness* têm maior probabilidade de serem selecionados para reprodução. Essa função de avaliação é extremamente importante para o AG, pois apresenta o quanto o indivíduo está próximo da melhor solução ou de uma solução aceitável.

A evolução da população ocorre após a aplicação dos operadores de seleção, reprodução e mutação. Os operadores genéticos são a base do funcionamento do AG, permitindo que

uma população inicial se movimenta por um dado número de iterações (gerações) e consiga gerar populações sucessivas com novos indivíduos possuindo genes mais adaptados que os da geração anterior. Portanto, a população gerada é totalmente nova e possui descendentes com características de bons indivíduos das gerações anteriores, consequentemente, diversificando o universo de soluções e aumentando o espaço de busca para encontrar uma boa solução para o problema (HOLLAND, 1992).

Para realizar o cruzamento é necessário um método para selecionar alguns indivíduos da população baseado em seus valores de aptidão. Existem diversas estratégias para a seleção de indivíduos, como os métodos de: roleta, torneio e amostragem universal estocástica (MITCHELL, 1998).

A seleção por roleta é um dos métodos mais utilizados. A seleção ocorre como um sorteio; os indivíduos com maior valor de *fitness* possuem maior probabilidade de serem selecionados. Esse método simula uma roleta, e cada indivíduo possui uma probabilidade correspondente a fatia total dividida de acordo com o seu grau de aptidão, conforme apresentado no exemplo a seguir (Tabela 1).

No exemplo abaixo, o indivíduo x_1 apresenta 44,4% da somatória do *fitness*, ou seja, ele possui a maior parte da roleta, ao passo que o indivíduo x_3 possui 2,6% do total dos valores, assim ocupando a menor parte e, consequentemente, com menores chances de ser sorteado (Tabela 1).

Tabela 1 – Cálculo da porcentagem do *fitness* (roleta)

Indivíduos	<i>Fitness</i> (Aptidão)	% do Total
x_1	151	44,4
x_2	38	11,2
x_3	9	2,6
x_4	142	41,8
Total	340	100

Fonte: Elaborada pelo autor.

Após o cálculo da probabilidade do *fitness* da população a roleta é dividida proporcionalmente à porcentagem. Em seguida a roleta é girada n vezes, o número determinado de acordo com o tamanho da população. A cada giro é selecionado um indivíduo para ser efetuado o cruzamento, que é normalmente realizado em pares (Figura 9).

O cruzamento tem um papel fundamental no desempenho dos AGs. Seu principal objetivo é fazer com que os descendentes gerados herdem características dos indivíduos mais aptos da população. Dentre diversos métodos de cruzamento, pode-se destacar o “Cruzamento com um ponto de corte” que, dado um ponto de corte no cromossomo, realiza a troca dos genes dos indivíduos pais originando novos filhos da mistura de genes (Figura 10) (WEISE, 2009).

Outro operador genético presente nos AGs é a mutação, que tem como objetivo a alteração

Figura 9 – Probabilidade dos indivíduos de acordo com o *fitness* utilizando o método de seleção roleta

Fonte: Elaborada pelo autor.

Figura 10 – Representação de cruzamento com um ponto de corte

Fonte: Elaborada pelo autor.

aleatória de um gene do cromossomo. Por meio desse operador é efetuada uma mudança na constituição cromossômica do indivíduo com o aparecimento de um caractere inexistente nas gerações passadas, garantindo maior diversidade entre os indivíduos. Assim, esse processo de mutação funciona como um operador de manutenção da diversidade genética, pois modifica as características de um indivíduo aleatoriamente (Figura 11).

Figura 11 – Exemplo da operação de mutação

Fonte: Elaborada pelo autor.

Com base nas definições dos operadores citadas acima, a estrutura do AG tradicional é

apresentada no Pseudo-código 1.

Pseudo-código 1 – Algoritmo Genético

procedimento AG

Iniciar a população;

Avaliar a população;

enquanto critério de parada não for atingido **faça**

 Selecionar indivíduos para próxima população;

 Aplicar cruzamento e mutação;

 Avaliar a população;

fim enquanto

fim procedimento

Fonte: Adaptada de [Russell e Norvig \(2003\)](#).

Para garantir que determinados indivíduos não se percam durante as gerações, existem estratégias, como o elitismo, que consiste em preservar n indivíduos da população corrente, de modo que características importantes para a solução não desvançam. Essa estratégia garante que o desempenho do AG sempre aumente no decorrer das execuções, porém, dependendo do seu uso, o elitismo pode ocasionar uma convergência prematura na busca.

Os AGs têm sido amplamente aplicados em problemas de ES, em função da facilidade na compreensão e manipulação. Em teste de software, o AG tem sido utilizado em vários contextos, como geração de dados de teste ([MICHAEL; MCGRAW; SCHATZ, 2001](#)), seleção de casos de teste ([ALSMADI, 2010](#)), priorização de casos de teste ([WALCOTT et al., 2006](#)), testes não funcionais ([BRIAND; LABICHE; SHOUSHA, 2004](#)) e teste funcionais ([SINGH; R., 2010](#)).

2.5.2 Otimização por Enxame de Partículas

PSO é uma técnica de otimização global³ proposta por [Kennedy e Eberhart \(1995\)](#), que foi usada inicialmente para a otimização de funções contínuas e não lineares. A técnica é baseada no comportamento social de pássaros, peixes e outros grupos sociais, tendo como principal característica a rápida convergência. Essa característica o torna melhor ao ser comparado com muitos algoritmos de otimização global, como Têmpera Simulada e Otimização por Colônia de Formigas.

O PSO possui algumas semelhanças com as técnicas de computação evolucionária, tal como os GAs. O algoritmo é iniciado com uma população de soluções aleatórias e atualizadas a cada geração. O que a difere das demais técnicas são os operadores para evolução e atualização de cada iteração.

³ Técnicas de otimização global visam encontrar um ponto x em um conjunto de soluções X (chamado de região factível), em que uma determinada função f (chamada de função objetivo), atinge um valor mínimo ou máximo ([HORST; PARDALOS; THOAI, 2000](#)).

Essa técnica pode também ser considerada uma heurística baseada em população, consistindo de um enxame composto por um conjunto de soluções candidatas denominadas de “partículas”. Tais partículas percorrem um espaço n -dimensional (o espaço de busca do problema) e são capazes de armazenar informações passadas além de compartilhar informações com outras partículas. Essas informações são correspondentes ao aprendizado individual e social. Quando aplicado em problemas de otimização, o PSO tem alcançado soluções satisfatórias com um tempo de processamento relativamente baixo, uma de suas principais vantagens quando comparado com outras técnicas meta-heurísticas e de otimização (PARK *et al.*, 2005).

Considerando a analogia biológica, muitos animais se favorecem de algum tipo de comportamento social, seja ele para fugir de predadores, dividir tarefas, buscar alimentos e até mesmo acasalar. Ao observar um grupo de pássaros em busca de alimento, inicialmente, percebe-se que o grupo segue o pássaro que está mais próximo do objetivo, ou seja, os indivíduos compartilham a informação da busca.

Traçando um paralelo com a natureza, cada partícula do enxame (pássaro) representa uma solução para o problema no espaço de busca, e possui uma posição e velocidade (local e velocidade da busca pelo alimento). O ajuste da velocidade e posição das partículas (voo dos pássaros) é realizado de acordo com sua própria experiência e com a experiência de busca das demais partículas. A posição em que uma partícula se localiza representa de fato uma possível solução que foi encontrada pela mesma naquele momento, sendo obtida por meio da função de *fitness* que se deseja otimizar (PARSOPOULOS; VRAHATIS, 2002).

Cada partícula mantém um registro das coordenadas no espaço de busca associado ao melhor valor de *fitness* que obteve até o momento. A notação utilizada para se referir à melhor aptidão individual é o p_{best} . Por fim, o PSO mantém o registro das coordenadas do melhor valor de *fitness* encontrado entre todas as partículas, denominada “melhor solução global”, e denotado por g_{best} .

O movimento de cada partícula no espaço de busca é determinado em função de sua posição, velocidade, a melhor posição encontrada e a melhor posição encontrada pelo grupo. A cada ciclo ou iteração são atualizados os parâmetros de velocidade (v) e posição (x) de cada partícula, considerando as seguintes equações:

$$v_{t+1} = \omega * v_t + r_1 * c_1 * (bestsolution_{particle} - x_i) + r_2 * c_2 * (bestsolution_{group} - x_i) \quad (2.2)$$

$$x_{t+1} = x_t + v_{t+1} \quad (2.3)$$

onde ω é o peso de inércia e tem a finalidade de balancear a busca local e global. Para Eberhart e Shi (2000), o parâmetro ω atua sobre a velocidade fazendo com que as partículas

percorram maiores ou menores áreas do espaço de busca e seu valor geralmente decai linearmente entre 0.9 e 0.4; já r_1 e r_2 são ambas variáveis aleatórias com valores entre (0,1); e c_1 e c_2 são fatores de aceleração pessoal e global.

Considerando a ideia de que as partículas são atraídas pela melhor solução individual e global, os parâmetros c_1 e c_2 são responsáveis por controlar o quanto as partículas irão se movimentar em cada iteração. Geralmente possuem valores iguais entre (0, 2). O v_t representa a velocidade no instante t ; x_i representa sua posição atual; o $bestsolution_{particle}$ representa a melhor posição x_i visitada até então; e $bestsolution_{group}$ representa a melhor posição visitada pelo enxame.

A Equação 2.2 calcula a nova velocidade para cada partícula com base na inércia (ω), velocidade atual (v_t), posição em que a melhor solução foi encontrada até o momento (p_{best}) e a melhor solução encontrada pelo enxame (g_{best}). A atualização da velocidade do PSO consiste em três partes: “social”, “cognitivo” e dinâmico. O equilíbrio entre essas partes determina a estabilidade entre a capacidade de pesquisa global e local, influenciando diretamente no desempenho da técnica. A Equação 2.3 é utilizada para atualizar a nova posição de cada partícula, baseada na posição atual e velocidade. O PSO apesar de possuir um conceito complexo e, considerado muito eficiente em diversos contextos. O procedimento inicial para a execução do algoritmo consiste nas instruções exibidas no Pseudo-código 2.

Como na maioria dos algoritmos de otimização, saber o momento de parada é uma tarefa muito importante, pois uma parada de forma prematura pode acarretar na perda da melhor solução. Para o PSO existem três critérios comuns para o término da execução: *i*) o critério de convergência, ou seja, quando o algoritmo não consegue um melhor valor de *fitness* depois de um determinado número de iterações; *ii*) número de iterações pré-estabelecido; e *iii*) critério de parada por valor de *fitness*.

Quando comparado com outras técnicas o PSO possui algumas características interessantes. Por exemplo, o uso de memória faz com que o conhecimento de boas soluções seja mantido em todas as partículas, enquanto que em outros algoritmos como o AG, algumas informações relevantes para solução do problema podem ser perdidas no decorrer das gerações. Adicionalmente, existe uma cooperação construtiva entre as partículas, pois compartilham informações entre elas e fazem com que a busca seja muito mais eficaz (JUANG, 2004).

2.5.3 Têmpera Simulada

A Têmpera Simulada (do inglês, *Simulated Annealing* - SA) é uma técnica probabilística de busca local, proposta originalmente por Kirkpatrick, Gelatt e Vecchi (1983). Essa técnica, também conhecida como Recozimento Simulado, é análoga à termodinâmica, ao simular o arrefecimento de um metal, operação também conhecida como “recozimento”. Os procedimentos gerais dessa técnica podem ser descritos na seguinte ordem:

Pseudo-código 2 – Otimização por enxame de partículas

```

procedimento PSO(partículas, posições, velocidades)
  para cada partícula ←
    até Iniciar partícula; faça
  fim para
  enquanto critério de parada não for atingido faça
    para cada partícula ←
      até Calcular o valor de fitness  $f(p)$ ; faça
      Atualizar o melhor valor de fitness da partícula até o momento;
      se ( $f(p)$  melhor que  $p_{best}$ ) então
        Definir o valor atual como o novo  $p_{best}$ ;
      fim se
    fim para
    Atualizar o melhor valor de fitness da população até o momento;
    Definir o  $g_{best}$  com o melhor valor de fitness de todas as partículas;
    para cada partícula ←
      até calcular velocidade e posição; faça
    fim para
  fim enquanto
fim procedimento

```

Fonte: Adaptada de [Ghatei et al. \(2012\)](#).

- um metal é aquecido em uma alta temperatura, de modo que ele derreta e seus átomos possam se movimentar com maior facilidade;
- a temperatura (T) do metal derretido é lentamente reduzida de maneira que, a cada mudança de temperatura, os átomos possam se movimentar o suficiente para tomarem uma orientação mais estável; e
- permitir que os átomos do metal sejam capazes de atingir um estado de equilíbrio máximo (energia mínima), produzindo um cristal.

A ideia foi desenvolver uma técnica fundamentada no processo empregado para alterar o estado de um determinado material, ou seja, alterá-lo para um estado de energia mínima. Na Tabela 2 são exibidos os principais termos da técnica SA e sua associação com a termodinâmica.

Essa técnica inicia a busca da melhor solução a partir de uma solução inicialmente aleatória. Em seguida, a solução atual é substituída por uma solução vizinha selecionada ao acaso e avaliada por meio de uma função objetivo. O procedimento principal consiste em um laço que gera aleatoriamente em cada iteração um único vizinho da solução corrente, controlado pelo parâmetro T. Enquanto o valor de T é alto, são realizadas explorações globais no espaço de busca. Conforme T vai diminuindo, as explorações se tornam mais específicas ao redor de regiões promissoras (próximas da solução ótima). No Pseudo-código 3 é apresentada a estrutura da SA.

Tabela 2 – Associação do SA com a Termodinâmica

Termodinâmica	<i>Têmpera Simulada</i>
Estado	Solução do problema
Energia de Busca	Valor retornado pela função de avaliação
Estado de equilíbrio	Ótimo local
Estado de equilíbrio máximo	Ótimo global
Temperatura	Parâmetro de controle
Recozimento	Busca por meio da redução de T
Distribuição de Boltzmann-Gibbs	Probabilidade da seleção de um novo ponto

Pseudo-código 3 – Têmpera Simulada

procedimento SA(problema, *annealing schedule*)

Entradas: problema annealing schedule (um valor inicial para a temperatura e regras para baixá-la)

variáveis locais: solução corrente, próxima solução, T (temperatura)

corrente $\leftarrow$ criarNo(*estadoInicial*[problema])

para $t \leftarrow 1$ **to** $\infty \leftarrow$

até $T \leftarrow$ *escalonamento*[t] **faça**

se ($T = 0$) **então retorna** *corrente*

proximo $\leftarrow$ um sucessor da solução corrente selecionado ao acaso

$\Delta E \leftarrow$ Valor[*proximo*] - Valor[*corrente*]

fim se

se ($\Delta E > 0$) **então**

corrente $\leftarrow$ *proximo*

senão(*corrente* $\leftarrow$ *proximo* somente com probabilidade $e^{\Delta E/T}$)

fim se

fim para

fim procedimento

Fonte: Adaptada de [Russell e Norvig \(2003\)](#).

O parâmetro T normalmente é implementado com um decréscimo geométrico, pois um T decrescendo lentamente tem maiores chances de encontrar a melhor solução. Quando o valor de T está próximo de 0, a probabilidade da técnica encontrar soluções ruins são menores.

Como em AG e PSO, a principal vantagem da SA é a flexibilidade e robustez para ser utilizada em uma gama de problemas como um método de busca global. No entanto, a SA é uma técnica relativamente cara para determinados problemas. Para tentar suprir essa desvantagem, autores propuseram variações dessa técnica, mas que não são muito utilizadas, em função da dificuldade de implementação.

2.5.4 Subida da Encosta

A técnica Subida da Encosta (do inglês *Hill Climbing-HC*), é baseada na abstração de uma escalada no Monte Everest em meio de um nevoeiro denso durante uma crise de amnésia.

O objetivo é chegar ao pico mais alto apenas examinando os locais mais próximos (vizinhos) esquecendo os caminhos anteriores, e termina quando atingir um pico em que nenhum vizinho tenha o valor mais alto (RUSSELL; NORVIG, 2003).

Segundo Russell e Norvig (2003), a técnica HC é simplesmente um laço que se movimenta de forma contínua na direção do valor crescente e termina quando encontra o pico mais alto. O HC é uma técnica de busca heurística que combina um método de busca geral (gere-e-teste) com funções heurísticas para a avaliação dos estados gerados pelo método gere-e-teste, a fim de encontrar o melhor caminho para seguir a busca e retornar um resultado satisfatório.

Nesse contexto, o algoritmo consiste em selecionar um estado inicial (nó corrente), avaliá-lo e em seguida compará-lo com todos os estados que podem ser gerados a partir da posição atual. O melhor estado dentre todos é o escolhido para ser o novo nó corrente. Esse processo se repete até que o estado-objetivo seja alcançado ou até que a condição de parada seja verdadeira. O algoritmo não mantém uma árvore de busca, portanto somente é necessário armazenar o nó corrente e o valor de sua função objetivo. A estrutura da técnica HC é apresentada no Pseudo-código 4.

Pseudo-código 4 – Subida da Encosta

procedimento HC(problema) **retorna** uma solução

Entradas: problema

variáveis locais: solução corrente, solução vizinha

corrente $\leftarrow$ *criarNo*(*estadoInicial*[problema])

enquanto critério de parada não for atingido **faça**

vizinho $\leftarrow$ um sucessor da solução corrente com valor mais alto

se (*valor*[*vizinho*] $\leq$ *valor*[*corrente*]) **então retorna** *estado*[*corrente*]

fim se

fim enquanto

fim procedimento

Fonte: Adaptada de Russell e Norvig (2003).

A técnica HC é extremamente simples de ser implementada. Além disso, tem alcançado soluções razoáveis em grandes espaços de busca. Uma das principais desvantagens desses tipos de métodos é a possibilidade da busca estacionar em um mínimo ou máximo local que não seja a solução para o problema, pois acaba impossibilitando a exploração de outras áreas no espaço de busca. Portanto, o sucesso do HC depende muito da topologia do espaço de estados. Se existem poucos máximos ou mínimos locais, com o auxílio de reinícios aleatórios, possivelmente uma boa solução pode ser encontrada rapidamente.

2.6 Considerações Finais

Neste capítulo foram apresentados os fundamentos do teste de software, definindo detalhes sobre as técnicas e os critérios, e enfatizando o teste de mutação. Também foi discutido

o papel fundamental que tem o teste de software para que o produto final seja obtido com qualidade. Nessa perspectiva, as técnicas e critérios de teste são utilizadas para apoiar a geração e avaliação da adequação dos casos de teste para que a atividade de teste seja realizada de maneira satisfatória.

Nesse contexto, o teste de mutação é um critério de teste baseado em defeitos, que utiliza informações de erros que podem ser cometidos durante o desenvolvimento do software para criar programas semelhantes ao programa em teste. Para o teste de mutação, um conjunto de teste adequado é aquele que alcança um escore de 100%, isto é, detecta defeitos em todos os mutantes não-equivalentes e identifica um comportamento distinto entre o programa em teste e os programas mutantes. Também foram abordados neste capítulo, alguns problemas relacionados com o alto custo do teste de mutação, como a geração de dados de teste, a quantidade de mutantes gerados e a identificação de mutantes equivalentes

Neste capítulo também foram apresentados os conceitos e aspectos gerais de teste de software baseado em busca e as técnicas AG, PSO, SA e HC. Dentre as técnicas mencionadas, o AG, é uma das mais utilizadas na literatura, visto que existem inúmeras fontes (livros, artigos, tutoriais) que auxiliam no seu entendimento. O HC apesar de ser uma técnica menos robusta em relação às demais, foi uma das primeiras técnicas utilizadas para geração de dados de teste por [Korel \(1990\)](#) e ainda tem sido aplicada nos dias atuais com êxito.

Nesse contexto, a fim de tentar encontrar possíveis soluções para a automatização da atividade de geração de dados de teste no domínio de teste de mutação, um Mapeamento Sistemático foi realizado para identificar técnicas existentes na literatura para esse domínio. Os detalhes da sua condução e resultados são apresentados no capítulo seguinte (Capítulo 3).

MAPEAMENTO SISTEMÁTICO

3.1 Considerações Iniciais

A condução de uma revisão da literatura exige um protocolo sistemático e rigoroso com a finalidade de garantir a confiabilidade dos dados coletados. Nesse contexto, a Engenharia de Software baseada em Evidência (do inglês, *Evidence-Based Software Engineering (EBSE)*) utiliza métodos para coletar e analisar sistematicamente todos os dados disponíveis sobre um determinado fenômeno para obter uma perspectiva mais completa e mais ampla do que pode ser identificado por meio de um estudo individual ([KITCHENHAM; BUDGEN; BRERETON, 2015](#)).

O Mapeamento Sistemático (MS) (do inglês, *Systematic Mapping Study (SMS)*), um dos métodos que faz parte do paradigma EBSE, tem como objetivo avaliar e interpretar os estudos disponíveis para um determinado tema de pesquisa. Os resultados obtidos com o MS auxiliam na identificação de *clusters* que indicam possíveis áreas de pesquisa a serem investigadas por meio de um outro método denominado de Revisão Sistemática da Literatura (RSL) (do inglês, *Systematic Literature Review (SLR)*) ([KITCHENHAM et al., 2010](#)).

Dessa forma, foi realizado um MS para a identificar as possíveis técnicas e abordagens de geração de dados de teste no contexto de teste de mutação. Apesar de a atividade de geração de dados ser um tema bastante abordado em diversos contextos, ainda é um problema que não foi solucionado completamente dado a sua complexidade. Assim, é essencial investigar e analisar as técnicas de geração de dados de teste para teste de mutação utilizando um método de revisão mais rigoroso.

Este capítulo está organizado da seguinte forma: Seção 3.2 descreve a forma como o MS foi planejado; na Subseção 3.2.1 a estratégia de busca e as fontes de busca utilizadas neste MS são descritas; na Subseção 3.2.2 são descritos os critérios utilizados para apoiar a seleção dos estudos; na Subseção 3.2.3, o esquema de classificação das técnicas e o processo de extração

dos dados são detalhados; a Seção 5.3 apresenta o processo de seleção dos estudos primários; na Seção 3.4, as principais descobertas e as questões de pesquisas são discutidas; a Seção 5.5 apresenta as ameaças à validade do MS; por fim, a Seção 3.6 descreve as considerações finais deste capítulo.

3.2 Planejamento do Mapeamento Sistmático

Um MS é um método sistemático que visa identificar evidências relevantes para um determinado tema. O objetivo desse MS é identificar técnicas e abordagens relacionadas à geração de dados de teste para o teste de mutação. Esse MS foi conduzido baseado no processo proposto por Kitchenham *et al.* (2010), que é composta por três fases: (i) Planejamento; (ii) Condução; (iii) Análise.

O protocolo é um artefato fundamental no MS que descreve o processo e os métodos que serão adotados. Nesse protocolo são especificadas as questões de pesquisa, as bases de dados consultadas, as *strings* de busca, os critérios de inclusão e exclusão, bem como a forma que será realizada a extração dos dados (KITCHENHAM *et al.*, 2010).

As questões de pesquisa são estruturas e analisadas utilizando a técnica *Population, Intervention, Comparison e Outcomes (PICO)* (KITCHENHAM *et al.*, 2010). Neste MS foram considerados os seguintes termos:

- **população:** diz respeito a uma área de pesquisa específica. No contexto desse MS, a população é o teste de mutação;
- **intervenção:** refere-se à metodologia de software, ferramenta, tecnologia ou procedimento. No contexto desse MS, a intervenção são as técnicas para geração de dados de teste;
- **comparação:** não é aplicada no contexto desse MS;
- **resultados:** espera-se como resultados uma visão dos estudos primários que descrevem o uso de técnicas para geração de dados de teste no contexto de teste de mutação.

A partir da definição desses termos foram definidas duas questões de pesquisa:

- **QP₁.** Quais técnicas têm sido utilizadas para geração de dados de teste no contexto de teste de mutação?
- **QP₂.** Como as técnicas identificadas podem ser classificadas?

3.2.1 Estratégia de Busca

A partir da definição das questões de pesquisa, foi estabelecida a estratégia de busca dos estudos, na qual foi determinada a *string* de busca e as fontes de busca eletrônicas a serem utilizadas.

O processo de criação da *string* de busca foi definido por meio de quatro passos: (i) identificação das principais palavras-chaves a partir das questões de pesquisa; (ii) identificação dos sinônimos baseados em estudos relevantes sobre geração de dados de teste e teste de mutação; (iii) utilização do operador booleano “OR” entre os sinônimos identificados; e (iv) utilização do operador booleano “AND” para ligar as palavras-chaves (Figura 12).

Figura 12 – *String* de busca das questões de pesquisa

Fonte: Elaborada pelo autor.

Com a criação das strings, o próximo passo foi definir em quais fontes de busca a *string* deve ser executada. As fontes de busca utilizadas com base na seleção realizada por Brereton *et al.* (2007) são apresentadas na Tabela 3.

Tabela 3 – Fontes de busca

Fontes de Busca	Localização
ACM Digital Library	www.portal.acm.org
Compendex	www.engineeringvillage.com
IEEE Xplore	www.ieeexplore.ieee.org
ScienceDirect	www.sciencedirect.com
Scopus	www.scopus.com
Springer	www.springerlink.com
Web of Knowledge	www.isiknowledge.com

Fonte: Elaborada pelo autor.

3.2.2 Critérios para Seleção dos Estudos

Com a finalidade de apoiar a seleção dos estudos quanto a sua relevância e auxiliar no esclarecimento das questões de pesquisa propostas foram utilizados Critérios de Inclusão (CI) e Critérios de Exclusão (CE). A inclusão de um estudo foi determinada pela relevância, análise do título, resumo, introdução e conclusão. Nesse contexto, os seguintes critérios de inclusão e exclusão foram definidos:

- **Critérios de Inclusão:**

- **CI₁:** Estudos primários que apresentam pelo menos uma técnica/abordagem para geração de dados de teste para o teste de mutação.

- **Critério de Exclusão:**

- **CE₁:** Estudos primários que mencionam as técnicas de geração de dados de teste somente no *abstract*;
- **CE₂:** Estudos primários repetidos, neste caso, o estudo menos completo será excluído;
- **CE₃:** Estudos primários que não estejam escritos em inglês ou português; e
- **CE₄:** Estudos primários que não sejam *full paper* ou *short paper* (por exemplo, pôsters, tutoriais, relatórios técnicos, teses e dissertações);
- **CE₅:** Estudos primários com texto ou resultados incompletos.

3.2.3 Extração dos Dados

Com a finalidade de auxiliar a extração e o registro dos dados e posterior análise, todos os detalhes referentes aos estudos selecionados foram armazenados usando o software *JabRef*¹, gerenciador *open source* de referência bibliográfica. Na *JabRef* foi utilizado o recurso disponível de “exportação” que permite exportar automaticamente os seguintes detalhes de cada estudo primário: título, autor(es), resumo, palavras-chave, ano de publicação, país da instituição, bases de dados e um resumo destacando as principais contribuições.

Durante o processo de extração, informações sobre cada estudo primário foram independentemente coletadas por três pesquisadores que participaram do MS. É importante destacar que a primeira execução desse MS foi realizada em 2013. Posteriormente, foi conduzido novamente em novembro de 2016 com o objetivo de atualizá-lo.

Conforme os dados dos estudos foram extraídos e sintetizados, cada técnica identificada foi classificada de acordo com as seguintes categorias: “C₁-Baseada em Busca”; “C₂-Baseada

¹ <http://jabref.sourceforge.net/>

em sistemas de restrições”; “C₃-Apoiada pela cobertura de código”; e “C₄-Híbrida”. Essas categorias foram definidas baseadas nas características de cada técnica identificada.

A categoria “C₁-Baseada em Busca” compreende técnicas que usam heurísticas para identificar a melhor solução em um espaço de busca de um determinado problema; “C₂-Baseada em sistemas de restrições” envolve técnicas que utilizam resolução de sistemas de restrições; “C₃-Apoiada pela cobertura de código” aborda técnicas que utilizam conceitos e métodos baseados em cobertura de código, como a análise do fluxo de dados e de controle; e “C₄-Híbrida” são aquelas técnicas que combinam duas ou mais estratégias.

3.3 Execução do Mapeamento Sistemático

É importante ressaltar que durante a aplicação da *string* nas fontes a maioria das buscas foram realizadas sem nenhuma dificuldade, com exceção da base *IEEE*. Essa fonte apresenta um problema quando os termos da *string* de busca foram trocadas de posição (Figura 13). Nesse caso, a quantidade de estudos retornados é diferente do resultado da *string* na sua estrutura original (Figura 12). Para eliminar esse problema foram analisados os resultados de ambas *strings*, a fim de agrupar todos os estudos relevantes. Apesar da diferença na quantidade de estudos retornados, os estudos relevantes a partir da *string* com os termos invertidos foram contemplados nos estudos retornados da *string* na sua estrutura original.

Figura 13 – *String* de busca com os sinônimos invertidos

("mutation test" OR "mutation testing" OR "mutation analysis") AND ("test data generation" OR "test data generator" OR "data generation" OR "test case generation")

Fonte: Elaborada pelo autor.

Com a aplicação da *string* nas 7 fontes de busca selecionadas, um conjunto de estudos foi coletado, a partir do qual o MS foi conduzido. Na Figura 14 são apresentados os passos relacionados ao processo de condução desse MS.

Após a definição do protocolo realizou-se a pesquisa nas bases selecionadas (Figura 14). Na fase 1 foram retornados 439 estudos no *Scopus*, 437 estudos na *IEEE*, 198 estudos no *Compendex*, 143 estudos na *ACM*, 125 estudos no *Science*, 46 estudos no *SpringerLink* e 41 estudos no *Web of Knowledge*. Dos 1429 estudos retornados, os títulos e resumos foram lidos e neles aplicados os critérios de inclusão e exclusão (fase 2). Na fase 3, foram lidas a introdução e conclusão e os critérios de inclusão e exclusão foram aplicados nos 154 estudos selecionados na etapa anterior, resultando em 25 estudos. Por fim, na fase 4, os 25 estudos foram lidos na íntegra e 4 estudos foram excluídos, totalizando 21 estudos incluídos.

Figura 14 – Processo de seleção dos estudos

Fonte: Elaborada pelo autor.

3.4 Síntese e Análise dos Dados

Com os 21 estudos selecionados, a atividade de síntese foi realizada a partir da extração dos dados relevantes de cada estudo primário. Antes de apresentar os resultados e a análise de cada questão de pesquisa, uma breve descrição das características gerais dos estudos é realizada.

É importante destacar que os 21 estudos identificados são voltados para academia e 90,47% (19/21) realizaram experimentos para validar as técnicas/abordagens propostas.

Neste MS foram identificados, também, os países que mais têm pesquisado nessa área. A classificação dos países é baseada no número de estudos produzidos e no país de origem dos autores. Caso o estudo tenha diversos autores de países distintos, um ponto para cada país foi atribuído. Essa distribuição é apresentada na Figura 15.

Os pesquisadores de Luxemburgo e da Grécia são os que mais têm publicado nessa área, com 5 estudos cada, como autor e co-autor. Os Estados Unidos, China, Canadá e Brasil ocupam a terceira posição com 2 estudos cada. A partir dos estudos analisado, foi possível notar que 11 países têm contribuído para a geração de dados de teste para teste de mutação. A

Figura 15 – Evidência dos estudos por país

Fonte: Elaborada pelo autor.

colaboração internacional entre os pesquisadores é pouco explorada, pois somente 8 estudos do total de 21 tiveram a colaboração entre dois ou mais países. Os artigos restantes foram escritos por pesquisadores de um único país.

Com base nos estudos selecionados foram identificadas 11 técnicas para geração de dados de teste para teste de mutação. Na Figura 16 é apresentada a distribuição desses estudos de acordo com as suas respectivas técnicas e ano de publicação.

Figura 16 – Evidência dos estudos de acordo com as suas respectivas técnicas

Fonte: Elaborada pelo autor.

As duas dimensões do gráfico representam as categorias das técnicas empregadas e o ano de publicação dos trabalhos. O tamanho de cada bolha representa o número de estudos primários que foram classificados em uma determinada categoria de acordo com o seu ano de publicação. O gráfico fornece uma visão geral das pesquisas desenvolvidas ao longo dos anos.

A categoria C₁ é composta por seis técnicas distintas (*Alternating Variable Method*

(AVM), *Ant Colony Optimization* (ACO), *Bacteriological Algorithms* (BA), *Genetic Algorithm* (GA) / *Elitist GA*, *Hill Climbing* (HC), *Immune Inspired Algorithm* (IIA)); a categoria C₂ por duas técnicas (*Constraint-Based Testing* (CBT), *Distinguishing Test Cases* (DTC)); a categoria C₃ por duas técnicas (*Enhanced Control Flow Graph* (ECFG), *Improved Iterative Relaxation Method* (IIRM)); e a categoria C₄ por apenas uma técnica (*Dynamic Symbolic Execution* (DSE)).

Foi possível observar que um estudos ((HARMAN; JIA; LANGDON, 2011)) utiliza duas técnicas (HC e DSE) simultaneamente que são representadas na categoria “C₁ e C₄”. Além disso, as técnicas classificadas como “C1-Baseada em Busca” foram mencionadas a partir de 2005, pois SBST é relativamente novo.

Como pode ser visto, a categoria que tem o maior número de técnicas distintas é a categoria C₁. Nessa categoria, a técnica mais utilizada foi a AG, sendo mencionada em 6 estudos ((LOUZADA *et al.*, 2012), (HAGA; SUEHIRO, 2012), (RAD; AKBARI; BAKHT, 2010), (FRASER; ZELLER, 2010), (MISHRA *et al.*, 2010) e (MASUD *et al.*, 2005)). Vale ressaltar que somente a técnica CBT, classificada na categoria C₂, foi desenvolvida especificamente para o teste de mutação e a maioria delas são oriundas de outras áreas da Ciência da Computação.

Para obter o conhecimento sobre as técnicas utilizadas no contexto desse MS, as 11 técnicas identificadas e suas aplicações nos estudos são detalhadas a seguir.

- **C1-Baseada em Busca**

Alternating Variable Method (KOREL, 1990): foi um dos primeiros métodos utilizados em SBST e é utilizado em diversos contextos da ES, especialmente em abordagens de geração de dados de teste baseado em caminhos. O algoritmo consiste em encontrar um conjunto de valores para variáveis de entrada por meio da busca local e exploratória.

Inicialmente as variáveis de entrada (tipo numérico) de um programa são selecionadas aleatoriamente. Em seguida, os chamados movimentos exploratórios em uma vizinhança são realizados para cada variável, uma por vez. Esses movimentos são compostos por adição ou subtração de um valor na variável. Normalmente o valor inicial utilizado é 1. Se qualquer movimento leva a um valor que melhora a função objetivo, uma fase padrão é introduzida. Nessa fase, um movimento maior é efetuado na direção em que o objetivo tende a melhorar. Uma série de movimentos semelhantes é realizada até que o valor-objetivo seja encontrado para a variável. Esse processo é executado até que os melhores valores para as variáveis sejam encontrados ou todas as variáveis sejam exploradas sem sucesso. Nesse último caso, a busca pode ser reiniciada para selecionar uma nova entrada aleatoriamente, e a busca é executada até que condição de parada seja verdadeira.

Ant Colony Optimization (M.; CARO, 1999): uma abordagem baseada em inteligência de grupos sociais que é formalizada para resolver problemas de otimização e aqueles que possam

ser representados como um problema de busca. O ACO é inspirado no comportamento de forragear de algumas espécies de formigas. Durante a busca por alimento as formigas depositam no chão um tipo de feromônio para marcar um caminho favorável que deve ser seguido por outros membros da colônia. ACO explora um mecanismo análogo para resolver problemas de otimização.

O processo de obtenção da solução adota uma representação por grafos, nos quais as formigas são agentes que processam a construção da solução por meio da seleção probabilística dos nós. Cada “formiga” constrói uma solução caminhando de vértice a vértice sobre o grafo, tendo como restrição não visitar um vértice que anteriormente foi visitado pela mesma em sua caminhada. A cada passo da construção da solução, uma formiga seleciona o vértice seguinte a ser visitado de acordo com um mecanismo estocástico, que é influenciado pelo feromônio. Ao final de cada iteração, a partir da qualidade das soluções produzidas pelas formigas, os valores dos feromônios são modificados a fim de intensificá-los ou amenizá-los, para que seja possível construir soluções semelhantes baseadas nos melhores caminhos construídos em busca da melhor solução.

Genetic Algorithm (GOLDBERG, 1989): como já mencionado no Capítulo 2, o AG consiste em um processo iterativo tendo como objetivo identificar a melhor solução dentre um conjunto de soluções (população) para um determinado problema. Nesse MS, foram identificados seis estudos que utilizaram essa técnica. De um modo geral, a aplicação do AG para geração de dados de teste envolve gerar dados inicialmente aleatórios e refiná-los de acordo com o objetivo de teste, apoiado por uma função de *fitness* baseada no escore de mutação ou em outras estratégias.

Para melhorar os dados de testes gerados inicialmente são aplicados os operadores de cruzamento e de mutação, isto é, são geradas novas combinações e avaliadas pela função de *fitness* para compor a nova população. Por exemplo, dado dois indivíduos: dados de teste 1 $< 1, 2, 3 >$ e dados de teste 2 $< 4, 5, 6 >$, com o cruzamento entre eles é gerado um novo indivíduo, dados de teste 3 $< 1, 5, 3 >$. Além disso, o operador de mutação é aplicado em uma pequena porcentagem de indivíduos com o objetivo de introduzir um fator de diversidade na população. Assim, os dados de teste vão sendo melhorados até que o AG alcance um número de gerações pré-definido, levando em consideração o custo de aplicação do teste de mutação e do propósito do teste.

Bacteriological Algorithms (BAUDRY *et al.*, 2005): foi inspirado na ecologia evolutiva, estudo de organismos vivos no contexto de seu ambiente com o objetivo de descobrir como eles se adaptam (PIANKA, 1999), para melhorar conjuntos de teste. Seu conceito é baseado na ideia de que em um ambiente heterogêneo, não é possível encontrar um indivíduo que se encaixe em todo o ambiente. Isso corresponde a intuição para o problema que se visa a resolver, como: não se pode gerar um único caso de teste perfeito para matar todos os mutantes, em vez disso,

é necessário gerar e melhorar um conjunto global de casos de teste (BAUDRY *et al.*, 2005). Nesse contexto, o BA destina-se à mutação na população inicial para adaptá-la a um ambiente particular. A adaptação é baseada em pequenas alterações nos indivíduos, que são chamados de bactérias.

O BA recebe como entrada um conjunto inicial de casos de teste e gera como saída um conjunto de casos de teste melhor do que a entrada. O algoritmo evolui gradativamente e consiste de uma série de mutações (pequenas alterações) nos casos de teste para explorar o espaço de soluções. O conjunto final é construído de forma incremental, armazenando os casos de teste que podem melhorar a qualidade do conjunto, que é avaliado por meio de uma função de aptidão. A execução do algoritmo termina quando alcança um determinado critério de parada, como o número de gerações ou quando o conjunto de dados atinge um valor mínimo de aptidão.

Hill Climbing (RUSSELL; NORVIG, 2003): como apresentado no Capítulo 2, o HC é uma técnica que combina uma estratégia de busca com o uso de funções para a avaliação dos estados gerados. O HC para geração de dados de teste, consiste em gerar dados inicialmente aleatórios que vão sendo melhorados a cada iteração, considerando o melhor dado de teste de sua vizinhança.

No estudo E5, o HC é utilizado como alternativa para gerar dados de teste caso a técnica DSE (discutida a seguir) não encontre dados de teste que sejam capazes de matar os mutantes. Para avaliação desses dados de teste é utilizada uma função com base nas condições de alcançabilidade, necessidade e suficiência propostas por DeMillo e Offutt (1991).

Immune Inspired Algorithm (CASTRO; TIMMIS, 2002): o sistema imunológico humano é capaz de proteger um corpo por identificação, aprendizagem, adaptação e defesa contra organismos invasores. Sistemas imunológicos artificiais são sistemas adaptativos, inspirados no sistema imunológico humano e aplicados na resolução de problemas (CASTRO; TIMMIS, 2002).

Um IIA utiliza modelos, princípios e teorias do sistema imunológico humano como metáforas para ajudar a desenvolver novos sistemas. Uma vantagem do sistema imunológico humano é que ele pode ser ensinado a reconhecer novas infecções por meio do uso de vacinas. Uma analogia útil pode ser extraída a partir disso, um programa em teste é comparável a um corpo que se deseja proteger de infecções (falhas). Assim, uma vacinação é necessária para proteger o processo de desenvolvimento de software a partir de tais falhas (MAY; MANDER; TIMMIS, 2003).

O algoritmo consiste em um processo iterativo, que evolui a partir de uma população de anticorpos. Cada anticorpo corresponde a um dado de teste simples (por exemplo, $< 1, 2, 3 >$ para um determinado programa), buscando aqueles que podem matar pelo menos um programa

mutante que ainda não foi morto por um dado de teste existente no conjunto memória (repositório para armazenar os dados de teste adequados). Os anticorpos (testes) encontrados que podem ser úteis são adicionados à memória que deverá ser retornada para o testador ao final do processo. A evolução dos anticorpos ocorre por meio do processo de seleção clonal², guiada pelos valores de afinidade ou adequação (o valor de adequação neste caso é seu escore de mutação). Os anticorpos com elevada afinidade geram mais clones do que aqueles com baixa afinidade. Alguns anticorpos sofrerão mutação genética (alterações nos dados de teste) com taxas elevadas e inversamente proporcionais às suas afinidades (MAY; TIMMIS; MANDER, 2007).

- **C2-Baseada em sistemas de restrições**

Constraint-Based Testing (DEMILLO; OFFUTT, 1991): essa técnica utiliza os conceitos de teste de mutação para gerar os casos de teste e um sistema de restrições que representa as condições sob as quais um caso de teste pode matar um mutante. Para aplicação dessa técnica foi desenvolvido um conjunto de ferramentas denominado *Godzilla*, a fim de gerar e resolver as restrições para a criação dos casos de teste. O *Godzilla* foi integrado ao sistema de teste de mutação *Mothra* (CHOI *et al.*, 1989) com a finalidade de gerar casos de teste e matar os mutantes criados pelo *Mothra*.

Distinguishing Test Cases (NICA, 2011): essa técnica permite discriminar dois programas (programa original e mutante) por meio de uma entrada. Ambos, o programa original e os mutantes vivos, são convertidos para um problema de satisfação de restrições (*Constraint Satisfaction Problem (CSP)*), que consiste de um conjunto de variáveis com domínios em que são permitidas combinações de valores especificadas como restrições (SOININEN; GELLE, 1999).

Em seguida, um *solver* é utilizado a fim de resolver as restrições determinadas, identificando uma entrada idêntica para os dois programas, de tal modo que eles diferem em pelo menos um valor de saída. Assim, quando for encontrada uma entrada que não viole as restrições, é possível discriminar o programa original e o mutante utilizando o caso de teste gerado.

- **C3-Apoiada pela cobertura de código**

Enhanced Control Flow Graph (PAPADAKIS; MALEVRIS, 2009): tem como principal objetivo selecionar os melhores caminhos do programa para atividade de teste, ou seja, caminhos que possuam maior probabilidade em facilitar essa atividade, seja pela sua complexidade, testabilidade, eficácia ou executabilidade (LETICIA, 1999).

² A seleção clonal descreve as características básicas de uma resposta imunológica a um estímulo antigênico. Ele estabelece que apenas as células que reconhecem o antígeno são selecionadas a proliferar (gerar células clones) (CASTRO; ZUBEN, 2002).

Nesse contexto, a técnica ECFG é baseada em duas observações. A primeira de que existe uma influência significativa do esforço associado à geração de dados de teste para o teste de mutação e do número de caminhos de teste que devem ser gerados, a fim de satisfazer as condições de alcançabilidade, necessidade e suficiência dos mutantes. A segunda observação surge especificamente a partir da aplicação do teste de ramo, sendo que o critério de cobertura de ramo satisfaz a condição de alcançabilidade de todos os mutantes e nem todos para as condições de necessidade e de suficiência.

Além disso, na maioria dos casos, não há necessidade de gerar caminhos adicionais para alcançar o cumprimento de todas as condições. A abordagem proposta tenta minimizar a sobrecarga da geração de dados de teste causada pelos caminhos não-executáveis contidos no programa, evitando armadilhas ou passar muito tempo em possíveis mutantes equivalentes ou naqueles que são difíceis de serem mortos.

Improved Iterative Relaxation Method (SHAN *et al.*, 2003): consiste na melhoria do Método de Relaxamento Iterativo proposto por Gupta, Mathur e Sofia (1998), que aborda a geração de dados de teste que pode executar um determinado caminho de um programa por linearização das principais funções adotando o princípio do *program slicing*³. Assim, a técnica proposta visa a geração de dados de teste para matar múltiplos mutantes em teste de mutação. Os dados de teste são gerados com base nas informações das condições de alcançabilidade e condições de necessidade dos mutantes gerados. A ideia é baseada na observação de que as condições de alcançabilidade dos mutantes que sofreram mutação no mesmo local são as mesmas e as condições de necessidade destes mutantes são de estrutura semelhante. Assim, é possível reutilizar as condições de alcançabilidade e combinar algumas condições de necessidade para evitar a geração de um dado de teste para cada mutante (LIU *et al.*, 2006).

• C4-Híbrida

Dynamic Symbolic Execution (SEN; MARINOV; AGHA, 2005): é uma técnica também conhecida como execução *concolic* destinada à produção de dados de teste. A DSE é baseada na execução real do programa em teste, nos métodos de minimização de funções e na análise de fluxo de dados.

A técnica combina a execução concreta (real) com a execução simbólica de um programa para geração de dados de testes, isto é, o programa sob teste é executado de forma concreta e ao mesmo tempo executa computação simbólica. Assim, durante a execução concreta de um programa, ao longo de seu caminho de execução, é gerado um conjunto de restrições simbólicas. As restrições devem ser resolvidas para que sejam gerados os dados de teste adequados para

³ *Program Slicing* é o cálculo de um conjunto de instruções de programas (partes do programa), que podem afetar os valores em alguns pontos de interesse, referido como um critério de corte (KOREL, 1997)

guiar a execução do programa. Se as restrições não puderem ser resolvidas, propõe-se a simples substituição dos dados de teste por valores aleatórios (SEN; MARINOV; AGHA, 2005).

Adicionalmente, algumas vantagens e desvantagens foram constatadas baseadas nos conceitos e relatos presentes nos estudos sobre o uso das técnicas (Tabela 4). Essas informações podem servir como ponto de partida para a construção de uma abordagem, a fim de automatizar a atividade de geração de dados de teste.

3.4.1 Discussão dos Estudos

Com base nos resultados obtidos nos 21 estudos selecionados, verificou-se que vários autores tentaram reduzir custos e otimizar a atividade de geração de dados de teste para o teste de mutação. O principal objetivo consiste em reduzir o número de dados de teste necessário para matar os mutantes sem perder a eficácia do teste de mutação, pois a principal fonte de custo computacional é inerente à quantidade de dados de teste necessários para confrontar com o alto número de mutantes.

A fim de analisar a qualidade dos estudos empíricos realizados pelos estudos estão descritas na Tabela 5 informações sobre avaliações comparativas (coluna 3), o número médio de mutantes utilizados (coluna 4) e se a identificação de mutantes equivalentes (coluna 5) foi realizada, associadas aos estudos (colunas 1) e às técnicas identificadas (colunas 2).

O número de mutantes foi analisado, a fim de se obterem evidências de que as técnicas podem ser aplicadas em ambientes reais, pois a grande quantidade de mutantes gerados influencia no tempo de aplicação do teste da mutação, bem como na obtenção de um alto escore de mutação.

Com a análise dos resultados, observou-se que a maioria dos estudos estão relacionados com o uso de técnicas baseadas em busca ou propostas com base no CBT e condições que devem ser satisfeitas para matar um mutante, como descrito por DeMillo e Offutt (1991). É importante ressaltar que dos estudos que utilizaram técnicas baseadas em busca, os estudos (SOUZA *et al.*, 2016) (PAPADAKIS; MALEVRIS, 2013), (LOUZADA *et al.*, 2012), (HAGA; SUEHIRO, 2012), (HARMAN; JIA; LANGDON, 2011), (FRASER; ZELLER, 2010), (AYARI; BOUKTIF; ANTONIOL, 2007), (MAY; TIMMIS; MANDER, 2007) e (MASUD *et al.*, 2005), empregaram algum tipo de função de *fitness* especial e somente o (RAD; AKBARI; BAKHT, 2010) utilizou o escore de mutação como função para avaliar e orientar o processo de busca.

A maioria dos estudos relataram resultados satisfatórios para os experimentos que foram conduzidos, mas sem considerar variáveis como tempo, custo computacional e mutantes equivalentes. Quando um grande número de mutantes é considerado para o teste, o tempo necessário para a análise dos mutantes, geração de dados de teste e execução é mais longo. Além disso, no atual estado da arte, a maioria dos estudos identificados usam programas de laboratório para realização dos experimentos.

Com o pressuposto da utilização de programas de diferentes naturezas para possibilitar

Tabela 4 – Categorização das técnicas identificadas

Estudos	Técnicas	Vantagens	Desvantagens
(SOUZA <i>et al.</i> , 2016)	AVM	Possui baixo custo para gerar dados de teste adequados.	Utiliza apenas operadores lógicos e relacionais.
(FRASER; ARCURI, 2014)	AG	Alcança bons escores de mutação com menos tempo de execução.	Não há nenhuma garantia de que um AG vai encontrar um ótimo global.
(PAPADAKIS; MALEVRIS, 2013)	AVM/ Esquema de Fitness	Necessita de poucas execuções para alcançar uma boa cobertura.	É suscetível aos ótimos locais e exige reinicializações durante seu processo.
(LOUZADA <i>et al.</i> , 2012)	AG Elitista	Garante um aumento no desempenho e na velocidade para convergência ao longo das gerações.	Existe o risco de não explorar áreas promissoras do espaço de busca.
(HAGA; SUEHIRO, 2012)	AG	Os conjuntos de dados gerados são mais diversificados.	Não há nenhuma garantia de que um AG vai encontrar um ótimo global.
(PAPADAKIS; MALEVRIS, 2012)	ECFG	Diminui o esforço envolvido no teste de mutação, evitando a presença de caminhos não-executáveis.	A geração de mais de 19 caminhos adicionais por mutante aumenta o esforço do teste de mutação consideravelmente.
(HARMAN; JIA; LANGDON, 2011)	DSE e HC	Capacidade de atingir altos níveis de cobertura para a mutação forte e de segunda-ordem.	Abordagens híbridas tendem a ser mais custosas.
(NICA, 2011)	DTC	Pode ser capaz de melhorar um determinado conjunto de teste, como relatado no estudo (experimentos preliminares).	É dependente de um <i>solver</i> eficiente.
(PAPADAKIS; MALEVRIS; KALLIA, 2010)	ECFG	Converte o problema de matar mutantes em um problema de cobertura de ramos ou arcos do grafo.	O custo é considerado elevado quando os mutantes são analisados sem o uso do método <i>mutant schema</i> .
(RAD; AKBARI; BAKHT, 2010)	BA/AG	O BA não possui muitos parâmetros para serem ajustados.	Convergência mais lenta em relação a outras meta-heurísticas.
(FRASER; ZELLER, 2010)	AG	Os conjuntos de dados gerados são mais diversificados.	Não há nenhuma garantia de que um AG vai encontrar um ótimo global.
(PAPADAKIS; MALEVRIS, 2010a)	DSE	Pode produzir casos de teste de alta qualidade a um custo relativamente baixo.	É ineficiente na presença de laços e tarefas computacionais complexas.
(MISHRA <i>et al.</i> , 2010)	AG Elitista	Garante um aumento no desempenho e na velocidade para convergência ao longo das gerações.	Existe o risco de não explorar áreas promissoras do espaço de busca.
(ZHANG <i>et al.</i> , 2010)	DSE	Pode produzir casos de teste de alta qualidade a um custo relativamente baixo.	É ineficiente na presença de loops e tarefas computacionais complexas.
(PAPADAKIS; MALEVRIS, 2009)	ECFG	Minimiza a sobrecarga da geração de testes causados por caminhos não-executáveis do programa.	Análise de novos caminhos para aumentar a cobertura implica no aumento do custo de todo o processo.
(AYARI; BOUKTIF; ANTONIOL, 2007)	ACO	Evita a convergência para ótimos locais e alcança bons escores de mutação.	Tempo de convergência é incerto e existem parâmetros que precisam ser ajustados.
(MAY; TIMMIS; MANDER, 2007)	IIA	Bons escores de mutação.	Possuem diversos parâmetros internos que necessitam ser ajustados.
(LIU <i>et al.</i> , 2006)	IIRM	Um dado de teste gerado por meio da abordagem pode matar múltiplos mutantes.	Requer mais tempo para geração dos dados de teste em relação aos métodos tradicionais.
(MASUD <i>et al.</i> , 2005)	AG	Os conjuntos de dados gerados são mais diversificados.	Não há nenhuma garantia de que um AG vai encontrar um ótimo global.
(DEMILLO; OFFUTT, 1991) (OFFUTT, 1990)	CBT	Desenvolvido especificamente para o teste de mutação	Não pode ser usado para linguagens mais recentes como Java

Fonte: Elaborada pelo autor.

experimentos mais próximos de teste de software reais, neste MS foram coletadas informações referentes aos tipos e às quantidades de programas utilizados. Durante a análise, foram identificados os seguintes termos relacionados com os gêneros dos programas usados nos experimentos dos estudos: i) *Programas de laboratório*, que consistem de programas acadêmicos e não-comerciais; ii) *Programas industriais*, que são programas ou parte de programas do âmbito industrial e

Tabela 5 – Análise da qualidade dos estudos primários

Estudos	Técnicas	Estudo Comparativo	Número de Mutantes	Identificação de Equivalentes
(SOUZA <i>et al.</i> , 2016)	AVM	Diferentes funções de fitness e teste aleatório	1.137	134
(FRASER; ARCURI, 2014)	AG	Cobertura de Ramos	1.380,302	2636
(PAPADAKIS; MALEVRIS, 2013)	AVM/ Esquema de fitness	AVM utilizando funções de fitness diferentes e teste aleatório	345	Não informado
(LOUZADA <i>et al.</i> , 2012)	AG Elitista	AG com e sem elitismo	171	Sim
(HAGA; SUEHIRO, 2012)	AG	Com e sem o AG / geração de dados automática e manual	-	Não informado
(PAPADAKIS; MALEVRIS, 2012)	ECFG	-	150	Sim
(HARMAN; JIA; LANGDON, 2011)	DSE e HC	Variações do DSE	1.206	Não informado
(NICA, 2011)	DTC	-	-	-
(PAPADAKIS; MALEVRIS; KALLIA, 2010)	ECFG	Ferramentas JPF-SE, Concolic e eTOC	173	Sim
(RAD; AKBARI; BAKHT, 2010)	BA/AG	BA e AG	282	Não informado
(FRASER; ZELLER, 2010)	AG	Geração de dados manual e automática	20.002	Sim
(PAPADAKIS; MALEVRIS, 2010a)	DSE	-	391	Sim
(MISHRA <i>et al.</i> , 2010)	AG Elitista	-	-	-
(ZHANG <i>et al.</i> , 2010)	DSE	Ferramenta Pex para teste estrutural	235	Sim
(PAPADAKIS; MALEVRIS, 2009)	ECFG	Teste de ramo	190	Sim
(AYARI; BOUKTIF; ANTONIOL, 2007)	ACO	HC, AG e Busca Aleatória	99	Sim
(MAY; TIMMIS; MANDER, 2007)	IIA	AG	Não informado	Não informado
(LIU <i>et al.</i> , 2006)	IIRM	Ferramenta que gera dados de teste para matar um único mutante por vez	109	Sim
(MASUD <i>et al.</i> , 2005)	AG	Não realizado	-	-
(DEMILLO; OFFUTT, 1991)	CBT	Não realizado	2.083	Não informado
(OFFUTT, 1990)	CBT	-	-	-

Fonte: Elaborada pelo autor.

comercial; e *iii) Programas não-triviais*, tratam-se de programas que possuem métodos e funções mais complexas.

Na Tabela 6 são apresentadas algumas informações relacionadas com a quantidade e os tipos de programas (coluna 3) para a realização dos experimentos nos estudos, a linguagem de programação (coluna 4), o programa com mais linhas de código e o escore de mutação (colunas 5 e 6, respectivamente).

Na Tabela 7 é apresentada a análise em relação ao tempo e iterações para geração dos dados de teste mencionados nos estudos. A coluna 3 apresenta o tempo para a geração dos dados de teste e a coluna 4, o número de iterações/avaliação de *fitness* que foi definido nos estudos que utilizaram técnicas baseadas em busca. Cinco estudos ((HAGA; SUEHIRO, 2012), (HARMAN; JIA; LANGDON, 2011), (ZHANG *et al.*, 2010), (LIU *et al.*, 2006) e (DEMILLO; OFFUTT, 1991)) forneceram o tempo para a geração dos dados de teste, mas não o tempo total do processo. Quanto aos estudos que utilizaram técnicas baseada em busca, apenas o estudo (SOUZA *et al.*, 2016) forneceu informações sobre a quantidade de iterações, e o tempo necessário para o processo, apesar de serem medidas importantes para estimar parcialmente o custo de aplicação de uma técnica.

Tabela 6 – Análise dos programas utilizados nos estudos primários

Estudos	Técnicas	Número de programas	Linguagem	Maior Programa	Escore de Mutação
(SOUZA <i>et al.</i> , 2016)	AVM	18 programas	C	49 LoC	74,81%
(PAPADAKIS; MALEVRIS, 2013)	AVM/ Esquema de <i>Fitness</i>	8 programas de laboratórios	Java	90 LoC	-
(LOUZADA <i>et al.</i> , 2012)	AG Elitista	3 <i>benchmarks</i> comerciais	Java	-	92,2%
(HAGA; SUEHIRO, 2012)	AG	1 programa de laboratório	C	93 LoC	96%
(PAPADAKIS; MALEVRIS, 2012)	ECFG	30 programas de unidade	Várias linguagens	96 LoC	-
(HARMAN; JIA; LANGDON, 2011)	DSE e HC	8 programas industriais e 10 programas de laboratórios	C	9.564 LoC	Escore para mutação de primeira - ordem: 69% e mutação de segunda - ordem: 71%
(NICA, 2011)	CSP	8 classes	Java	100 LoC	-
(PAPADAKIS; MALEVRIS; KALLIA, 2010)	ECFG	10 programas de Laboratórios	Java	-	86,3%
(RAD; AKBARI; BAKHT, 2010)	BA/AG	1 programa de laboratório	Java	-	87,5% /90,6%
(FRASER; ZELLER, 2010)	AG	2 exemplos não-triviais	Java	413 classes	72%
(PAPADAKIS; MALEVRIS, 2010a)	DSE	2 exemplos não-triviais + 3 pequenos exemplos <i>siemens</i>	C	500 LoC	63%
(ZHANG <i>et al.</i> , 2010)	DSE	5 programas de algoritmos e estrutura de dados	.Net	472 LoC	Mutação fraca: 87,05% / Mutação forte: 80% (Experimentos de 1 segundo) / Mutação fraca: 89,83% / Mutação forte: 83,40%(Experimentos com mais de 1 segundo)
(PAPADAKIS; MALEVRIS, 2009)	ECFG	8 programas de unidade	Java	45 LoC	90,2%
(AYARI; BOUKTIF; ANTONIOL, 2007)	ACO	2 programas de laboratórios	Java	72 LoC	88,5%
(MAY; TIMMIS; MANDER, 2007)	IIA	4 programas de laboratórios	-	-	89,2%
(LIU <i>et al.</i> , 2006)	IIRM	5 programas de laboratórios	-	-	94,2%
(DEMILLO; OFFUTT, 1991)	CBT	5 programas de laboratórios	Fortran	55 LoC	98,4%

Fonte: Elaborada pelo autor.

Com a análise dos estudos, surgiram alguns termos utilizados que são importantes mencionar. De acordo com a literatura, o teste de mutação pode ser classificado, como Mutantes de Primeira Ordem (do inglês, *First Order Mutants (FOMs)*) ou Mutantes de Ordem Superior (do inglês, *Higher Order Mutants (HOMs)*), em relação a quantidade de defeitos inseridos. FOMs são gerados por meio da inserção de apenas falhas simples por um único operador de mutação (HARMAN; JIA; LANGDON, 2011). Ao passo que, HOMs utilizados por Jia e Harman (2008) são gerados por meio da aplicação de operadores de mutação em um programa mais de uma vez.

Neste MS, também foram identificados os conceitos de Mutação de Forte (do inglês, *Strong Mutation*) e Mutação Fraca (do inglês, *Weak Mutation*). Mutação Forte refere-se à atividade do teste de mutação original que é baseado na ideia de que se o dado de teste faz com que uma mudança de estado se propague para uma saída, na qual uma falha pode ser visivelmente observada, então o mutante é dito como “fortemente” morto. Enquanto que a Mutação Fraca

Tabela 7 – Análise do tempo e iterações para a geração dos dados de teste

Estudos	Técnicas	Tempo de geração dos dados de teste	Iterações/ Avaliação do <i>Fitness</i>
(SOUZA <i>et al.</i> , 2016)	AVM	241,05 segundos médio	1.000 avaliação de <i>fitness</i>
(PAPADAKIS; MALEVRIS, 2013)	AVM/ Esquema de <i>Fitness</i>	-	50.000 avaliação de <i>fitness</i>
(LOUZADA <i>et al.</i> , 2012)	AG Elitista	-	300/ 500/ 1000 gerações
(HAGA; SUEHIRO, 2012)	AG	137 segundos médio	-
(PAPADAKIS; MALEVRIS, 2012)	ECFG	-	1/ 50/ 300/ 5.000/ 50.000 caminhos
(HARMAN; JIA; LANGDON, 2011)	DSE e HC	307/ 301 minutos	-
(NICA, 2011)	CSP	-	-
(PAPADAKIS; MALEVRIS; KALLIA, 2010)	ECFG	-	-
(RAD; AKBARI; BAKHT, 2010)	BA/AG	-	16/ 17 gerações
(FRASER; ZELLER, 2010)	AG	-	50 gerações
(PAPADAKIS; MALEVRIS, 2010a)	DSE	-	8.395 iterações média
(ZHANG <i>et al.</i> , 2010)	DSE	1/5/10/20/40 segundos	-
(PAPADAKIS; MALEVRIS, 2009)	ECFG	-	-
(AYARI; BOUKTIF; ANTONIOL, 2007)	ACO	-	8.000/ 10.000 iterações
(MAY; TIMMIS; MANDER, 2007)	IIA	-	500 iterações
(LIU <i>et al.</i> , 2006)	IIRM	59,02 minutos	-
(DEMILLO; OFFUTT, 1991)	CBT	7 minutos	-

Fonte: Elaborada pelo autor.

executa a comparação dos resultados do programa original com o mutante imediatamente depois que a mutação inserida no programa é executada (HARMAN; JIA; LANGDON, 2011).

Alguns estudos, em adição à utilização de técnicas para a geração de dados de teste, usaram as abordagens FOMs, HOMs, mutação forte e fraca para reduzir o número de mutantes gerados e o custo computacional do teste de mutação, tal como HOMs (HARMAN; JIA; LANGDON, 2011), mutação fraca ((PAPADAKIS; MALEVRIS, 2012), (PAPADAKIS; MALEVRIS; KALLIA, 2010) , (PAPADAKIS; MALEVRIS, 2010a), (ZHANG *et al.*, 2010) e (PAPADAKIS; MALEVRIS, 2009)), mutação forte (ZHANG *et al.*, 2010) e a técnica *mutant schemata*⁴ (PAPADAKIS; MALEVRIS; KALLIA, 2010). Observou-se que tais estratégias podem ter sido utilizadas de modo a equilibrar o custo computacional necessário para o uso da ferramenta ou técnica para a geração de dados com o número de execuções associadas ao elevado número de mutantes produzidos.

3.5 Ameaças à Validade

- **Conclusão:** são ameaças identificadas no processo de seleção e extração. O processo de seleção foi realizado por meio da seleção das fontes de busca, sendo essas consideradas como as mais relevantes da área (BRERETON *et al.*, 2007), portanto propenso à inclusão da maioria dos estudos relevantes. Adicionalmente, é possível que as palavras-chave usadas para consultar as fontes de busca não representem todo o universo de interesse.

⁴ Consiste em uma abordagem para acelerar o processo de compilação e execução dos mutantes. O conjunto de mutantes gerado é inserido em um único programa denominado de *meta-mutante* (UNTCH; OFFUTT; HARROLD, 1993).

Outra ameaça constatada refere-se a forma como os dados foram extraídos a partir das fontes de busca. Em diversos estudos, alguns dados importantes não foram explicitamente descritos, obrigando os investigadores a procurarem determinadas informações em artigos relacionados, a fim de facilitar a interpretação desses estudos. No entanto, para garantir a validade, 10% dos estudos incluídos e excluídos foram analisados por outro pesquisador.

- **Interna:** essa ameaça surge por meio de alguns estudos considerados relevantes que não estão disponíveis. Além disso, a realização de uma busca manual poderia fornecer uma melhor cobertura dos estudos, consequentemente, aumentando a gama de estudos incluídos;
- **Construção:** uma ameaça dessa natureza é a possível exclusão de estudos relevantes. Para minimizar essa ameaça, foi definida uma estratégia de pesquisa rigorosa. Além disso, a fim de garantir um processo de seleção justo e evitar vieses, foi desenvolvido um protocolo de pesquisa de acordo com as diretrizes estabelecidas por [Kitchenham et al. \(2010\)](#); e
- **Externa:** essa ameaça está relacionada com a representatividade dos estudos em relação ao objetivo global desse MS. É perceptível que com o auxílio do protocolo de pesquisa foi possível obter um conjunto significativo e representativo dos estudos incluídos.

3.6 Considerações Finais

Teste de mutação é considerado um critério promissor e eficiente. No entanto, não é amplamente utilizado na prática em função de alguns problemas, tais como a geração de dados de teste. A geração de dados de teste é uma tarefa custosa, porém se sua automatização for realizada de maneira eficaz os custos dos testes podem ser significativamente reduzidos. O problema da geração de dados de teste no contexto de teste de mutação, consiste em selecionar um conjunto de dados de teste que maximiza o número de mutantes mortos.

A geração de dados de teste tem sido vista como um problema complexo, pois é necessário encontrar o equilíbrio entre um conjunto de dados de teste adequado com a quantidade de dados necessária para matar os mutantes. Assim, é possível conseguir uma redução de custos bem sucedida, permitindo, consequentemente, que o teste de mutação seja mais útil e gerenciável.

Os resultados deste MS indicam que ainda existe uma carência de pesquisas para este domínio, visto que, apesar de anos de pesquisa, foram identificados 21 estudos com 11 técnicas diferentes relacionadas à geração de dados para teste de mutação e somente alguns estudos realizaram uma avaliação completa com programas variados para simular um ambiente de teste real. Além disso, nenhum estudo determinou o nível de complexidade para aplicar e mensurar a viabilidade das técnicas.

Nesse contexto, mesmo com o aumento no desempenho do teste de mutação e o uso de técnicas promissoras, a geração de dados de teste ainda é um problema que necessita de

investigação, de modo que se alcance um nível de maturidade admissível para que as abordagens propostas sejam aplicáveis e que atinjam um desempenho confiável em problemas práticos.

Embora as abordagens identificadas nos estudos sejam um avanço para este tema, existe a necessidade de realizar experimentos em programas reais, bem como em ambientes industriais, a fim de obter evidências convincentes de que as técnicas e abordagens possam ser utilizadas na prática e, assim, motivar o uso do teste de mutação em ambientes industriais. Outra lacuna identificada está relacionada com a aplicação das técnicas, uma vez que o custo de aplicação dessas técnicas pode ser proibitivo para o projeto de teste, principalmente se considerar o custo da técnica somado ao custo do teste de mutação.

Outro fator recorrente para a escassez de abordagens para a geração de dados de teste é a ausência de ferramentas que podem automatizar essa atividade. A geração de dados de teste é considerada por muitos pesquisadores como um impedimento para a automação total ou parcial do critério de teste de mutação. Acredita-se que como trabalhos futuros a análise da viabilidade das técnicas, verificando a complexidade, tempo e custo, seja a primeira etapa para automatizar a geração de dados de teste para o teste de mutação.

RIP-MUT: UMA ABORDAGEM PARA GERAÇÃO DE DADOS DE TESTE E SUGESTÃO DE MUTANTES EQUIVALENTES

4.1 Considerações Iniciais

A geração de dados de teste é o processo de identificação de entradas válidas para um programa de acordo com os critérios de teste. Quanto maior e mais complexo um programa, mais difícil é gerar tais dados de entrada. Apesar da existência de alguns trabalhos voltados para geração de dados de teste no contexto de teste de mutação ([AYARI; BOUKTIF; ANTONIOL, 2007](#); [MISHRA *et al.*, 2010](#); [PAPADAKIS; MALEVRIS, 2011](#); [LOUZADA *et al.*, 2012](#)), este é um problema que necessita ser investigado dada sua complexidade e a baixa aplicabilidade das técnicas existentes.

Embora o teste de mutação seja amplamente considerado como muito eficaz e poderoso, também é bastante caro. O custo provém da quantidade de mutantes produzidos, do número de dados de teste necessário para matar os mutantes e da dificuldade de identificar mutantes equivalentes. Entre essas questões, a geração de dados de teste é uma das mais críticas e suscetíveis a erros, principalmente se for realizada manualmente. Por estas razões, a automação na geração de dados de teste é um passo importante para reduzir os custos do teste de mutação e, adicionalmente, aumentar a confiabilidade nos testes.

Nesse cenário, a existência de mutantes equivalentes ainda é um obstáculo para a aplicação do teste de mutação, pois constituem pontos para os quais não é possível obter uma solução, ou seja, não é possível chegar a um dado de teste que diferencie o mutante do programa em teste. Isso pode elevar o custo da atividade de teste, mesmo que se utilize a geração automática de dados de teste. Por essa razão, ao se desenvolver uma abordagem para geração de dados é necessário um método para reduzir os custos referentes à identificação dos mutantes equivalentes. Métodos

que permitam identificar os mutantes equivalentes podem auxiliar nesse processo e, por isso, foram, também, alvo de estudo nesta tese. Acredita-se que com dados de teste que sejam capazes de matar a maioria dos mutantes não-equivalentes, será possível identificar os equivalentes com mais facilidade. Em outras palavras, se o método de geração for eficiente para todos os mutantes não equivalentes, aumenta-se a certeza de que os mutantes vivos remanescentes são equivalentes.

Neste projeto foi empregado o conceito de teste de software baseado em busca, que consiste em solucionar problemas de teste utilizando técnicas de otimização baseadas em busca. Dessa forma, é possível utilizar técnicas de otimização baseadas em busca guiada por uma determinada função de *fitness* projetada para o problema abordado. Neste capítulo é apresentada a abordagem proposta. A Seção 4.2 descreve as principais características da abordagem. A Seção 4.3 apresenta a primeira parte da abordagem proposta, detalhando as funções de *fitness* utilizadas. A Seção 4.4 apresenta a segunda parte da proposta que visa auxiliar a sugestão dos mutantes equivalentes e, finalmente, a Seção 4.5 provê as considerações finais deste capítulo.

4.2 Abordagem RIP-MuT

A abordagem proposta denominada de *Reach, Infect and Propagation to Mutation Testing (RIP-MuT)* foi motivada por um estudo anterior proposto por Papadakis e Malevris (2013), o qual apresenta um método para geração de dados para o teste de mutação com o objetivo de testar programas escritos em Java. A presente abordagem estende um esquema de *fitness* com base no modelo RIP e focando principalmente na condição de propagação para programas desenvolvidos na linguagem C. Além disso, foi proposto um método para sugestão de mutantes equivalentes e incorporado no processo de geração de dados de teste.

RIP-MuT é uma abordagem para geração de dados de teste e sugestão de mutantes equivalentes que combina mutação fraca, firme e forte. A abordagem proposta automatiza o processo de geração de teste e a identificação de mutantes equivalentes, focando na produção de dados de teste que sejam capazes de matar fortemente ¹ a maioria dos mutantes não-equivalentes. A atividade de matar um mutante fracamente utilizando mutação fraca é tratada como um problema de cobertura de ramos. Dessa maneira, assumindo cada mutante como um ramo, é possível gerar dados de teste com uma ferramenta de teste estrutural. Por definição, se o um dado de teste conseguir a cobertura do ramo alvo (correspondente ao mutante), então consequentemente a instrução mutada foi executada e a condição de alcançabilidade foi cumprida.

Uma visão geral da RIP-MuT é apresentada na Figura 17, a qual é dividida em dois módulos principais: (1) geração de dados de teste e, (2) sugestão de mutantes equivalentes. O processo da abordagem é composto pelas seguintes etapas: (a) geração de dados de teste; (b) execução do conjunto de testes nos mutantes; (c) remoção dos mutantes mortos; (d) avaliação

¹ Termo referido aos mutantes mortos no teste de mutação tradicional

do escore de mutação; (e) melhorar os dados de teste gerados; e f) analisar e sugerir mutantes equivalentes. Este processo é repetido até atingir um limiar de escore predefinido.

Figura 17 – RIP-MuT: uma abordagem para geração de dados de teste e sugestão de mutantes equivalentes

Fonte: Elaborada pelo autor.

No primeiro módulo da RIP-MuT, foi utilizado um método baseado no algoritmo Subida da Encosta, conhecido como *Alternating Variable Method* (AVM) combinado com um esquema de função *fitness* composto pelas seguintes métricas: (i) *Reach Distance* (RD), (ii) *Mutation Distance* (MD), e (iii) *Impact Distance* (ID). Essas métricas são usadas para medir a adequação dos dados de teste e direcionar o processo de busca para uma solução ótima no espaço de busca. A primeira parte da função de *fitness*, RD, guia a busca para o ponto de mutação. A segunda, MD, para infectar o estado do programa no ponto de mutação, ou seja, fazer uma diferença na execução entre o mutante e o programa original, e a terceira, ID, para propagar a infecção, ou seja, manifestar o estado defeituoso para a saída do programa.

Para matar um mutante, os dados de teste devem: a) **Alcançar** a declaração mutada, b) **Infectar** o estado do programa, e c) **Propagar** a infecção para a saída do programa. Assim, a função de *fitness* proposta é usada para ajustar a direção da busca a fim de encontrar um conjunto de dados de teste que possa alcançar o maior escore de mutação possível. Cada uma das três condições mencionadas anteriormente é tratada por diferentes partes da função de *fitness*: (a) por RD, (b) por MD e (c) por ID. Assim, para matar um mutante, o dado de teste é gerado, executado e melhorado incrementalmente pelo AVM utilizando RD, MD e ID.

Na Figura 18 são ilustradas como as condições de alcançabilidade e infecção são satisfeitas por um dado de teste a partir do exemplo para o programa *cal* com as entradas *month1* = 8 e *month2* = 8.

Para as entradas definidas, a condição de alcançabilidade é satisfeita pois o dado de teste é capaz de executar a instrução mutada na linha 5. Para satisfazer a condição de infecção é

necessário que o dado de teste execute a mutação $month2! = month1$ avaliando-a com um valor diferente em relação à instrução original $month2 == month1$. Neste caso, o estado de infecção foi alcançado pois a saída de $month2 == month1$ para as entradas foi avaliada como *verdadeira* e a instrução mutada $month2! = month1$ foi avaliada como *falsa*.

Figura 18 – Exemplo de alcançabilidade para função de *fitness* RD e infecção para função de *fitness* MD

Fonte: Elaborada pelo autor.

Além das métricas de RD e MD, três medidas de impacto diferentes foram propostas para compor a função de *fitness* ID. O objetivo de uma medida de impacto é orientar a busca de dados de teste que maximizem o número de mudanças no comportamento entre o mutante e o programa original. A ideia fundamental das medidas de impacto é baseada na análise dos *traces* dos programas a fim de verificar a diferença entre suas execuções. Em outras palavras, ao comparar as instruções ou predicados condicionais, se uma instrução é executada no mutante e não executada no programa original ou vice-versa, isso sugere que o mutante foi impactado pelo dado de teste utilizado. Um alto impacto implica que um mutante é mais susceptível de ser morto do que aqueles sem impacto.

Para aumentar a chance de matar um mutante, um dado de teste deve maximizar o número de diferenças entre os *traces* dos programas. Com o intuito de aprimorar a medida de impacto, também foi proposta nesta tese uma estratégia para calcular a proximidade que um dado de teste está em fazer os predicados divergirem. A Figura 19 apresenta um exemplo do funcionamento básico de como um dado de teste pode impactar os programas.

Conforme pode ser observado, DT1 e DT2 representam diferentes dados de teste e os asteriscos representam por quais linhas o programa foi percorrido. A coluna impacto representa

original e do mutante.

Diferentes medidas de impacto geralmente podem produzir resultados distintos, uma vez que os dados de teste são capazes de expor mudanças em partes específicas dos programas. Portanto, quanto maior for a diferença nas execuções do programa, mais provável que seus comportamentos externos sejam alterados e assim, a semântica do programa afetada.

No segundo módulo da abordagem RIP-MuT, referente à sugestão de mutantes equivalentes, foi proposto um novo método com base nas análises das condições RIP, ao contrário da maioria dos estudos que exploram apenas a condição de propagação para analisar o impacto dos mutantes equivalentes em relação àqueles difíceis de serem mortos. Além do esquema de *fitness* proposto para a geração de dados de teste, o propósito também foi apresentar um método automatizado para auxiliar a detecção de mutantes equivalentes no mesmo processo. Assim, a ideia é empregar este método no processo de geração de dados de teste e analisar os mutantes que permaneceram vivos em tempo de execução para determinar se um mutante ainda pode ser morto ou se ele é um potencial equivalente.

Normalmente, existem três cenários sobre a relação entre o domínio de entrada do programa em teste e os seus mutantes:

1. mutantes fáceis de matar: existe um grande número de dados de teste que matam o mutante;
2. mutantes difíceis de matar: existem poucos dados de teste no domínio de entrada capazes de matar o mutante;
3. mutantes equivalentes: não existem dados de teste capazes de matar este tipo de mutante.

De um modo geral, o funcionamento da abordagem proposta se inicia a partir da geração de dados de teste por meio do AVM para o programa em teste. O conjunto inicial de testes é aplicado no conjunto de mutantes e verifica-se se eles foram fracamente e fortemente mortos. Os mutantes fortemente mortos são removidos e o processo de melhora dos dados de teste é iniciado. Nesta etapa, é selecionado um mutante vivo e os dados de teste gerados anteriormente são melhorados de acordo com o processo de busca e o esquema de *fitness* ou seja, RD, MD e, ID.

À medida que uma parte da função é satisfeita são analisadas as saídas dos programas visando verificar se o mutante foi morto ou se ainda permanece vivo. Este processo de melhoria segue até que mais nenhum mutante remanescente possa ser morto e neste ponto que se inicia o processo de identificação de mutantes equivalentes seguindo as seguintes etapas: 1) um mutante remanescente é selecionado e o processo de geração de dados é iniciado; 2) durante o processo de busca, uma verificação das condições RIP é realizada, o objetivo é averiguar quais das condições falharam; 3) caso todas as condições tenham sido satisfeitas, o valor de impacto do mutante é calculado e; 4) o mutante é sugerido para o testador como potencial equivalente ou difícil de ser morto com base nas análises 2 e 3.

Para que todas essas etapas de geração de dados e recomendação pudessem ser executadas, foi necessário sua implementação em uma ferramenta que automatizasse o processo. Para geração dos dados de teste foi utilizada a ferramenta *AUGmented Search-based TestING* (AUSTIN) (LAKHOTIA; HARMAN; GROSS, 2010), que é uma ferramenta de testes para programas em C que combina geração de dados com teste estrutural e uma técnica baseada em busca. Foi utilizada a implementação da AUSTIN como base para gerar dados de teste, uma vez que ela possui a estrutura básica para o desenvolvimento da nova função de *fitness* proposta neste trabalho.

A AUSTIN utiliza o critério cobertura de ramos para selecionar e avaliar os dados de teste gerados. Tratar cada mutante como um ramo ajuda o processo de geração em focar nas mutações fazendo com que uma técnica possa gerar dados adequados capazes de matá-los. No entanto, o teste de mutação requer a execução tanto do programa original quanto dos mutantes para analisar as saídas e verificar se o mutante morreu fortemente ou mesmo para analisar o comportamento interno (morte em relação a uma mutação fraca). Assim, para se beneficiar das vantagens dos critérios de teste estruturais e aplicar o teste de mutação, foi desenvolvida uma extensão da AUSTIN, de tal modo que seja possível executar os programas e identificar as mutações para matar os mutantes fracamente e fortemente. Matar os mutantes fortemente exige verificar as saídas do programa original e dos mutantes. Para isso, a AUSTIN foi utilizada em conjunto com a ferramenta de teste de mutação denominada de *Program Testing Using Mutants* (PROTEUM) (DELAMARO, 1993).

A PROTEUM é uma ferramenta de teste de mutação para a linguagem C que implementa 78 operadores de mutação e eles são divididos em quatro classes: declaração, operador, variável e constante. Ela foi utilizada na abordagem para geração dos mutantes e como base para calcular o escore de mutação.

O algoritmo de busca utilizado na abordagem foi o AVM conforme implementado pela AUSTIN. AVM foi apresentado inicialmente por Korel (1990) e tem sido bastante eficaz na geração de dados de teste para a critérios de teste estrutural (HARMAN; MCMINN, 2007), e, portanto, acredita-se que ele também pode ser adequado para o teste de mutação. A função de *fitness* original da AUSTIN é composta de duas métricas: (i) *approach level* e (ii) *branch distance*.

A abordagem RIP-MuT utiliza um esquema de *fitness* (RD & MD & ID) que guia incrementalmente o processo para gerar dados de teste que alcancem, infectam e que possivelmente matem os mutantes. A diferença de trabalhos anteriores é que ela visa matar os mutantes, focando na condição de propagação, ao invés de medidas de infecção. Assim, o esquema de *fitness* proposto mede quão perto estão os dados de teste em fazer uma divergência em cada instrução executada. Trabalhos anteriores, ou seja, (FRASER; ZELLER, 2010), (HARMAN; JIA; LANGDON, 2011) e (FRASER; ARCURI, 2014), medem o número de instruções ou predicados que divergem e não a proximidade em fazer os predicados divergirem, como apresentado anteriormente na Figura 19.

Assim, a abordagem inicia com a geração de dados de teste almejando matar uma grande proporção de mutantes, em seguida, continua gerando incrementalmente para cada um dos mutantes restantes vivos. Quando não existir mais mutantes que possam ser mortos, a abordagem analisa as condições RIP e impacto nos mutantes remanescentes para estimar se eles são equivalentes ou apenas mutantes difíceis de serem mortos. Assim, é possível direcionar os esforços de tentar matar os mutantes difíceis e indicar ao testador quais deles têm mais chance de serem equivalentes.

4.3 RIP-MuT: Geração de dados de teste

Muitos pesquisadores propuseram métodos e ferramentas para SBST focando em diferentes técnicas, tais como testes de mutação, testes estruturais e testes funcionais. Grandes esforços foram dedicados à atividade de geração de dados de teste usando técnicas baseadas em busca. Um estudo empírico realizado por [Harman e McMinn \(2010\)](#) para analisar diversas técnicas de busca, indica que um algoritmo subida da encosta pode ser mais eficaz do que outras técnicas para esse contexto e o AVM é o mais utilizado, principalmente em critérios de cobertura de ramos.

AVM é uma forma de teste dinâmico que usa informações do código-fonte combinado com funções objetivo para produzir soluções adequadas. Esse método de busca é composto por duas etapas principais, conhecidas como movimentos exploratórios e movimentos padrão. A primeira inicia selecionando valores aleatórios para as variáveis de entrada do programa. O programa apresentado na Figura 20 que tem como requisito de teste executar o ramo verdadeiro da instrução na linha 5. Os dados de teste para o programa são gerados aleatoriamente, por exemplo, $a = 10$, $b = 20$, $c = 30$. Em seguida, o AVM seleciona a primeira variável para iniciar os movimentos exploratórios que consistem em ajustes nos valores das variáveis de entrada por meio de pequenos incrementos e decrementos, normalmente um valor δ igual à 1 para inteiros e 0.1 para *floats*.

Figura 20 – Programa Exemplo

```
1 void exemplo {int a, int b, int c, int d}
2 {
3     if (a >= b)
4     {
5         if (b <= c)
6         {
7             if (c == d)
8             {
9                 // target
10            ...
11 }
```

Fonte: Elaborada pelo autor.

Neste exemplo a variável a é selecionada e os incrementos são realizados a fim de

estabelecer uma direção em que as soluções melhoram ($dir \in \{-1, 1\}$, em que -1 corresponde à direção negativa e 1 à direção positiva). Ao incrementar o valor de δ em a , foi observado uma melhora na solução, ou seja, os dados de testes estão mais próximos de executar a condição $if(a \geq b)$ e alcançar o requisito de teste na linha 5 e, portanto, corresponde a direção em incrementos positivos.

Quando os movimentos exploratórios são capazes de encontrar uma direção para a qual existem soluções melhoradas, os movimentos padrões iniciam. Esta etapa é realizada por meio de grandes movimentos, isto é, são aplicados aumentos nos valores de ajustes para acelerar a melhoria da função objetivo e para obter uma maior exploração do espaço de busca. O valor adicionado nas variáveis (a, b e c) no movimento padrão é calculado usando a fórmula $2^n * dir * \delta$. Em que n é o número de movimentos padrões (inicia com o valor 1 e continua até identificar um valor satisfatório para a variável selecionada) e $dir = 1$ correspondendo à direção positiva identificada pelos movimentos exploratórios. Assim, os valores para a variável a nos movimentos exploratórios são 12, 14, 18 e 26.

Após isso, uma busca é realizada para cada variável de entrada, até que uma solução tenha sido encontrada e mais nenhuma melhoria tenha sido obtida na função objetivo. Se não for possível encontrar melhorias para uma variável, a busca continua explorando o espaço de busca para próxima variável e caso necessário, a busca pode reiniciar com a primeira variável. Se a busca não conseguir encontrar mais nenhum caminho que leve a uma melhoria, ela é reiniciada para selecionar um outro local no espaço de busca aleatoriamente. Essa estratégia é conhecida como reinício aleatório e é utilizada para superar problemas de ótimos locais, assim como permitir que a busca possa explorar diversos pontos do espaço de busca.

Tipicamente, os dados de teste são produzidos manualmente ou aleatoriamente até que se atinja um objetivo de teste específico. No entanto, na maioria dos casos, essas práticas são pobres, tendem a ser demoradas e não têm garantia de encontrar um conjunto de dados de teste que seja capaz de atingir uma parte específica do programa ou matar alguns mutantes específicos (HARMAN; MCMINN, 2010). O uso de técnicas baseadas na busca guiadas por uma excelente função de *fitness* pode fornecer um conjunto de teste de alta qualidade.

4.3.1 Funções de fitness: RD, MD e ID

Na RIP-MuT, foi utilizado um esquema de *fitness* que combina quatro métricas. As duas primeiras foram propostas por Wegener, Baresel e Sthamer (2001), chamadas de *RD*, aplicadas para critérios de cobertura de ramos em testes estruturais e incluem as medidas *approach level* e *branch distance*. Em adição a essas métricas foram desenvolvidas as funções *MD* e *ID*. Para a distância de impacto foram propostas três novas estratégias distintas que também são utilizadas para analisar e sugerir os mutantes equivalente e difíceis de serem mortos.

- *Reach Distance (RD)*:

Para realizar o cálculo da função RD, é necessário verificar se a instrução mutada foi alcançada ou computar a distância para um dado de teste alcançá-la. Para isso são utilizadas as métricas *approach level* e *branch distance*. Para auxiliar esse processo, o critério de cobertura de ramos é aplicado, assim, sendo possível tratar as mutações como ramos, isto é, a instrução referente à mutação dentro do código fonte do mutante e do programa original é considerada como um ramo. O objetivo do critério de cobertura de ramos é selecionar um dado de teste de tal modo que cada condição em uma instrução de decisão cubra todos os resultados possíveis pelo menos uma vez. Portanto, por meio desse critério é possível verificar se uma instrução foi executada e analisar o seu comportamento nos programas.

A medida *approach level* mede o quão próximo um dado de teste está para cobrir uma determinada instrução com base no caminho seguido pela estrutura de decisão do programa, mais especificamente a medida computa a distância entre o nó do qual o fluxo se desvia e o nó alvo. Quanto menos nós dependentes de controle forem executados, mais longe a entrada estará de alcançar a instrução alvo em termos de fluxo de controle, em outras palavras, mais longe o dado de teste estará de executar o ponto de mutação.

A Figura 21 mostra como o valor do *approach level* pode ser calculado. O código-fonte à direita apresenta o programa *exemplo* e possui três predicados representados pelas estruturas de decisão *if* e à esquerda tem-se o gfc que o representa. Cada *if* pode obter um valor verdadeiro ou falso durante a execução do programa, dependendo do valor de entrada. O valor do *approach level* é calculado pelo número de predicados que afastam a execução do nó alvo. Assumindo que o alvo é o nó 3, se o dado de teste executa o predicado como falso no nó 1, então o valor do *approach level* é 2. Se o dado de teste alcançar o predicado no nó 2 com resultado falso, o valor do *approach level* é 1 e assim por diante. Para os nós em que o fluxo desvia do nó alvo (instrução avaliada como falsa), a distância necessária para o dado de teste executar a instrução como verdadeira é calculada utilizando a métrica *branch distance* proposta por Korel (1990).

Figura 21 – Medida *Approach Level*

Fonte: Adaptada de McMinn e Holcombe (2006).

A *branch distance* visa calcular o quão próximo um predicado está para ser avaliado como verdadeiro, ou seja, alternando um ramo verdadeiro para um falso ou vice-versa. Essa medida é calculada usando os valores das variáveis ou constantes nos predicados dos nós de ramificações que fizeram o fluxo da execução desviar do ramo alvo. Esta medida é calculada

seguindo as expressões apresentadas na Tabela 8.

Tabela 8 – Fórmulas para predicados lógicos e relacionais

Expressão	Ramo Verdadeiro	Ramo Falso
$a == b$	$\text{abs}(a - b)$	$a == b ? k : 0$
$a != b$	$a != b ? 0 : k$	$\text{abs}(a != b ? a - b : 0)$
$a < b$	$\text{abs}(a < b ? 0 : a - b + k)$	$\text{abs}(a < b ? a - b + k : 0)$
$a \leq b$	$\text{abs}(a \leq b ? 0 : a - b)$	$\text{abs}(a \leq b ? a - b : 0)$
$a > b$	$\text{abs}(a > b ? 0 : a - b + k)$	$\text{abs}(a > b ? a - b + k : 0)$
$a \geq b$	$\text{abs}(a \geq b ? 0 : a - b)$	$\text{abs}(a \geq b ? a - b : 0)$
$a \parallel b$	$\min[\text{fit}(a), \text{fit}(b)]$	$\text{fit}(a) + \text{fit}(b)$
$a \&\& b$	$\text{fit}(a) + \text{fit}(b)$	$\min[\text{fit}(a), \text{fit}(b)]$

Fonte: Adaptada de Papadakis e Malevris (2013).

Ainda considerando o exemplo na Figura 21, se um dado de teste executa o predicado como falso no nó 3, o valor da *branch distance* para que o dado de teste esteja próximo de executar o predicado como verdadeiro é calculado utilizando a fórmula $\text{abs}(c - d)$, uma vez que o predicado contido no *if* é do tipo $a == b$, como mostrado na coluna “expressão” da Tabela 8. Para essa medida, quanto mais próximos os valores de c e d , menor o valor da distância e consequentemente, mais próximo o ramo estará de ser executado como verdadeiro.

- *Mutation Distance (MD)*:

Para alcançar o estado de infecção, foi utilizada a função de *fitness* denominada de *Mutation Distance (MD)*, que é uma generalização do estudo proposto por Bottaci (2001), a qual foi descrita uma função de *fitness* para um algoritmo genético utilizando as condições apresentadas por DeMillo e Offutt (1991). A condição de infecção define se dado de teste pode fornecer um estado diferente no ponto de mutação em relação ao mesmo ponto no programa original.

Se esta condição for satisfeita, então o mutante difere do programa original no ponto de mutação ou em algum momento durante a sua execução e, assim, o mutante pode ser considerado como fracamente morto. Por outro lado, se a execução de um dado de teste no mutante não levar a um estado de execução diferente, a condição de infecção não foi satisfeita. Por exemplo, para infectar a expressão $a > b$ e a mutada $a \geq b$ é necessário um dado de teste igual para ambas as variáveis para obter um resultado diferente.

MD mede as distâncias dos ramos nos mutantes, ou seja, mede o quão perto os dados de teste estão de expor uma diferença na instrução mutada de acordo com a *branch distance*. Para calcular esta medida, primeiramente, é necessário quantificar a distância para obter-se uma mudança entre os predicados dos mutantes e o do programa original. Esta distância é calculada de acordo com a Expressão (4.1).

$$\begin{aligned} & (Opred == T \& \& Mpred == F) || \\ & (Opred == F \& \& Mpred == T) \end{aligned} \quad (4.1)$$

em que $Opred$ e $Mpred$, representam os predicados do programa original e dos mutantes e T e F são verdadeiro e falso, respectivamente. Se uma das condições da função for satisfeita, então o dado de teste foi capaz de infectar o mutante.

De acordo com os cálculos da métrica *branch distance* mostrados na Tabela 8, a função de *fitness* relacionada com a Expressão 4.2 é chamada *predicate mutation distance* (pmd). Quando não for possível satisfazer a Expressão 4.1, então o pmd especifica a distância dos dados de teste para fazer o predicado se comportar distintamente no ponto de mutação.

$$pmd = \min[Tfit(O) + Ffit(M), Tfit(M) + Ffit(O)] \quad (4.2)$$

em que $Tfit(O)$ e $Ffit(O)$ são os valores de *branch distance* para o predicado do programa original executado como verdadeiro e o predicado executado como falso; $Tfit(M)$ e $Ffit(M)$ são os valores de *branch distance* para o predicado do mutante executado como verdadeiro e como falso, respectivamente. Desde modo, se a busca encontrou um dado de teste que obteve $pmd = 0$, então, a instrução mutada foi infectada, pois uma diferença entre as instruções original e a mutada foi obtida. No entanto, se o pmd não for 0, os dados de teste não foram adequados para expor uma diferença de comportamento na instrução original e na mutada.

- *Impact Distance (ID)*:

Após a mutação ser executada, os dados de teste devem propagar quaisquer alterações induzidas pelo ponto de mutação de tal forma que elas possam ser observadas na saída produzida pelo mutante. Precisamente, para alcançar o estado de impacto, os dados de teste devem satisfazer duas condições: Primeiro, a mutação precisa infectar o estado (condição de necessidade) e, a mais difícil, a mutação precisa propagar a infecção para um estado observável (condição de suficiência). Estudos anteriores demonstraram que os mutantes com alto impacto são mais propensos a serem mortos em relação àqueles com menor impacto independentemente da estratégia para medir o impacto.

Além de calcular as diferenças entre as instruções, as métricas de ID propostas medem também quão perto os dados de teste estão de impactar cada predicado percorrido. Isto é realizado medindo as *branch distances* a partir do ponto de mutação até a instrução de saída do programa. Pesquisas anteriores, tais como, Fraser e Zeller (2010), Fraser e Arcuri (2014) e Harman, Jia e Langdon (2011), apenas medem o número de predicados que divergem ignorando as distâncias de ramos sobre os predicados que não têm um impacto. As três medidas de ID para compor a RIP-MuT são apresentadas a seguir.

Impact Distance 1 (ID_1): é computada por meio da Expressão 4.4, na qual *statementNum* é o número de instruções condicionais percorridas em ambos programas original e mutante. O *Impact* (Expressão 4.3), é calculado por instrução: se a saída do predicado difere entre os programas o valor do impacto é igual a 1. Caso contrário, é considerada a *branch distance* e calculado por meio da expressão 4.3, na qual M_{branch} e O_{branch} são os valores de *branch distance* para os mutantes e o programa original. A partir do *statementNum* e do valor total de impacto ($\sum Impact$) é possível demonstrar o quão distante o conjunto de dados de teste está de impactar todas as instruções percorridas, isto é, exibir uma divergência entre as instruções do mutante e do programa original.

$$Impact = \frac{abs(O_{branch} - M_{branch})}{(abs(O_{branch} - M_{branch}) + 1)} \quad (4.3)$$

$$(ID_1) = statementNum - \sum Impact \quad (4.4)$$

Impact Distance 2 (ID_2): o objetivo do ID_2 é maximizar o número de divergências entre o programa original e o mutante, considerando a diferença nos caminhos após o ponto de mutação. Portanto, se um dado de teste percorrer um caminho diferente após o ponto de mutação, significa que o mutante é susceptível de ser morto. O valor da *fitness* é calculado por meio da Expressão 4.5, em que *diffStatementNum* é o número de instruções que divergem na execução do mutante e do programa original, que é computado após a instrução mutada tenha sido executada.

$$ID_2 = diffStatementNum \quad (4.5)$$

Na Figura 22 é apresentado o programa original cal e um mutante. O impacto é calculado analisando as instruções de cada programa (original e mutante) a partir da instrução mutada como pode ser vista na linha 5. O processo inicia com a execução do dado de teste $DT1 = (2, 1, 8, 24, 1900)$ nos programas original e mutante e após a execução do ponto de mutação o impacto é computado. Caso uma instrução seja executada no programa original e não executada no mutante ou vice-versa, isso representa que a instrução foi impactada. Quanto maior for a frequência de comportamentos distintos, maior é o valor do ID_2 . Assim, para o exemplo apresentado, o valor de ID_2 é igual a 8.

Impact Distance 3 (ID_3): para calcular a terceira função proposta, foi utilizada uma medida ponderada considerando cada instrução nos mutantes. Nessa medida de impacto, o objetivo é maximizar as divergências ponderadas em cada instrução de programa (i) usando a expressão 4.6. Em que, *NoImpKilled* é o número de vezes que uma instrução i foi impactada e o mutante foi morto; *NoImpLive* é o número de vezes que a instrução foi impactada, e o mutante

Figura 22 – Exemplo da utilização da função de *fitness* ID₂

Programa Original	Dado de Teste	Mutante	Dado de Teste
	DT1		DT1
<pre> 1 int daysIn[] = {0, 31, 0, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}; 2 int cal (int month1, int day1, int month2, int day2, int year) 3 { 4 int numDays, i; 5 if (month2 == month1) 6 numDays = day2 - day1; 7 else 8 { 9 int m4 = year % 4; 10 int m100 = year % 100; 11 int m400 = year % 400; 12 if ((m4 != 0) ((m100 == 0) && (m400 != 0))) 13 daysIn[2] = 28; 14 else 15 daysIn[2] = 29; 16 numDays = day2 + (daysIn[month1] - day1); 17 for (i = month1 + 1; i <= month2-1; i++) 18 numDays = daysIn[i] + numDays; 19 } 20 return (numDays); 21 }</pre>	<pre> *</pre>	<pre> 1 int daysIn[] = {0, 31, 0, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}; 2 int cal (int month1, int day1, int month2, int day2, int year) 3 { 4 int numDays, i; 5 if (month2 != month1) 6 numDays = day2 - day1; 7 else 8 { 9 int m4 = year % 4; 10 int m100 = year % 100; 11 int m400 = year % 400; 12 if ((m4 != 0) ((m100 == 0) && (m400 != 0))) 13 daysIn[2] = 28; 14 else 15 daysIn[2] = 29; 16 numDays = day2 + (daysIn[month1] - day1); 17 for (i = month1 + 1; i <= month2-1; i++) 18 numDays = daysIn[i] + numDays; 19 } 20 return (numDays); 21 }</pre>	<pre> *</pre>
Número de instruções executadas:	10	Número de instruções executadas:	2

Fonte: Elaborada pelo autor.

permaneceu vivo; e *totalKilled* é o número de vezes que um dado de teste executa a instrução e o mutante morreu.

$$score(i) = \frac{100 * (NoImpKilled(i))}{\sqrt{totalKilled(i)(NoImpKilled(i) + NoImpLive(i))}} \quad (4.6)$$

O ID para um mutante é calculado pela soma dos escores de cada instrução por meio da Expressão 4.7.

$$ID_3 = \sum_i score(i) \quad (4.7)$$

Finalmente, a abordagem RIP-MuT é guiada por um esquema de *fitness* baseado no modelo RIP e, conseqüentemente, nas três condições necessárias para matar um mutante. Todo o processo é guiado pela função de *fitness* de RD, levando a busca para alcançar o ponto de mutação (condição de alcançabilidade); MD, para induzir um comportamento diferente no ponto de mutação em relação ao programa original no mesmo ponto (condição de necessidade); e ID para guiar a busca para os elementos do programa que podem ser impactados e, em última instância, expor o programa mutante (condição de suficiência), de acordo com a Expressão 4.8.

$$\text{esquema de fitness} = RD + MD + ID$$

$$RD = 2 * approachlevel + branchdistance$$

$$MD = pmd$$

$$ID = ID1 \quad \text{ou} \quad ID2 \quad \text{ou} \quad ID3 \quad (4.8)$$

Para ilustrar o esquema de *fitness* apresentado na expressão 4.8, o diagrama na Figura 23 detalha o processo geração de dados de teste pelo AVM utilizando RD, MD e ID.

Figura 23 – Processo da geração de dados de teste utilizando o esquema de *fitness* proposto

Fonte: Elaborada pelo autor.

O processo inicia com a geração dos dados de testes (DT) para o programa P. Em seguida, M é selecionado do conjunto total de mutantes e os dados de teste gerados são executados visando verificar se a primeira condição do modelo RIP foi cumprida.

Por meio da análise dos dados de teste no mutante, é verificado se a instrução mutada é alcançada (RD) e realizada uma das seguintes etapas: *i*) caso a condição de alcançabilidade não tenha sido cumprida, é calculada a distância com a *branch distance* e *approach level*. Baseada nessa distância, os dados de teste são melhorados pelo AVM; e *ii*) caso a condição de alcançabilidade tenha sido cumprida, então é verificado se o mutante foi morto. Se o mutante foi

morto, os dados de teste são executados em todos os mutantes vivos, caso contrário o processo continua.

A partir dos dados de testes gerados é verificado se a condição de infecção foi cumprida (MD) por meio da comparação das saídas da instrução original e da instrução mutada. Caso a instrução não tenha sido infectada, é calculada a distância utilizando o *pmd*. Baseado no valor do *pmd*, os dados de teste são melhorados pelo AVM até que a condição de infecção tenha sido cumprida ou esteja próximo disso. Caso a instrução tenha sido infectada, então é verificado se o mutante foi morto. Se o mutante foi morto, os dados de teste são executados nos mutantes remanescentes a partir da execução da etapa anterior.

Na última etapa, para aqueles mutantes que não foram mortos é calculado o impacto (ID) por meio de uma das medidas propostas. Os dados de teste são melhorados para maximizar o valor de impacto até que a condição de parada (número de gerações) seja alcançada e então é verificado se o mutante foi morto. Caso ele tenha sido morto, os dados de teste são aplicados nos mutantes vivos remanescentes, caso contrário um novo mutante é selecionado e o processo é realizado novamente.

4.4 RIP-MuT: Sugestão de mutantes equivalentes

O processo tradicional de teste de mutação consiste em: (i) gerar um conjunto de mutantes; (ii) selecionar um mutante e (iii) produzir dados de teste para tentar matá-lo. Se o mutante selecionado for morto, os dados de teste são executados em todos os mutantes e os mutantes mortos são removidos do conjunto de mutantes. Se o testador determinar um mutante como equivalente, ele também será removido. Este processo é repetido até que mais nenhum mutante não-equivalente possa ser morto.

Identificar mutantes equivalentes manualmente é uma atividade impraticável devido ao número de mutantes gerados e à complexidade para analisá-los. Ignorá-los também não é uma boa estratégia, pois quando existem mutantes equivalentes, o escore de mutação pode ser impreciso. Idealmente, uma abordagem que produz um conjunto de dados de teste de alta qualidade pode maximizar o número de mutantes mortos e, assim, o esforço manual de analisar mutantes equivalentes pode ser reduzido.

No entanto, além da existência de mutantes equivalentes, existem aqueles mutantes que são tão difíceis de serem mortos de tal forma que podem ser confundidos com mutantes equivalentes, assim, analisá-los todos manualmente é uma prática inviável. De acordo com [Namin et al. \(2015\)](#), um mutante é difícil de matar se algumas das seguintes condições forem válidas:

1. a mutação não causa um comportamento errôneo no programa;

2. a mutação causa um baixo impacto no programa, ou seja, qualquer dado de teste provoca apenas uma pequena alteração no fluxo do programa;
3. a mutação foi produzida em uma parte do programa que é difícil de executar;
4. apenas um ou poucos dados de teste no domínio de entrada podem ser capazes de matar o mutante.

Assumindo essas considerações, o objetivo do método proposto é analisar os mutantes difíceis de serem mortos e os equivalentes usando condições RIP e em particular o valor de impacto calculado durante a execução do módulo de geração de dados de teste. O modelo RIP define as três condições necessárias para matar um mutante e, de acordo com Yao, Harman e Jia (2014) implicitamente, também define as condições para um mutante ser equivalente.

O método proposto é desenvolvido com o objetivo de analisar se uma ou mais das condições falham e calcular o impacto desses mutantes que permanecem não-qualificados, isto é, aqueles difíceis de matar e equivalentes. Os mutantes equivalentes são caracterizados como possuindo muito baixo ou nenhum impacto na sua execução. Para sugerir se o mutante é difícil de ser morto ou se é equivalente, o processo inicia com a geração de dados de teste para os mutantes vivos restantes e, em tempo de execução, são analisadas as condições RIP considerando quatro premissas:

A_1 : o mutante é sugerido como equivalente se as três condições necessárias falharem para todos os casos de teste. Neste caso, a instrução mutada não pode ser alcançada, infectada e propagada por quaisquer dados de teste;

A_2 : o mutante é sugerido como equivalente se a condição de *alcançabilidade* for satisfeita, mas as condições de *infecção* e *propagação* falharem. Para esta premissa, a instrução mutada é atingida por pelo menos um dado de teste, embora nenhum dado de teste possa causar uma infecção no estado, isto é, a saída da instrução mutada não difere da saída da instrução original;

A_3 : o mutante é sugerido como equivalente se as condições de *alcançabilidade* e *infecção* forem satisfeitas, mas a *propagação* falhar. Neste caso, a instrução mutada é atingida e infectada, mas nenhuma infecção é propagada para uma saída observável;

A_4 : o mutante é sugerido como equivalente quando tem um baixo valor de impacto. Em contraste, o mutante é sugerido como difícil de matar quando tem um alto valor de impacto. Aqui, se a instrução mutada for alcançada, infectada e propagada, o valor de impacto do mutante é calculado comparando cada instrução tanto no programa original quanto no mutante.

A Figura 24 ilustra um exemplo de um mutante sugerido como equivalente de acordo com a premissa A_2 para o programa cal. Para o dado de teste $DT1 = 2, 1, 8, 24, 1900$, a condição de alcançabilidade é satisfeita (isto é, a instrução $m100 \leq 0$ foi executada). No entanto, o estado de infecção não pode ser alcançado devido às saídas das instruções terem sido avaliadas

com o mesmo valor em $m100 == 0$ e $m100 <= 0$ e, por definição, o estado infectado também não pode ser propagado.

Figura 24 – Sugerindo um mutante como equivalente utilizando a premissa A_2 para o programa cal

Programa Original	Mutantes Equivalentes
<pre> 1 int daysIn[] = {0, 31, 0, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}; 2 int cal (int month1, int day1, int month2, int day2, int year) 3 { 4 int numDays, i; 5 if (month2 == month1) 6 numDays = day2 - day1; 7 else 8 { 9 int m4 = year % 4; 10 int m100 = year % 100; 11 int m400 = year % 400; 12 if ((m4 != 0) ((m100 == 0) && (m400 != 0))) 13 daysIn[2] = 28; 14 else 15 daysIn[2] = 29; 16 numDays = day2 + (daysIn[month1] - day1); 17 for (i = month1 + 1; i <= month2-1; i++) 18 numDays = daysIn[i] + numDays; 19 } 20 return (numDays); 21 }</pre>	<pre> 1 int daysIn[] = {0, 31, 0, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}; 2 int cal (int month1, int day1, int month2, int day2, int year) 3 { 4 int numDays, i; 5 if (month2 == month1) 6 numDays = day2 - day1; 7 else 8 { 9 int m4 = year % 4; 10 int m100 = year % 100; 11 int m400 = year % 400; 12 if ((m4 != 0) ((m100 <= 0) && (m400 != 0))) * 13 daysIn[2] = 28; 14 else 15 daysIn[2] = 29; 16 numDays = day2 + (daysIn[month1] - day1); 17 for (i = month1 + 1; i <= month2-1; i++) 18 numDays = daysIn[i] + numDays; 19 } 20 return (numDays); 21 }</pre>

Fonte: Elaborada pelo autor.

Assim, a ideia deste método é reduzir o esforço para detectar os mutantes equivalentes sugerindo ao testador aqueles com mais chances de serem equivalentes e aqueles que ainda podem ser mortos. Quando um conjunto de mutantes equivalentes é removido, o módulo de geração de dados de teste pode melhorar os dados de testes focando naqueles mutantes sobreviventes que possuem o maior valor de impacto e que ainda podem ser mortos.

4.5 Considerações Finais

Este capítulo apresentou a abordagem RIP-MuT para geração de dados de teste e sugestão de mutantes equivalentes. A abordagem utiliza a técnica baseada em busca AVM, a qual é guiada por um esquema de *fitness* projetado com base nas condições necessárias para matar um mutante. O esquema de *fitness* é composto por quatro métricas, RD (sendo, *branch distance* e *approach level*), a qual se refere à condição de alcançabilidade; MD a condição de infecção; e ID, que se refere a condição de propagação. A abordagem baseia-se no cumprimento das condições do modelo RIP e, mais importante, caso um dado de teste não satisfaça uma condição, são calculadas as distâncias necessárias para que ela seja cumprida. Dessa forma, com essas informações o AVM pode gerar e melhorar dados de testes para que sejam capazes de satisfazer as condições RIP e assim, conseguir matar os mutantes não-equivalentes.

Também foi apresentado um método para auxiliar a detecção de mutantes equivalentes, sugerindo ao testador aqueles mutantes que possuem mais chances de serem equivalentes e aqueles que ainda podem ser mortos. O método é aplicado naqueles mutantes que ainda permaneceram vivos após a execução dos testes. Acredita-se que um método para identificação

de mutantes equivalentes empregado durante o processo de geração automática de dados de teste pode ser mais eficiente em relação a um método que não está incluso no processo.

Primeiramente, devido à quantidade de mutantes que necessita ser analisada normalmente ser menor, e o fato de que, identificar e remover mutantes equivalentes durante o processo de geração permite a técnica focar somente naqueles mutantes que ainda podem ser mortos.

A atividade de geração de dados de teste é considerada por muitos pesquisadores um problema clássico de busca e otimização, devido ao espaço de busca e a representação de soluções candidatas serem claramente definidas. No entanto, um dos maiores desafios é como avaliar essas soluções candidatas, ou seja, o sucesso de uma técnica baseada em busca está fortemente ligado a uma função objetivo apropriada, uma vez que um dos princípios mais fundamentais das técnicas baseadas na busca é uma combinação de funções objetivas e heurísticas para ajudar a decidir qual solução candidata deve ser testada e como a próxima solução pode ser produzida.

Uma abordagem automática de geração de dados de teste que trate tais princípios, pode fornecer subsídios para redução de custos do teste de mutação em relação à construção e execução dos conjuntos de teste. Além disso, dados de testes de qualidade permitem uma diminuição no esforço para identificação de mutantes equivalentes. Espera-se, desse modo, que a utilização da abordagem proposta para geração de dados e sugestão de mutantes equivalentes, além de reduzir os custos associados ao teste, mantenha alta eficácia em revelar a presença de erros.

No próximo capítulo é apresentado um estudo experimental para avaliar a abordagem proposta considerando ambos os métodos para geração de dados de teste e sugestão de mutantes equivalentes.

AVALIAÇÃO EXPERIMENTAL

5.1 Considerações Iniciais

No capítulo anterior foi proposta uma abordagem para a geração de dados de teste utilizando um AVM e um método para auxiliar na detecção de mutantes equivalentes.

Para validar as abordagens, neste capítulo é apresentada uma avaliação experimental focando no esquema de *fitness* proposto, na qualidade dos dados de teste, bem como no método para sugestão de mutantes equivalentes. A qualidade dos dados de teste gerados por meio da abordagem proposta também foi comparada com algoritmo genético e teste aleatório.

Este estudo empiricamente avalia a eficácia da abordagem proposta para automatizar o processo de geração de teste e para sugerir mutantes equivalentes em um conjunto de programas C e nos mutantes gerados com base nos operadores de mutação implementados na ferramenta PROTEUM. Também é avaliada a contribuição das três partes do esquema de *fitness* (RD & MD & ID) individualmente e utilizando as três diferentes medidas de impacto propostas.

Na Seção 5.2 são detalhados o planejamento e os objetivos da avaliação, além das questões de pesquisas e hipóteses a serem validadas. Na Seção 5.3 é descrito o procedimento para condução dos experimentos e na Seção 5.4 são apresentados os resultados obtidos, assim como as discussões relacionadas a eles. Por fim, na Seção 5.6 são apresentadas as considerações finais do capítulo.

5.2 Planejamento

O presente estudo tem como objetivo investigar experimentalmente o esquema de *fitness* (RD & MD & ID) com três novas medidas de impacto para geração de dados de teste. O propósito é analisar a contribuição alcançada por cada uma das partes que compõem a função de *fitness* e qual medida de impacto proposta é mais eficiente. Além disso, também é avaliada a eficiência

da abordagem para sugestão de mutantes equivalentes, focando principalmente na quantidade de sugestões verdadeiras indicadas pelo método. Dessa forma, são consideradas as seguintes Questões de Pesquisas (QPs):

QP_1 : Qual é o melhor esquema de *fitness* para gerar e melhorar dados de teste para matar mutantes?

Para responder a QP_1 , analisamos o esquema de *fitness* (RD & MD & ID) de acordo com as três medidas de impacto (ID_1 , ID_2 e ID_3), que foram denominados de FS_1 , FS_2 , FS_3 . Para geração dos dados de teste é utilizado um conjunto de mutantes oriundos dos operadores lógicos e relacionais (L-R) (mutantes L-R) implementados na PROTEUM.

Seja M o conjunto com todos os mutantes vivos e M' o conjunto dos mutantes remanescentes, isto é, um subconjunto de M com os mutantes vivos. Primeiro, são gerados e melhorados os conjuntos de testes usando o esquema de *fitness* para conseguir uma alta adequação de mutação. Neste caso, são gerados os dados de teste usando RD, e verificado quantos mutantes de M foram mortos; MD para melhorar os dados de teste para matar M' e ID para melhorar os dados de teste gerados a partir de MD, a fim de matar o restante de M' e calculado o escore de mutação final. Esse processo é repetido 10 vezes e calculada a média do escore de mutação para os programas utilizados e para cada medida de impacto diferente. Esta questão de pesquisa permite avaliar o grau de adequação dos dados de teste utilizando cada medida de impacto.

Em segundo lugar, é utilizado o melhor conjunto de dados de teste gerado pelo melhor esquema de *fitness* (aquele com melhor ID) para verificar o escore de mutação alcançado quando executado com todos os mutantes, isto é, aqueles mutantes produzidos usando todos os operadores implementados na PROTEUM. Esse experimento também foi realizado 10 vezes e calculada a média do escore de mutação para o melhor esquema de *fitness*.

Para esta questão de pesquisa as seguintes hipóteses foram definidas:

$H1_0$: Não existe diferença entre os escores de mutação fortes obtidos pelo FS_1 (μFS_1), pelo FS_2 (μFS_2) e pelo FS_3 (μFS_3), assim:

$$H1_0: \mu FS_1 = \mu FS_2 = \mu FS_3$$

$H1_1$: Existe diferença entre os escores de mutação fortes obtidos pelo FS_1 (μFS_1), pelo FS_2 (μFS_2) e pelo FS_3 (μFS_3), assim:

$$H1_1: \mu FS_1 \neq \mu FS_2 \neq \mu FS_3$$

QP_2 : Quão eficaz é o esquema de *fitness* proposto (RD & MD & ID) em comparação com a função de *fitness* RD utilizada em estudos anteriores?

Para responder a QP_2 , o esquema de *fitness* é comparado com a função RD individualmente. Neste caso, apenas a função *Reach Distance* é utilizada para verificar o número de

mutantes fracamente e fortemente mortos. Esse experimento também é executado 10 vezes e calculada a média de escore para mutação fraca e mutação forte. Para a análise, as seguintes hipóteses foram definidas para essa questão de pesquisa:

$H2_0$: Não existe diferença entre os escores de mutação fortes obtidos pela abordagem proposta (μFS) e pela função RD (μRD), assim:

$$H2_0: \mu FS = \mu RD$$

$H2_1$: Existe diferença entre os escores de mutação fortes obtidos pela abordagem proposta (μFS) e pela função RD (μRD), assim:

$$H2_1: \mu FS \neq \mu RD$$

O objetivo dessa questão é confirmar que abordagem utilizando as três condições RIP para compor a função de *fitness* pode ser melhor do que uma função utilizando somente a condição de alcançabilidade.

QP_3 : Quão eficaz é o esquema de *fitness* (RD & MD & ID) em relação à função de *fitness* (RD & MD)?

Para responder QP_3 , o esquema de *fitness* é comparado e executado de maneira semelhante a QP_2 com RD & MD. Ao usar a função de *fitness* RD & MD, primeiro é executada RD para gerar um conjunto de testes para tentar matar os mutantes M . Em seguida, MD para produzir um novo conjunto de testes por meio da melhoria dos dados de teste gerados anteriormente pelo RD, a fim de matar os mutantes M' . Esse experimento é executado 10 vezes e calculado o escore de mutação fraca e forte. Para esta questão, foram definidas as seguintes hipóteses:

$H3_0$: Não existe diferença entre os escores de mutação fortes alcançados pela abordagem proposta (μFS) e pela função RD e MD ($\mu RD\&MD$), assim:

$$H3_0: \mu FS = \mu RD\&MD$$

$H3_1$: Existe diferença entre os escores de mutação forte alcançados pela abordagem proposta (μFS) e pela função RD e MD ($\mu RD\&MD$), assim:

$$H3_1: \mu FS \neq \mu RD\&MD$$

QP_4 : Quão eficaz é o esquema de *fitness* para gerar e melhorar dados de teste para matar mutantes em comparação com algoritmo genético e teste aleatório?

A fim de responder a QP_4 , é realizada uma comparação entre o esquema de *fitness* com o melhor ID, obtido a partir da QP_1 , com o algoritmo genético e Teste Aleatório (TA). Utilizou-se um algoritmo genético empregando o escore de mutação como função de *fitness* para avaliar a

adequação dos dados de teste gerados. Primeiro, são gerados dados de teste usando o esquema de *fitness*, algoritmo genético e testes aleatórios para matar os mutantes L-R. Em um segundo experimento, os dados de teste obtidos são executados no conjunto de mutantes referentes a todos operadores para verificar suas adequações. Para esta questão, foram definidas as seguintes hipóteses:

$H4_0$: Não existe diferença entre os escores de mutação fortes obtidos pela abordagem proposta (μFS), pelo algoritmo genético (μAG) e pelo teste aleatório (μTA), assim:

$$H4_0: \mu FS = \mu AG = \mu TA$$

$H4_1$: Existe diferença entre os escores de mutação fortes obtidos pela abordagem proposta (μFS), pelo algoritmo genético (μAG) e pelo teste aleatório (μTA), assim:

$$H4_1: \mu FS \neq \mu AG \neq \mu TA$$

Com os resultados dessa questão se almeja demonstrar que existem diferenças significativas entre as abordagens comparadas e que o escore de mutação obtido pela abordagem proposta é mais adequado em relação aos escores do algoritmo genético e teste aleatório.

QP_5 : Quão eficiente é a abordagem proposta em relação ao tempo de geração de dados de teste para testes de mutação?

A eficiência refere-se ao tempo de geração de dados de teste. O tempo é computado para cada esquema de *fitness* e inclui todos os tempos para gerar dados de teste e sua melhoria. O tempo total (T) é calculado pela expressão 5.1.

$$T = time(RD) + \sum time(MD) + \sum time(ID_x) \quad (5.1)$$

onde $time(RD)$ é o tempo de geração de dados de teste para o programa original, $time(MD)$ é a soma do tempo de melhoria dos dados do teste usando MD para cada mutante vivo selecionado. Finalmente, o $time(ID_x)$ é a soma do tempo de melhoria dos dados do teste utilizando cada medida de ID para os mutantes vivos selecionados. O tempo é gravado por meio de uma aplicação nativa do Linux chamada *time utility* em um laptop com processador Intel Core i7 1.8 GHz CPU, 8GB de memória no sistema operacional Ubuntu 14.04

QP_6 : Quão eficiente é a abordagem proposta em relação à quantidade de dados gerados comparado com algoritmo genético e teste aleatório?

O número de dados de testes necessários para matar os mutantes também contribui para o custo da abordagem, pois cada teste deve ser identificado, armazenado e executado diversas vezes. Assim, o objetivo da questão QP_6 é avaliar o esquema de *fitness* considerando a quantidade de dados de testes necessários para alcançar o escore de mutação. Para isso foram armazenados

os dados de testes gerados para cada abordagem em 10 execuções diferentes e calculada a média das execuções. Para esta questão as seguintes hipóteses foram definidas:

$H6_0$: Não existe diferença entre a quantidade de dados gerados pela abordagem proposta (μFS), pelo algoritmo genético (μAG) e pelo teste aleatório (μTA), assim:

$$H6_0: \mu FS = \mu AG = \mu TA$$

$H6_1$: Existe diferença entre a quantidade de dados gerados pela abordagem proposta (μFS), pelo algoritmo genético (μAG) e pelo teste aleatório (μTA), assim:

$$H6_1: \mu FS \neq \mu AG \neq \mu TA$$

QP_7 : Quão eficaz é o método proposto em sugerir mutantes equivalentes?

Para responder a QP_7 , é analisado o método para auxiliar na detecção de mutantes equivalentes focando na quantidade de sugestões corretas para mutantes equivalentes e mutantes difíceis de serem mortos. Neste método, também analisamos as condições RIP no processo de geração de dados de teste no conjunto de mutantes que permanecem vivos visando verificar aqueles com mais chances de serem equivalentes e aqueles que ainda podem ser mortos.

Na primeira parte do experimento, verificou-se a relação entre as condições RIP e os mutantes analisados para sugerir se o mutante é difícil de matar ou equivalente. Para isso, as sugestões foram classificadas como: (i) *True Suggestion (TS)* e, (ii) *False Suggestion (FS)*.

Para distinguir as sugestões dos mutantes realizadas corretamente e incorretamente, são utilizadas as seguintes definições apresentadas na literatura (MANNING; RAGHAVAN; SCHUTZE, 2008):

- *True Suggestion equivalent (TS_{eq})*: refere-se aos mutantes equivalentes classificados **corretamente** como possíveis equivalentes;
- *False Suggestion equivalent (FS_{eq})*: refere-se aos mutantes difíceis de serem mortos classificados **incorretamente** como possíveis equivalentes;
- *True Suggestion hard-to-kill (TS_{hk})*: refere-se aos mutantes difíceis de serem mortos classificados **corretamente** como possíveis mutantes difíceis de serem mortos;
- *False Suggestion hard-to-kill (FS_{hk})*: refere-se aos mutantes equivalentes classificados **incorretamente** como possíveis mutantes difíceis de serem mortos;

Para a segunda parte do experimento, é analisada a hipótese de que mutantes com alto impacto são mais susceptíveis de serem mortos, portanto, não-equivalentes. A eficiência do método proposto está diretamente relacionada com as sugestões verdadeiras para detectar

mutantes equivalentes e mutantes difíceis de matar. O total de sugestões verdadeiras (TS) é calculado pela expressão 5.2.

$$TS = \frac{\sum TS_{eq} + \sum TS_{hk}}{RM} \quad (5.2)$$

onde TS_{eq} é o total de sugestões verdadeiras para indicar um mutante como equivalente. TS_{hk} consiste no total de sugestões corretas para revelar os mutantes difíceis de matar. Finalmente, RM representa o conjunto de mutantes remanescentes.

Para avaliar a capacidade do método proposto em sugerir mutantes equivalentes, foram utilizadas as seguintes métricas: precisão (do inglês, *precision*) e revocação (do inglês, *recall*). Para calcular a precisão (P) e a revocação (R) do método são utilizadas as expressões 5.3 e 5.4, respectivamente.

$$P = \frac{TS_{eq}}{TS_{eq} + TF_{eq}} \quad (5.3)$$

$$R = \frac{TS_{eq}}{TS_{eq} + TF_{hk}} \quad (5.4)$$

A métrica de precisão quantifica a capacidade do método em sugerir mutantes corretamente como equivalentes, enquanto que, o valor de revocação representa o percentual dos mutantes que, a partir do conjunto de mutantes sugeridos corretamente como equivalentes, são de fato equivalentes.

5.2.1 Definição dos objetivos

Para definir os objetivos desse experimento foi utilizado o modelo *Goal/Question/Metric* (GQM) proposto por Basili e Weiss (1986). Esse modelo apresenta o experimento dividido em cinco partes: objeto de estudo, propósito, perspectiva, foco qualitativo e contexto.

- **Objeto de estudo:** o objeto do estudo é a abordagem proposta.
- **Propósito:** o propósito deste experimento é avaliar a abordagem proposta considerando a qualidade dos dados de teste gerados e a sugestão de mutantes equivalentes.
- **Perspectiva:** este experimento é executado do ponto de vista de um pesquisador.
- **Foco qualitativo:** o principal efeito sob investigação é a eficácia da abordagem que é medida pelo score de mutação e a eficiência que é medida pelo tempo e a quantidade de dados de teste.

- **Contexto:** este experimento foi realizado utilizando 19 programas em C.

O experimento pode ser resumido utilizando o seguinte modelo proposto por Wohlin *et al.* (2012): **Analisar** a abordagem proposta; **com o propósito de** avaliar a qualidade dos dados testes gerados e a sugestão de mutantes equivalentes; **com respeito à** “eficácia” e “eficiência”; **do ponto de vista de** um pesquisador; **no contexto de** 19 programas em C.

5.2.2 Definição das variáveis

Para todos os experimentos realizados foram controladas seis variáveis independentes: (i) função de *fitness* RD, (ii) função de *fitness* RD & MD, (iii) esquema de *fitness* (FS) RD & MD & ID; (iv) FS com melhor impacto; (v) AG e; (vi) TA.

As variáveis dependentes coletadas a partir das funções de *fitness* e abordagens são os conjuntos de teste gerados e as métricas que deles se podem extrair como: escore de mutação, número de caso de teste, tempo de geração, etc.

5.2.3 Seleção dos sujeitos

O estudo realizado utiliza no total 19 sujeitos de diferentes domínios e tamanhos variados. A Tabela 9 apresenta os detalhes sobre os programas. Esses programas variaram em linhas de código de 7 a 49 linhas, totalizando 314 linhas de código. A coluna *Programas* representa os nomes dos programas e a coluna *LOC* exibe a quantidade de linhas de código. Estes programas foram escolhidos pois estavam implementados em C e foram utilizados em diversos estudos empíricos envolvendo teste de mutação, como Delamaro *et al.* (2014), Souza *et al.* (2016).

5.3 Condução

O procedimento realizado durante a execução do experimento consiste nas seguintes etapas: 1) geração de dados de teste para matar mutantes lógicos e relacionais utilizando o esquema de *fitness* (RD & MD & ID) avaliando três medidas de impacto (ID_1 , ID_2 , ID_3); 2) execução dos dados de teste gerados na etapa anterior em um conjunto com mais mutantes gerados por 78 operadores de mutação; 3) comparação da abordagem proposta com algoritmo genético e teste aleatório; e 4) análise das condições RIP e medida de impacto para sugestão de mutantes equivalentes. Esses experimentos foram realizados em oito etapas:

1. **Seleção dos sujeitos:** foram selecionados 19 programas em C, tal como $P = (p_1, p_2, p_3, \dots, p_{19})$.
2. **Geração dos mutantes:** o conjunto de mutantes $M = (m_1, m_2, \dots, m_x)$ é gerado por meio da ferramenta PROTEUM. O m_i é o conjunto de mutantes gerados para cada programa p_i .

Tabela 9 – Sujeitos do experimento

Programas	LOC
Cal	18
checkIt	9
countPositive	9
DigitReverser	17
findLast	10
findVal	7
InversePermutation	15
Jday—jdate	49
lastZero	9
NumZero	10
oddOrPos	9
power	11
printPrimes	35
quicksort	23
RecursiveSort	17
stats	19
sum	7
testPad	24
twoPred	10

Fonte: Elaborada pelo autor.

obtido primeiramente por operadores lógicos e relacionais (L-R), como pode ser visto na Tabela 10, e então para um segundo experimento, produzido por todos os 78 operadores de mutação implementados na PROTEUM.

3. **Geração de dados de teste:** os dados de teste $T = (t_1, t_2, \dots, t_n)$ são gerados usando a ferramenta AUSTIN adaptada ao teste de mutação e a função de *fitness* proposta. Esses dados de teste são gerados iterativamente até que um dado de teste obtenha um valor de *fitness* adequado para uma determinada parte da função. Para o processo de geração de teste são considerados até 1000 avaliações de *fitness* para selecionar dados de teste para cada programa p_i . Esta etapa é dividida em três sub-etapas:

- a) **RD *fitness*:** os dados de teste são gerados para o programa original pelo AVM usando o RD para obter cobertura de ramo, isto é, cada ramo no programa original é avaliado como verdadeiro e falso se for possível. Após isso, os dados de teste são aplicados nos mutantes e o escore de mutação atualizado da seguinte maneira: i) se uma entrada executa o ramo verdadeiro para o programa original e falsa para o mutante ou o oposto para uma instrução o mutante é considerado fracamente morto, e ii) as saídas do programa original e do mutante são analisadas, caso sejam diferentes, o mutante é considerado morto fortemente;
- b) **MD *fitness*:** um mutante é selecionado dentre aqueles que estão vivos e os dados de

testes são melhorados pelo AVM usando a medida MD. Se os dados de teste atingirem um valor de *fitness* igual a zero, isso significa que o mutante foi fracamente morto e são executadas as seguintes etapas: i) as saídas do programa original e do mutante são analisadas, caso sejam diferentes, o mutante é considerado morto fortemente, e ii) os dados de teste são executados no restante dos mutantes vivos. Caso contrário, se o mutante estiver vivo, outro mutante é selecionado e o processo reinicia. Esta etapa é realizada até que o maior número de mutantes tenha sido alcançado. Depois disso, o escore de mutação forte e fraco são atualizados e os mutantes mortos são removidos;

- c) **ID *fitness*:** um mutante é selecionado dentre aqueles que estão vivos e os dados de teste são melhorados com o AVM usando uma medida de ID (ID_1 , ID_2 ou ID_3). Os dados do teste com os melhores valores de *fitness* são executados nos mutantes vivos e verifica-se se foram mortos fortemente, em seguida, o escore de mutação é computado.
4. **Cálculo do MS:** o total do escore de mutação é calculado após a aplicação dos subpassos anteriores. Para avaliação do módulo de geração de dados de teste, apresentado na Seção 4.3, os mutantes equivalentes foram identificados e removidos manualmente com a finalidade de calcular o real escore de mutação obtido pela abordagem.
 5. **Cálculo do tempo:** o tempo total (em segundos) de geração de dados de teste usando a abordagem proposta é computado por meio da expressão 5.1.
 6. **Geração dos dados usando algoritmo genético:** é gerado um conjunto de dados de teste por meio de um algoritmo genético para matar m . Neste AG foi empregado o escore de mutação como função de *fitness*. Essa etapa é dividida em três sub-etapas:
 - a) o AG é iniciado com os seguintes parâmetros: População de 100 indivíduos, taxa de cruzamento de 0,5, taxa de mutação de 0,2 e taxa de elitismo de 0,7. Esses parâmetros foram baseados em outros estudos e por meio da realização de testes para verificar a melhor combinação;
 - b) geração da população inicial;
 - c) avaliação de *fitness* para cada indivíduo da população;
 - d) repetir esse item até que a condição de parada seja satisfeita, ou seja, atingir 1000 avaliações de *fitness*. Para isso, primeiro são selecionados os indivíduos mais aptos para reprodução. Após isso, o AG cria novos indivíduos por meio de cruzamento e operações de mutação. Em seguinte, é realizada a avaliação de *fitness* dos novos indivíduos. Finalmente, a substituição da população menos apta por novos indivíduos.

7. **Geração dos dados usando teste aleatório:** os dados de testes são gerados 10 vezes para cada programa e executados no conjunto de mutantes. Para cada execução é calculado o escore de mutação e, ao final, foi computada a média utilizando os escores obtidos.
8. **Comparação entre a abordagem RIP-MuT, o algoritmo genético e o teste aleatório:** o escore de mutação para os dados de teste gerados por cada abordagem foi comparado, a fim de verificar qual abordagem é mais eficaz.
9. **Sugestão de mutantes equivalentes:** o método para sugerir mutantes equivalentes é executado e avaliado. Esta etapa é dividida em 8 sub-etapas:
 - a) gerar os dados de teste usando a abordagem proposta;
 - b) executar os dados de teste gerados nos mutantes;
 - c) remover os mutantes mortos;
 - d) melhorar os dados de teste e executar o passo 2 até que mais nenhum mutante seja morto;
 - e) analisar as condições RIP nos mutantes remanescentes conforme descrito na Seção 4.4;
 - f) calcular e analisar a medida de impacto naqueles mutantes que não falharam as condições RIP;
 - g) sugerir ao testador um conjunto de mutantes potencialmente equivalentes;
 - h) verificar manualmente se os mutantes potencialmente equivalentes são de fato equivalentes;
 - i) remover os mutantes equivalentes;
 - j) calcular o escore de mutação.

5.4 Análise dos Dados

Nesta seção, as QP_s são respondidas e apresentadas a partir da análise dos resultados referentes à eficácia e eficiência da abordagem proposta. Para responder as QP_s foi utilizada a média dos escores de mutação para uma determinada abordagem ou parte da função de *fitness*. Foram utilizados também o tempo médio e a quantidade de dados de teste média para a avaliação de eficiência. Para a abordagem de sugestão de mutantes equivalentes, foram utilizadas as informações com base nas sugestões verdadeiras e falsas obtidas nos experimentos.

5.4.1 Análise Descritiva

Esta seção consiste em uma análise descritiva dos dados coletados durante a execução do experimento. A primeira análise a ser realizada está relacionada a eficácia da abordagem.

Tabela 10 – Operadores lógicos e relacionais

Operators	Description
ORRN	Relational Operator by Relational Operator
ORAN	Relational Operator by Arithmetic Operator
ORSN	Relational Operator by Shift Operator
ORBN	Relational Operator by Bitwise Operator
ORLN	Relational Operator by Logical Operator
OLSN	Logical Operator by Shift Operator
OLRN	Logical Operator by Relational Operator
OLLN	Logical Operator by Logical Operator
OLAN	Logical Operator by Arithmetic Operator
OLBN	Logical Operator by Bitwise Operator
OLNG	Logical Negation
OCNG	Logical Context Negation

Fonte: Elaborada pelo autor.

- **Melhor esquema de *fitness* (QP_1):**

Essas questões consideram a eficácia da abordagem proposta avaliada referente à geração de dados de teste. A Figura 25 apresenta o escore de mutação alcançado pelos esquemas de *fitness* usando cada medida de impacto proposta (FS_1 , FS_2 , FS_3). O eixo x representa os programas utilizados como sujeitos e o eixo y representa o escore médio de 0 a 100. Em geral, as eficiências dos esquemas quando usam diferentes medidas de impacto são muito similares e se mostram relativamente constantes. A função de *fitness* utilizando a medida ID_3 que é representada por FS_3 obteve a maior pontuação. Em contraste, em alguns programas os conjuntos de dados de teste produzidos pelas abordagens FS_1 , FS_2 , FS_3 mataram a mesma quantidade de mutantes, como pode ser visto na Tabela 11.

Para fornecer mais detalhes sobre a QP_1 , a Tabela 11 apresenta algumas informações em relação a cada FS. A Tabela 11 exibe o número de mutantes, o número de mutantes mortos de acordo com cada esquema de *fitness*, isto é, FS_1 , FS_2 e FS_3 por programa. É possível notar que na maioria dos programas o número de mutantes mortos não sofreu muitas mudanças, o melhor esquema de *fitness* (FS_3) foi capaz de matar em média 86,02% dos mutantes para todos os programas.

Como apresentado na QP_1 , a segunda parte da avaliação da abordagem visa verificar a adequação dos dados de teste gerados pelo melhor esquema de *fitness* (FS_3) quando aplicado em um conjunto maior de mutantes. Os resultados obtidos são apresentados na Tabela 12 para os

Figura 25 – Escore de Mutação de cada esquema de *fitness*

Fonte: Elaborada pelo autor.

Tabela 11 – Número de mutantes mortos por cada esquema de *fitness*

Programas	Mutantes	Equivalentes	Número de mutantes mortos		
			FS1	FS2	FS3
Cal	127	19	70	97	99
checkIt	41	3	25	27	28
countPositive	36	3	29	29	29
DigitReverser	36	16	19	19	19
findLast	36	1	19	20	20
findVal	36	3	28	28	28
InversePermutation	117	18	94	92	94
Jday-jdate	36	0	33	33	33
lastZero	36	3	31	31	31
NumZero	36	3	30	30	30
oddOrPos	110	28	68	68	68
power	36	1	29	32	32
printPrimes	90	10	76	71	76
quicksort	127	9	71	83	83
RecursiveSort	48	2	42	39	42
stats	54	9	42	40	42
sum	18	4	13	12	13
testPad	110	7	89	89	93
twoPred	51	2	32	32	32
Total	1.181	141	840	872	892

Fonte: Elaborada pelo autor.

dois conjuntos de operadores (L-R operadores e 78 operadores) respectivamente. Essa tabela exhibe os dados de teste (coluna DT), o número de mutantes (coluna Num. de Mut.), número de mutantes equivalentes (coluna Equiv.), número de mutantes mortos (coluna Mortos) e o escore de mutação (coluna %MS).

Analisando os resultados, pode-se observar que o conjunto de dados de teste gerado para matar os mutantes L-R, também é adequado para matar os mutantes gerados com todos

Tabela 12 – Adequação dos dados de teste gerados para mutantes L-R e aplicados nos mutantes derivados de 78 operadores

Programas	DT	Mutantes L-R				Todos Mutantes			
		Mutantes	Equiv.	Mortos	%MS	Mutantes	Equiv.	Mortos	%MS
Cal	7	127	19	99	91,67	891	67	748	90,78
checkIt	6	41	3	28	73,68	104	3	73	72,28
countPositive	3	36	3	29	87,88	151	9	118	83,10
DigitReverser	3	36	16	19	95,00	496	43	424	93,60
findLast	6	36	1	20	57,14	198	17	148	81,77
findVal	7	36	3	28	84,85	190	18	153	88,95
InversePermutation	6	117	18	94	94,95	576	71	464	91,88
Jday-jdate	4	36	0	33	91,67	2821	81	2390	87,23
lastZero	5	36	3	31	93,94	173	9	149	90,85
NumZero	5	36	3	30	90,91	151	17	122	91,04
oddOrPos	3	110	28	68	82,93	361	71	229	78,97
power	5	36	1	32	91,43	268	12	241	94,14
printPrimes	7	90	10	76	95,00	715	64	642	98,62
quicksort	3	127	9	83	70,34	1026	82	881	93,33
RecursiveSort	4	48	2	42	91,30	555	45	466	91,37
stats	6	54	9	42	93,33	884	101	708	90,42
sum	4	18	4	13	92,86	165	11	143	92,86
testPad	11	110	7	93	90,29	629	57	520	90,91
twoPred	10	51	2	32	65,31	246	24	200	90,09
Total	105	1.181	141	766	-	10.600	802	8.819	-
Média	5,52	62,15	7,42	54,71	86,02	557,89	42,21	464,15	89,06

Fonte: Elaborada pelo autor.

os operadores uma vez que o escore de mutação atingido pelo FS foi em média 89,06%. Isso pode ser explicado devido o conjunto de dados de teste ser gerado com auxílio da terceira parte do esquema de *fitness* (ID). A medida de impacto auxilia na produção de dados de teste que também são capazes de alcançar requisitos de outros mutantes em relação aos mutantes L-R, ou seja, maximizar o impacto em um mutante pode produzir um dado de teste que pode percorrer diferentes partes do programa e elementos no código que não são executados comumente, e dessa forma, matar mutantes não executados antes.

Para atestar a confiabilidade dos dados obtidos na QP_1 foi realizado teste de hipótese. Como não se espera que as amostras utilizadas tenham distribuição normal, foi utilizado teste não paramétrico para verificar as hipóteses. Para responder QP_1 foi necessário realizar um comparativo entre 3 grupos (FS_1 , FS_2 e FS_3). Para realizar essa comparação foi aplicado o teste *Kruskal-Wallis test* (KRUSKAL, 1947). A hipótese nula H_0 a ser testada trata que as médias das amostras são iguais. Portanto, H_0 somente é rejeitada se o teste produzir resultados significativos, reforçando a hipótese alternativa H_1 que defende que os dados analisados são estatisticamente diferentes.

Na QP_1 foram analisados os escores de mutação do esquema de *fitness* com diferentes medidas de impacto (FS_1 , FS_2 , FS_3). De acordo com o teste realizado, levando em consideração as 10 execuções, o valor de significância é maior que a margem de erro ($H(2) = 1.210666$ e $0.545892 > 0.05$), indicando que não existe uma diferença significativa entre os escores de mutação obtidos por FS_1 , FS_2 e FS_3 , assim, confirmando a hipótese H_{10} . Apesar de os resultados, apresentados na Tabela 11, indicarem que existe uma diferença entre os escores

de mutação alcançados por FS_1 , FS_2 e FS_3 , não é possível confirmar estatisticamente essa diferença.

Para responder a QP_2 foram gerados dados de testes somente para satisfazer RD e comparado com o esquema de *fitness* (RD & MD & ID) considerando a melhor medida de impacto obtida anteriormente. Os resultados são apresentados na Figura 26. Nessa Figura, as barras representam a média do escore de mutação obtido para cada combinação de função de *fitness* (RD; RD & MD; RD & MD & ID).

Figura 26 – Escore de mutação forte das funções de *fitness* RD & MD & ID, RD & MD e RD

Fonte: Elaborada pelo autor.

Conforme pode ser observado na Figura 26, o esquema de *fitness* melhora o escore de mutação em todos os programas comparados com a função de *fitness* RD. Além disso, o escore de mutação aumenta à medida que uma nova função de *fitness* é adicionada com a finalidade de gerar e melhorar os dados do teste.

- **Eficácia de RD & MD & ID x RD (QP_2) e RD & MD & ID x RD& MD (QP_3):**

Para fornecer detalhes adicionais em relação a QP_2 e QP_3 , os números de mutantes mortos por cada combinação da função de *fitness* são apresentados na Tabela 13. A Tabela mostra o número de mutantes (segunda coluna), o número de equivalentes (terceira coluna), os mutantes mortos fracamente e fortemente pelos dados de testes utilizando RD (quarta e quinta coluna, respectivamente), os mutantes mortos fracamente e fortemente utilizando RD & MD (sexta e sétima coluna) e o número de mutantes mortos fortemente pelo esquema de *fitness* RD & MD & ID na oitava coluna.

Os dados de teste gerados usando a função RD & MD são capazes de matar fracamente 877 mutantes, enquanto 680 mutantes foram mortos fortemente. É natural esperar que o escore de mutação fraco obtido seja maior ou igual ao escore de mutação forte, uma vez que um dado de teste que mata fortemente um mutante deve matá-lo fracamente, embora, não necessariamente, o contrário.

Tabela 13 – Número de mutantes mortos por cada função de *fitness*

Programas	Mutantes	Equiv.	RD		RD & MD		RD & MD & ID
			Fraco	Forte	Fraco	Forte	Forte
Cal	127	19	49	34	80	50	99
checkIt	41	3	22	22	25	24	28
countPositive	36	3	18	15	23	19	29
DigitReverser	36	16	12	11	21	17	19
findLast	36	1	13	2	30	18	20
findVal	36	3	15	8	32	24	28
InversePermutation	117	18	47	30	85	56	94
Jday-jdate	36	0	26	26	33	31	33
lastZero	36	3	26	23	33	28	31
numZero	36	3	26	23	33	28	30
oddOrPos	110	28	50	38	84	62	68
power	36	1	19	18	30	25	32
printPrimes	90	10	46	34	61	59	76
quicksort	127	9	38	31	73	48	83
RecursiveSort	48	2	24	20	38	35	42
stats	54	9	27	22	35	31	42
sum	18	4	12	9	17	11	13
testPad	110	7	54	49	96	87	93
twoPred	51	2	33	24	48	27	32
Total	1.181	141	557	439	877	680	892
Média	62,15	7,42	29,31	23,10	46,15	35,78	46,94

Fonte: Elaborada pelo autor.

A Figura 27 indica que mais mutantes podem ser mortos fortemente à medida que uma nova função do esquema de *fitness* é utilizada. Essa avaliação foi realizada visando verificar se o escore de mutação obtido por meio do esquema de *fitness* em um tempo específico (obtido a partir da execução de RD & MD & ID) também poderia ser obtido por RD & MD e apenas RD no mesmo tempo. A ideia é mostrar que as funções de *fitness* são complementares e não podem obter melhores resultados individualmente. Para essa avaliação foi considerado o tempo do esquema de *fitness* utilizando a medida de impacto ID_1 .

Figura 27 – Número de mutantes mortos pelas funções de *fitness* para um determinado tempo

Fonte: Elaborada pelo autor.

Note-se que pela adição de uma nova função de *fitness* a busca é conduzida a caminhos não atingidos pelos dados de teste iniciais. Além disso, considerando que o ramo alvo no

mutante foi atingido, essa ramificação possivelmente será percorrida no programa original. Assim, analisando os caminhos e as saídas dos programas é possível determinar se o mutante foi morto ou se os dados de teste falharam.

Para avaliar a confiabilidade dos dados coletados nas questões QP_2 e QP_3 foi aplicado o *Wilcoxon test* com a finalidade de comparar a diferença do esquema de *fitness* (FS_3) com as funções de *fitness* (RD) e (RD & MD). O *Wilcoxon test* é um teste estatístico geralmente utilizado para verificar se existem diferenças significativas entre os resultados de duas amostras.

Nas questões de pesquisa QP_2 e QP_3 , a hipótese nula H_{20} e H_{30} para o *Wilcoxon test* indicam que não existem diferenças entre os escores de mutação obtido pelo esquema de *fitness* (RD & MD & ID₃) em relação as funções de *fitness* RD e RD & MD respectivamente. Portanto, se o *Wilcoxon test* produzir evidências sobre a diferença entre RD & MD & ID₃ e RD (QP_2) e RD & MD & ID₃ e RD & MD (QP_3), ou seja, rejeitando H_{20} e H_{30} respectivamente, então as hipóteses alternativas H_{21} e H_{31} que consideram diferenças nos escores de mutação alcançados podem ser assumidas. Os resultados do *Wilcoxon test* são apresentados na Tabela 14.

Tabela 14 – Resultados do teste de *Wilcoxon* sobre a diferença entre os escores de mutação obtidos pelo esquema e cada função de *fitness*

Funções de <i>fitness</i> comparadas	<i>z</i>	<i>p-value</i>
RD - RD & MD & ID ₃	-3,823394	0,000132
RD & MD - RD & MD & ID ₃	-3,823007	0,000132

Fonte: Elaborada pelo autor.

Conforme pode ser visto na Tabela 14, as funções de *fitness* obtiveram *p-values* inferiores a margem de erro (0,05), então, com nível de confiança de 95%, existem evidências das diferenças entre os escores de mutação alcançados por RD, RD & MD e RD & MD & ID ($z = -3,823394$, $z = -3,823007$ e $0,000132 < 0.05$). Portanto, as hipóteses nulas H_{20} e H_{30} são refutadas e as hipóteses alternativas, que definem que o esquema de *fitness* alcança um escore de mutação melhor foram confirmadas.

- **Comparação entre FS, AG e teste aleatório (QP_4):**

A eficácia das abordagens foi avaliada por meio de um experimento comparativo entre FS, AG e teste aleatório, em termos de escore de mutação. Os resultados são apresentados nas Figuras 28 e 29 e, nas Tabelas 15 e 16. A Figura 28 e a Tabela 15 apresentam detalhes sobre os mutantes L-R e os conjuntos de testes gerados. A Figura 29 e a Tabela 16 referem-se aos resultados quando todos os mutantes são utilizados.

A Figura 28 apresenta a média dos escores de mutação obtida por FS, AG e teste aleatório para os mutantes L-R. As barras representam a média dos escores de mutação para cada abordagem e o eixo x os programas utilizados. Os resultados obtidos sugerem que o conjunto de

testes gerado pela abordagem proposta é sempre mais eficiente do que AG e geração com teste aleatório para os mutantes L-R. O resultado deste experimento também sugere que conjuntos de testes gerados por técnicas que usam apenas o escore de mutação como função de *fitness* podem não alcançar alguns requisitos de teste, mesmo que seja uma técnica robusta como algoritmo genético.

Figura 28 – Comparação entre FS, AG e Teste aleatório para mutantes L-R

Fonte: Elaborada pelo autor.

Além disso, como pode ser analisado na Figura 28, o resultado mostra que FS atinge um alto escore de mutação usando os operadores L-R. O escore de mutação atingido pelo FS foi 15,24% maior do que AG e 30,85% maior do que o teste aleatório. Nota-se também que em 18 dos 19 programas, FS obteve melhores escores de mutação do que AG. No entanto, um programa, *twoPred*, obteve o mesmo escore de mutação (65,31%) pelo FS e AG e o programa *findLast* atingiu o mesmo escore de mutação (48,57%) pelo AG e teste aleatório.

A Tabela 15 mostra com mais detalhes as informações do estudo comparativo, destacando o número de mutantes (segunda coluna), número de mutantes equivalentes (terceira coluna), número de mutantes mortos, dados de teste e escore de mutação para a abordagem proposta, AG e teste aleatório (quarta, quinta e sexta coluna).

Para avaliar as diferentes abordagens, também foi verificada a adequação dos dados de teste gerados por cada uma para os mutantes L-R quando aplicados no conjunto maior de mutantes gerados por 78 operadores.

A Figura 29 exibe a média dos escores de mutação obtida pela abordagem proposta (FS), AG e teste aleatório para matar os mutantes oriundos de 78 operadores de mutação. Foi possível observar que o FS obteve a maior média de escore de mutação com 89,06% em relação ao AG que obteve um escore de 70,81% e o teste aleatório com um escore de 53,13% para

Tabela 15 – Número de Mutantes L-R mortos por FS, AG e teste aleatório

Programas	Num. de Mut.	Equiv.	FS			AG			Teste Aleatório		
			Mortos	DT	EM	Mortos	DT	EM	Mortos	DT	EM
Cal	127	19	99	7	91,67	43	7	39,81	30	11	27,78
checkIt	41	3	28	6	73,68	25	8	65,79	14	9	36,84
countPositive	36	3	29	3	87,88	26	4	78,79	28	7	84,85
DigitReverser	36	16	19	3	95,00	15	4	75,00	18	10	90,00
findLast	36	1	20	6	57,14	17	8	48,57	17	7	48,57
findVal	36	3	28	7	84,85	25	7	75,76	19	7	57,58
InversePermutation	117	18	94	6	94,95	64	8	64,65	52	8	52,53
Jday-jdate	36	0	33	4	91,67	22	5	61,11	16	10	44,44
lastZero	36	3	31	5	93,94	29	5	87,88	18	6	54,55
NumZero	36	3	30	5	90,91	29	5	87,88	18	6	54,55
oddOrPos	110	28	68	3	82,93	60	6	73,17	39	5	47,56
power	36	1	32	5	91,43	28	4	80,00	24	7	68,57
printPrimes	90	10	76	7	95,00	70	10	87,50	27	15	33,75
quicksort	127	9	83	3	70,34	74	5	62,71	73	9	61,86
RecursiveSort	48	2	42	4	91,30	33	4	71,74	39	8	84,78
stats	54	9	42	6	93,33	31	4	68,89	26	6	57,78
sum	18	4	13	4	92,86	11	3	78,57	12	5	85,71
testPad	110	7	93	11	90,29	74	12	71,84	31	11	30,10
twoPred	51	2	32	10	65,31	32	11	65,31	13	12	26,53
Total	1.181	141	766	105	1.634,48	597	120	1.344,97	428	159	1.048,33
Média	62,15	7,42	54,71	5,52	86,02	31,42	6,31	70,78	22,53	8,36	55,17

Fonte: Elaborada pelo autor.

Figura 29 – Comparação entre FS, AG e Teste aleatório para mutantes derivados dos 78 operadores

Fonte: Elaborada pelo autor.

todos os programas. Três programas, *countPositive*, *DigitReverse* e *twoPred* atingiram o mesmo escore médio (79,58%; 88,30% e 76,13%, respectivamente) para o AG e geração utilizando teste aleatório.

A Tabela 16 apresenta o número de mutantes gerados utilizando os 78 operadores (segunda coluna), mutantes equivalentes (terceira colunas), número de mutantes mortos, número de dados de teste produzidos e o escore de mutação obtidos pelo FS, AG e testes aleatórios por programa. Os resultados obtidos seguem o mesmo padrão em relação aos resultados com os operadores L-R, abordagem proposta atingiu em média 18,25% do escore de mutação maior que AG e 35,93% maior que a geração aleatória.

É possível concluir com essa análise da QP_4 , que as funções de *fitness* desempenham

Tabela 16 – Adequação dos dados de teste gerados para mutantes L-R (FS, AG e Teste Aleatório) e aplicados nos mutantes derivados de 78 operadores

Programas	Num. de Mut.	Equiv.	FS			AG			Teste Aleatório		
			Mortos	DT	EM	Mortos	DT	EM	Mortos	DT	EM
Cal	891	67	748	7	90,78	645	7	78,28	538	11	65,29
checkIt	104	3	73	6	72,28	62	8	61,39	36	9	35,64
countPositive	151	9	118	3	83,10	113	4	79,58	113	7	79,58
DigitReverser	496	43	424	3	93,60	400	4	88,30	400	10	88,30
findLast	198	17	148	6	81,77	137	8	75,69	68	7	37,57
findVal	190	18	153	7	88,95	119	7	69,19	71	7	41,28
InversePermutation	576	71	464	6	91,88	298	8	59,01	129	8	25,54
Jday-jdate	2821	81	2390	4	87,23	2103	5	76,75	781	10	28,50
lastZero	173	9	149	5	90,85	100	5	60,98	80	6	48,78
NumZero	151	17	122	5	91,04	102	5	76,12	65	6	48,51
oddOrPos	361	71	229	3	78,97	208	6	71,72	203	5	70,00
power	268	12	241	5	94,14	167	4	65,23	92	7	35,94
printPrimes	715	64	642	7	98,62	235	10	36,10	162	15	24,88
quicksort	1026	82	881	3	93,33	719	5	76,17	574	9	60,81
RecursiveSort	555	45	466	4	91,37	227	4	44,51	217	8	42,55
stats	884	101	708	6	90,42	686	4	87,61	618	6	78,93
sum	165	11	143	4	92,86	140	3	90,91	130	5	84,42
testPad	629	57	520	11	90,91	411	12	71,85	211	11	36,89
twoPred	246	24	192	10	86,49	169	11	76,13	169	12	76,13
Total	10.600	802	8.819	105	1.692,17	7.041	120	1.345,52	4.657	159	1.009,54
Média	557,89	42,21	464,15	5,52	89,06	370,58	6,31	70,81	245,10	8,36	53,13

Fonte: Elaborada pelo autor.

um papel muito importante para as técnicas de busca, sobretudo para o problema de geração de dados de testes, uma vez que o espaço de busca, em geral, pode ser enorme. A função de *fitness* guia a busca para regiões promissoras no espaço de busca. Apesar do AG ser conhecido como uma técnica robusta, ainda é necessário projetar função que seja representativa para o problema abordado.

Para confirmar os resultados obtidos na questão QP_4 é importante realizar uma análise entre as abordagens FS_3 , AG e TA para verificar a eficácia na geração e melhoria dos dados de teste para matar os mutantes. Para essa análise, conforme mencionado anteriormente, o *Kruskal-Wallis test* foi aplicado para verificar os escores de mutação alcançados por cada abordagem.

De acordo com os resultados obtidos, o valor de significância é inferior à margem de erro ($H(2) = 25,970017$ e $0,000002 < 0,05$), que corresponde ao nível de confiança de 95%. Nesse contexto, a H_{40} para QP_4 , que alega que os dados de teste gerados e melhorados pelas abordagens (FS_3 , AG e TA) alcançam escore de mutação similares, pode ser estatisticamente rejeitada. Portanto, existem evidências suficientes que confirmam a hipótese alternativa (H_{41}) atestando que existe uma diferença estatisticamente significativa entre os escores de mutação devido gerar dados de teste mais adequados para matar mutantes.

Como o *Kruskal-Wallis test* informa somente se existe ou não uma diferença global significativa entre as amostras, logo não é possível ter conhecimento sobre como as amostras diferem entre si. Portanto, com a finalidade de identificar essas diferenças foi aplicado o *Post Hoc Tests*. Os *Post Hoc Tests* são realizados com objetivo de identificar onde as diferenças ocorrem entre as amostras. Nesse contexto, o teste *Mann-Whitney U* foi aplicado para verificar a diferença entre os tratamentos (MANN; WHITNEY, 1947). Na Tabela 17 mostram os resultados da aplicação desse teste com um intervalo de 95% de confiança.

Tabela 17 – Resultados do teste *Mann-Whitney U* relacionados ao escore de mutação obtidos por FS_3 , AG e TA

Teste Mann-Whitney U		
Comparação das abordagens	U	p-value
FS_3 - AG	56,50	0,000293
FS_3 - TA	28,50	0,000009
AG - TA	90,50	0,008590

Fonte: Elaborada pelo autor.

De acordo com a análise dos resultados de ambas as comparações (FS_3 versus AG e FS_3 versus TA), foi possível confirmar que FS_3 alcançou o melhor escore de mutação. Além disso, a partir destes dados, também pode-se concluir que FS_3 obteve um escore de mutação estatisticamente maior do que AG e TA ($U = 56,50$, $p = 0,000293$, $U = 28,50$ e $p = 0,000009$). Por outro lado, não foi possível identificar uma diferença estatisticamente significativa entre AG e TA.

- **Eficiência da abordagem (QP_5 , QP_6):**

Visando responder às questões de pesquisas 5 e 6 foi realizada uma análise em relação ao tempo necessário para geração dos dados de teste utilizando o esquema de *fitness* considerando o tempo de RD, MD e ID (Tabela 18). Além disso, também foi realizado um estudo comparativo entre o esquema de *fitness* proposto, AG e teste aleatório em relação a quantidade de dados de teste que foi necessário para alcançar os escores de mutação apresentados na Figura 28.

O tempo foi mensurado utilizando as três versões do esquema de *fitness* FS_1 , FS_2 e FS_3 . A Tabela 18 apresenta os programas na primeira coluna e nas colunas seguintes o tempo para cada versão da *FS*. É possível observar que existe uma notável diferença entre os tempos obtidos, principalmente o tempo referente a FS_3 . Isso ocorre devido ao fato de que FS_1 e FS_3 precisam satisfazer mais requisitos para melhorar o impacto do que FS_2 .

Ainda avaliando a eficiência da abordagem proposta, foi analisada a quantidade de dados de testes gerados, comparado com AG e teste aleatório. A Figura 30 apresenta o número de dados de teste no eixo y e os programas utilizados no eixo x.

Com a análise da Figura 30, é importante ressaltar que o esforço para executar e matar um mutante é diretamente proporcional à quantidade de dados do teste. Pode-se observar que para alcançar o escore obtido, a abordagem proposta precisou de em média 0,79% de dados de teste a menos do que AG e 2,84% menos do que a geração aleatória. Embora a diferença seja pequena, o esforço para se gerar ou melhorar apenas um dado de teste adicional para tentar satisfazer um requisito específico pode ser trabalhoso e levar muito tempo.

Além disso, durante o processo de geração de dados com a abordagem proposta, notou-se que à medida que uma nova parte da função de *fitness* é executada, mais dados de testes são

Tabela 18 – Tempo de geração para cada esquema de *fitness*

Programas	Tempo		
	FS1	FS2	FS3
Cal	496,8	491,2	1101,9
checkIt	6,60	5,3	52,5
countPositive	60,9	58,8	114,3
DigitReverser	265,3	257,3	757,0
findLast	115,8	100,2	518,2
findVal	2615,8	2597,9	5891,8
InversePermutation	20,2	17,6	52,6
Jday-jdate	14,2	11,5	98,3
lastZero	57,2	52,4	117,4
NumZero	87,5	81,1	368,4
oddOrPos	217,4	210	620,0
power	80,9	76,2	500,0
printPrimes	30,3	25,7	63,3
quicksort	29,5	26,2	292,6
RecursiveSort	25,7	20,2	53,6
stats	26,6	21	99,3
sum	20,7	16,2	80,9
testPad	27,3	22,3	305,7
twoPred	12,2	9,3	58,3
Total	4.210,9	4.101,0	11.149,7
Média	221,6	215,8	586,8

Fonte: Elaborada pelo autor.

necessários para matar os mutantes remanescentes e mais esforços são aplicados para melhorar os dados. Isso ocorre porque, conforme os mutantes vão sendo mortos, apenas aqueles mais difíceis de matar e equivalentes permanecem no processo e, conseqüentemente, existe a necessidade de dados de teste que satisfaçam os requisitos mais complexos.

Por fim, na questão QP_6 foi verificada a eficiência das abordagens (FS_3 , AG e TA) em relação à quantidade de dados de teste gerados para matar os mutantes, na qual foi aplicada também o teste *Kruskal-Wallis test*. A partir dos resultados do teste estatístico foi possível constatar que o valor de significância é inferior a margem de erro ($H(2) = 11,827296$ e $0,002702 < 0,05$), sugerindo que existe uma diferença significativa entre a quantidade de dados de teste gerados pelo FS_3 , AG e TA, assim, rejeitando a hipótese H_{60} . Para verificar como as amostras diferem entre si, o teste *Mann-Whitney U* também foi aplicado e os resultados são apresentados na Tabela 19.

Ao analisar a Tabela 19 é possível constatar que FS_3 gera uma quantidade de dados de teste estatisticamente diferente em relação ao TA ($U = 68, p = 0,000924$). Além disso, essa diferença não foi identificada para as demais comparações ($FS_3 - AG$ e $AG - TA$) uma vez que ambos *p-values* são inferiores à margem de erro (0,05%), complementado pelo nível de

Figura 30 – Número de dados de teste para FS, AG e Teste aleatório para os mutantes L-R

Fonte: Elaborada pelo autor.

Tabela 19 – Resultados do teste *Mann-Whitney U* relacionados à quantidade de dados de teste gerados por FS_3 , AG e TA

Teste Mann-Whitney U		
Abordagens comparadas	U	p-value
FS_3 - AG	148.50	0.344688
FS_3 - TA	68	0.000924
AG - TA	99	0.016554

Fonte: Elaborada pelo autor.

significância de 95%.

- **Eficácia do método na sugestão de mutantes equivalentes (QP_7):**

A segunda parte da abordagem proposta é referente a um método para auxiliar a detecção de mutantes equivalentes durante o processo de geração de dados de teste. Para avaliar o método proposto foi realizado um experimento a fim de verificar a qualidade das sugestões fornecidas em relação aos mutantes equivalentes e mutantes difíceis de serem mortos.

Assim, para responder a questão de pesquisa QP_7 , considerou-se a não-satisfação das condições RIP, alcançabilidade, infecção e de propagação em cada mutante remanescente isto é, aqueles mutantes que não foram mortos durante o processo de geração de dados de teste. Para isso, foi utilizado o conjunto de mutantes lógicos e relacionais, conforme apresentado na Tabela 11. Estas condições revelaram-se bastante poderosas na detecção e sugestão de mutantes equivalentes, principalmente quando empregado no processo de geração de dados de teste para conjuntos menores de mutantes.

A Figura 31 apresenta a porcentagem média de sugestões verdadeiras para detectar mutantes equivalentes (TS_{eq}) e difíceis de matar (TS_{hk}). O eixo x representa os diferentes

programas utilizados. O eixo y representa a média de sugestões verdadeiras obtida por cada programa em 10 execuções. podendo variar de 0 à 100. Para todos os 19 programas, em média, 76,09% das sugestões foram realizadas corretamente, enquanto que em três programas (*oddOrPos*, *quicksort* e *twoPred*) as sugestões verdadeiras foram menos de 60%.

Figura 31 – Sugestões verdadeiras para mutantes equivalentes e difíceis de serem mortos

Fonte: Elaborada pelo autor.

Para fornecer mais detalhes sobre os resultados obtidos, algumas informações sobre o experimento estão sumarizadas na Tabela 20. Esta tabela exhibe os mutantes remanescentes (coluna RM), as quatro premissas utilizadas para sugerir um mutante como equivalente (colunas A_1, A_2, A_3, A_4), total de sugestões verdadeiras para detectar mutantes equivalentes (coluna TS_{eq}), total de sugestões verdadeiras para mutantes difíceis de matar (coluna TS_{hk}), total de sugestões falsas para detectar mutantes equivalentes (coluna FS_{eq}), total de falsas sugestões para revelar mutantes difíceis de matar (coluna FS_{hk}) e a porcentagem total de sugestões verdadeiras para ambos os casos (coluna %TS).

Os resultados indicam uma forte correlação entre mutantes equivalentes e o não cumprimento das condições de infecção e propagação (A_2) e baixo impacto (A_4). Especificamente, 59,79% das sugestões verdadeiras são os equivalentes mais fáceis de serem identificados, uma vez que eles sempre não conseguem infectar e propagar o estado ou possuem baixo impacto. No entanto, também foi observado que alguns mutantes infectam o estado e em alguns casos eles têm um alto impacto e ainda assim são mutantes equivalentes, o que os tornam quase impossíveis de serem sugeridos como equivalente pelo método proposto.

O maior problema na identificação de mutantes equivalentes é distinguir o equivalente e aqueles mutantes difíceis de matar. A partir dos resultados, foi possível observar que geralmente existem mais mutantes difíceis de matar do que mutantes equivalentes, e por isso é comum um testador ou um método cometer um engano selecionando um mutante difícil de matar e determinando-o como equivalente. No presente experimento, o método proposto sugeriu equivocadamente 16,26% de todos os mutantes remanescentes como equivalente. Também

Tabela 20 – Sugestão de mutantes equivalentes baseada nas condições RIP

Programas	RM	A_1	A_2	A_3	A_4		TS_{eq}	TS_{hk}	FS_{eq}	FS_{hk}	%TS
					Baixo Impacto	Alto Impacto					
Cal	28	2	8	6	3	6	19	6	3	0	89,29
checkIt	13	0	2	1	0	7	3	7	3	0	76,92
countPositive	7	1	0	0	2	4	3	4	0	0	100,00
DigitReverser	17	3	2	2	3	1	10	1	0	6	64,71
findLast	16	0	1	0	0	12	1	12	3	0	81,25
findVal	8	0	0	0	2	4	2	4	1	1	75,00
InversePermutation	23	2	3	2	2	5	9	5	0	9	60,87
Jday-jdate	3	0	0	0	0	3	0	3	0	0	100,00
lastZero	5	0	0	1	1	2	2	2	0	1	80,00
NumZero	6	1	0	1	0	3	2	3	0	1	83,33
oddOrPos	42	5	3	2	5	6	16	6	8	12	52,38
power	4	0	1	0	0	3	1	3	0	0	100,00
printPrimes	14	2	2	0	2	3	6	3	1	4	64,29
quicksort	44	1	2	1	1	20	5	20	15	4	56,82
RecursiveSort	6	0	0	0	2	3	2	3	1	0	83,33
stats	12	2	2	0	3	2	7	2	1	2	75,00
sum	5	0	1	1	1	1	3	1	0	1	80,00
testPad	17	2	1	0	1	7	4	7	3	3	64,71
twoPred	19	1	0	0	1	9	2	9	8	0	57,89
Total	289	22	29	17	29	101	97	101	47	44	1445,79
Média	15,21	1,20	1,53	0,90	1,52	5,31	5,10	5,31	2,47	2,31	76,09

Fonte: Elaborada pelo autor.

notou-se que à medida que o número de mutantes difíceis de matar aumenta, mais enganos são cometidos, uma vez que é muito difícil encontrar um dado de teste para cumprir as condições RIP para esses tipos de mutantes.

Por definição um mutante que não pode ter sua mutação alcançada por qualquer dado de teste, também não pode infectar e também não pode propagar a infecção. Estudos anteriores têm demonstrado que um alto impacto está correlacionado com a não-equivalência e, consequentemente, o oposto com a equivalência. Portanto, se diferentes dados de teste não cumprem a condição de propagação, isso significa que o mutante não tem impacto e tem alta probabilidade de ser equivalente. No método proposto, em primeiro lugar, analisa-se se as condições RIP falharam considerando as três primeiras premissas apresentadas e, em seguida, é analisado o impacto de dados de teste sobre os mutantes.

Os resultados para o total de mutantes equivalentes mostram que 70,11% deles não tiveram impacto (premissas A_1 , A_2 e A_3) devido ao fato de não satisfazerem a condição de propagação, enquanto que 29,89% deles tiveram baixo impacto. No entanto, analisando o total de mutantes remanescentes, observou-se que ainda existem mutantes muito difíceis de matar, uma vez que apenas alguns dados de teste são capazes de impactá-los, e eles podem ser facilmente sugeridos como um mutante equivalente. Os resultados mostram que a premissa A_4 é mais propícia em sugerir falsos mutantes equivalentes devido às medidas de impacto serem fortemente dependentes da qualidade dos dados de teste. Mesmo com avanços significativos na geração de dados de teste automatizada é difícil gerar 100% de dados de teste adequados com um custo-benefício razoável. Assim, pode-se concluir que o número de sugestões falsas é diretamente proporcional ao número de mutantes muito difíceis de matar.

A Tabela 21 apresenta a sugestão de de mutantes equivalentes utilizando as métricas de precisão e revocação. As colunas “Equivalentes” e “Difícil de Matar” correspondem aos conjuntos de mutantes equivalentes e difíceis de matar, respectivamente. Suas sub-colunas apresentam os detalhes sobre o número de mutantes sugeridos corretamente (TS) e incorretamente (FS). Os valores médios de precisão e de revocação do método são apresentados nas duas últimas colunas, respectivamente. É importante destacar que o programa “Jday-jdate” não foi incluído nesta análise, pois não possui mutantes equivalentes.

Tabela 21 – Sugestão de mutantes equivalentes utilizando as métricas de precisão e revocação

Programas	Equivalentes		Difícil de Matar		Precisão	Revocação
	TS_{eq}	FS_{eq}	TS_{hk}	FS_{hk}		
Cal	19	3	6	0	90%	100%
checkIt	3	3	7	0	50%	100 %
countPositive	3	0	4	0	100%	100%
DigitReverser	10	0	1	6	100%	62%
findLast	1	3	12	0	25%	100%
findVal	2	1	4	1	67%	67%
InversePermutation	9	0	5	9	100%	50%
lastZero	2	0	2	1	100%	67%
NumZero	2	0	3	1	100%	67%
oddOrPos	16	8	6	12	67%	57%
power	1	0	3	0	100%	100%
printPrimes	6	1	3	4	86%	60%
quicksort	5	15	20	4	25%	55%
RecursiveSort	2	1	3	0	67%	100%
stats	7	1	2	2	87%	78%
sum	3	0	1	1	100%	75%
testPad	4	3	7	3	57%	57%
twoPred	2	8	9	0	20%	100%
Total	97	47	98	44	75,05%	77,50%

Fonte: Elaborada pelo autor.

Conforme pode ser observado na Tabela 21, o método proposto obteve uma precisão de 75,05% atestando que a maioria dos mutantes foram sugeridos corretamente como equivalentes. Similarmente, a revocação alcançada pelo método (77,50%) indica que a partir do conjunto de mutantes sugeridos corretamente como equivalentes são de fato equivalentes.

Os resultados podem ser considerados satisfatórios, uma vez que o método proposto sugere aqueles mutantes potencialmente equivalentes, podendo reduzir significativamente o número de mutantes que precisam ser analisados. Para esse experimento, considerando os 19 programas, foi possível identificar mais da metade desses mutantes uma vez que dos 141 mutantes equivalentes, 97 foram sugeridos corretamente.

Nesse contexto, a partir dos resultados, fica evidente que o método pode ser utilizado como base para removê-los mesmo sem uma análise. Portanto, é possível concluir que o método

proposto é capaz de reduzir consideravelmente o esforço relacionado à identificação manual de mutantes equivalentes e, mais importante, sem mudanças significativas no escore de mutação.

5.5 Ameaças à Validade

Várias ameaças à validade dos resultados do presente estudo podem ser identificadas. Nesta seção é apresentada uma visão geral das possíveis ameaças que podem comprometer os resultados do experimento.

- **Validade externa:** está relacionada à generalização dos resultados obtidos no experimento. Um problema muito comum identificado é a representatividade dos programas. Para mitigar essa ameaça, 19 programas amplamente utilizados, de diferentes domínios e com diversos tamanhos, foram selecionados como sujeitos experimentais. Outro aspecto a ser destacado que não é possível prever como o esquema de *fitness* proposto se comportaria quando utilizado com outras técnicas de busca, uma vez que o objetivo da avaliação é demonstrar a qualidade e a importância de uma função de *fitness* adequada para o problema de geração de dados de teste. Nesse contexto, replicações futuras são necessárias para corroborar os resultados obtidos.
- **Validade por construção:** aborda a relação entre a teoria e o que é observado. Uma questão é o número de avaliações de *fitness* pois um valor baixo pode influenciar na qualidade dos dados do teste. Em geral, técnicas baseadas em busca com poucas avaliações de *fitness* podem ocasionar um risco de convergência prematura ou impedimentos na busca para fazer progressos substanciais. Isso significa que em alguns casos o conjunto de teste adequado não pode sequer ser encontrado. Em contrapartida, um elevado número de avaliações de *fitness* pode acarretar um tempo elevado e altos custos computacionais para a abordagem.
- **Validade interna:** está relacionada ao grau de confiabilidade entre os resultados esperados e os resultados obtidos. A aleatoriedade das técnicas de busca pode comprometer a precisão dos resultados, pois comumente o espaço de busca para o problema de geração de dados de teste costuma ser muito grande mesmo para um problema pequeno. Devido as técnicas de busca utilizarem heurísticas em sua estrutura não existe garantia da técnica em encontrar sempre a melhor solução. Para mitigar esse risco, o experimento foi executado de forma independente dez vezes por cada abordagem (FS, AG e teste aleatório). Além disso, para assegurar que a confiabilidade dos resultados apresentados por meio de tabelas e gráficos sejam realmente coerentes e interpretados de forma apropriada foram aplicados teste estatísticos.

5.6 Considerações Finais

Neste Capítulo foi apresentado um estudo experimental para investigar a eficácia e a eficiência da abordagem proposta. Primeiramente, foi realizado um experimento visando verificar a contribuição individual de cada parte da função de *fitness* e qual medida de impacto (ID) é mais adequada para compor o esquema proposto. Além disso, foi apresentado também um estudo comparativo entre a abordagem RIP-MuT utilizando o melhor esquema de *fitness*, um algoritmo genético e geração de dados por meio de teste aleatório. E por fim, o segundo módulo da abordagem RIP-MuT referente a sugestão de mutantes equivalentes foi avaliada.

Dezenove programas desenvolvidos em C foram utilizados nos experimentos e a partir deles os mutantes foram produzidos. Dois conjuntos de mutantes foram empregados nos experimentos, o primeiro foi proveniente dos operadores lógicos e relacionais e o segundo foi gerado por 78 operadores de mutação implementados na PROTEUM. Após a obtenção dos mutantes, os dados de testes foram gerados por meio do AVM e o esquema de *fitness* proposto.

De acordo com os resultados obtidos pode-se confirmar que as medidas no esquema de *fitness* funcionam de maneira complementar e não podem obter melhores resultados individualmente. Adicionalmente, os resultados comprovaram que independente de como uma medida de impacto é calculada, aqueles mutantes com maior valor de impacto são mais susceptíveis de serem mortos.

Neste estudo, foram utilizadas três medidas de impacto. Apesar de não obterem resultados muito distintos, a medida ID_3 foi aquela que alcançou melhor escore de mutação em conjunto com RD e MD. Por outro lado, é a medida que requer mais tempo para melhorar os dados de teste e satisfazer seus requisitos. Assim, o melhor esquema de *fitness* (FS_3) foi capaz de alcançar cerca de 86,02% de escore de mutação para mutantes L-R e em média e 89,06% para todos operadores.

Com o estudo comparativo, foi possível observar que o AVM, quando aplicado como uma boa função de *fitness*, pode ser tão poderoso quanto um algoritmo genético. Portanto, o resultados reforçam a necessidade de uma função de *fitness* que seja representativa para o problema. O escore de mutação atingido pelo FS foi 18,25% maior do que AG e 35,93% maior do que o teste aleatório.

O segundo módulo da abordagem RIP-MuT é referente a um método para auxiliar a detecção de mutantes equivalentes durante o processo de geração de dados de teste. Para todos os 19 programas, em média, 76,09% das sugestões foram realizadas corretamente. Além disso, com a análise de impacto nos mutantes equivalentes, os resultados mostram que 70,11% deles não tiveram impacto devido ao fato de não satisfazerem a condição de propagação, enquanto que 29,89% deles tiveram baixo impacto.

CONSIDERAÇÕES FINAIS

Neste projeto de doutorado foi abordado como problema de pesquisa a geração de dados de teste para o teste de mutação. A geração de dados de teste é uma questão amplamente estudada, pois, trata-se de um problema crítico e indecível para todas as técnicas de testes. Além disso, somado a esta questão, a existência de mutantes equivalentes em teste de mutação é uma barreira que ocasiona uma falsa fraqueza nos conjuntos de testes. Assim, nesta tese, almejou-se propor uma abordagem automatizada para geração de dados de testes para o teste de mutação e um método para detecção e sugestão de mutantes equivalentes.

Portanto, os objetivos deste projeto podem ser sumarizados em dois itens principais: 1) produzir dados de teste de qualidade visando alcançar um alto escore de mutação e, 2) identificar mutantes equivalentes no processo de geração de dados visando reduzir os efeitos destes mutantes durante o processo de teste de mutação.

A abordagem proposta emprega um algoritmo Subida da Encosta conhecido como *AVM*, o qual é guiado por um esquema de *fitness* para gerar conjuntos de dados de teste baseados em mutação fraca, firme e forte. Inspirado em estudos anteriores, foi definido um esquema de função *fitness* que gera e melhora dados de teste, medindo o quão próximo os dados de teste candidatos estão de matar um mutante alvo.

Esta abordagem baseia-se principalmente em duas observações. Primeiramente, as técnicas de busca com uma função de *fitness* adequada são benéficas para gerar dados de teste mesmo sem nenhum conhecimento prévio do programa em teste. Em segundo lugar, um método para auxiliar na identificação dos mutantes equivalentes pode reduzir os esforços para gerar dados de teste e superar a fraqueza do teste de mutação.

A ideia da abordagem é produzir conjuntos de dados de teste usando um esquema de funções de *fitness* e no mesmo processo tentar identificar mutantes equivalentes. O principal ganho da análise de mutantes equivalentes no processo de geração de dados de teste é a integração de um método para sugeri-los e eliminá-los, desta forma, os esforços com geração podem ser

direcionados apenas nos mutantes que ainda podem ser mortos.

Nessa direção, foi proposto um esquema de função de *fitness* baseado no modelo RIP (referente às condições de alcançabilidade, necessidade e suficiência) e em uma técnica baseada em busca para gerar dados de teste e sugerir mutantes equivalentes. A parte inovadora da abordagem é a sua capacidade para lidar eficazmente com a condição de propagação dos mutantes e um método para auxiliar na detecção de mutantes equivalentes.

Como parte da pesquisa, um estudo experimental foi conduzido com o objetivo de avaliar a eficiência e as vantagens de utilizar a abordagem aqui apresentada. Os resultados mostram que técnica baseada em busca são é uma adequada alternativa para identificar dados de teste, embora, o diferencial seja o quão apropriada é a função de *fitness* para orientar a busca. O modelo RIP representa as três condições para um dado de teste para matar um mutante, isto é, se um dado de teste satisfaz estas condições, ele pode matar um determinado mutante. Os resultados obtidos indicam que o uso das condições RIP no esquema de *fitness* produzem dados de teste eficazes, que são capazes de matar fortemente a maioria dos mutantes em programas C, e também podem auxiliar na identificação de mutantes equivalentes e a aqueles que são apenas difíceis de serem mortos.

A fim de lidar com as particularidades de C, especialmente ponteiros e estruturas dinâmicas, foi desenvolvida a ferramenta AUSTIN-RIP. Além disso, a capacidade do teste de mutação de mapear todas as estruturas de uma dada linguagem torna possível gerar dados de teste que são capazes de atingir essas particularidades e assim identificar defeitos relacionadas a elas.

É importante destacar que, nos últimos 25 anos, um grande esforço tem sido dedicado à investigação de geração de dados de teste para o teste de mutação. DeMillo e Offutt (1991) foram pioneiros nessa área ao propor uma abordagem na qual as condições de alcançabilidade, infecção e propagação devem ser satisfeitas para matar um determinado mutante. Harman, Jia e Langdon (2011) apresentaram uma abordagem híbrida para geração de dados de teste chamada *Strong Higher Order Mutation (SHOM)* baseada no modelo RIP a fim de matar fortemente mutantes de primeira ordem e de ordem superior para programas em C.

Os resultados sugerem que o esquema de *fitness* (FS) da abordagem RIP-MuT para a geração de dados de teste obteve melhores escores de mutação em relação à abordagem proposta previamente, ao AG e ao teste aleatório. Considerando os mutantes gerados a partir de 78 operados, o FS alcançou um escore de mutação de 20,06% superior à abordagem proposta por Harman, Jia e Langdon (2011).

Estudos anteriores propuseram abordagens para geração de dados de teste ignorando a condição de propagação, tais como, Liu *et al.* (2006) e Papadakis e Malevris (2010a) para matar mutantes fracamente em programas em C e Fraser e Zeller (2010) para mutação firme em programas em Java. Assim, para melhor avaliar o esquema de *fitness* proposto, também foi analisada a geração de dados de teste utilizando somente a condição de alcançabilidade

(RD) e a condição de alcançabilidade juntamente com a condição de infecção (RD & MD). O resultado para os 19 programas apresentado nos experimentos indica que o FS alcançou um escore de mutação de 39,62% superior ao RD e de 16,91% em relação ao RD & MD. Portanto, os resultados demonstram que a geração de dados de teste utilizando as três condições é mais eficaz em relação às abordagens que utilizam apenas uma condição (alcançabilidade) e duas condições (alcançabilidade e infecção).

A identificação de mutantes equivalentes pode ser considerada como um problema que efetivamente inibe o uso generalizado do teste de mutação. Apesar de ser uma atividade crítica, poucos trabalhos têm abordado essa questão por se tratar de um problema indecidível. O método proposto para este contexto, se mostrou viável para redução de custos relacionado a esta atividade, pois ele obteve uma precisão de 75,05% para sugerir mutantes equivalentes.

Nas demais seções deste capítulo, são apresentadas as contribuições e limitações da abordagem (Seção 6.1 e 6.2, respectivamente) e por fim, o capítulo se encerra na Seção 6.3, com uma breve discussão sobre os possíveis trabalhos futuros a serem abordados.

6.1 Contribuições

A principal contribuição desta tese é prover uma abordagem automatizada que engloba a geração de dados de teste e a identificação de mutantes equivalentes. Mais especificamente, as contribuições podem ser sumarizadas como:

1. um esquema de *fitness* para auxiliar teste de software baseado em busca para teste de mutação;
2. uma nova abordagem para a condição de propagação por meio de medidas de impacto;
3. um *framework* estendendo ferramentas existentes de teste estrutural e teste de mutação a fim de produzir dados de teste adequados para mutação fraca e forte;
4. uma ferramenta para auxiliar a geração de dados de teste no teste de mutação;
5. um estudo empírico para avaliação da eficiência e custo do esquema de *fitness* utilizado pela abordagem, considerando os seguintes itens:
 - um experimento para avaliar a contribuição individual de cada parte do esquema de *fitness*, isto é, RD, MD e ID;
 - um experimento para avaliar a adequação de dados de teste gerados para mutantes lógicos e relacionas quando aplicados em um conjunto maior.
 - um experimento para avaliar a adequação das diferentes funções de impacto propostas; e

- um experimento comparativo entre algoritmo genético e teste aleatório.
6. um método para detecção e sugestão de mutantes equivalentes baseado no modelo RIP; e
 7. um estudo empírico para avaliação da eficiência da abordagem para identificar mutantes equivalentes, considerando os seguintes itens:
 - um experimento para analisar as sugestões indicadas corretamente; e
 - um experimento para analisar a influência de uma medida de impacto para sugestão de mutantes equivalentes e mutantes difíceis de serem mortos.
 8. colaboração entre os grupos de pesquisa Laboratório de Engenharia de Software (LabES) do Instituto de Ciências Matemáticas e de Computação (ICMC) da Universidade de São Paulo (campus São Carlos/SP) e o grupo de pesquisa de Engenharia de Software do *Interdisciplinary Centre for Security, Reliability and Trust* (SnT) na Universidade de Luxemburgo, devido à excelência e à experiência na área de automatização de teste de software.

O módulo de geração de dados de teste da abordagem RIP-MuT foi baseada no trabalho de [Papadakis e Malevris \(2013\)](#), que propôs uma abordagem baseada em busca para programas em Java. A abordagem aqui apresentada, usa um novo esquema de função *fitness* para matar fortemente mutantes em programas C, visando a propagação da infecção nos mutantes. A nova medida de impacto (ID), ao contrário de estudos anteriores, orienta a busca, medindo a proximidade para um dado de teste impactar cada predicado percorrido, a partir do ponto de mutação para a saída do programa. Portanto, a abordagem proposta estende o estudo realizado por [Papadakis e Malevris \(2013\)](#) uma vez foca em uma nova linguagem, uma nova função de *fitness* (em particular as medidas de impacto) e investiga a contribuição exata de cada parte do esquema, ou seja, RD, MD e ID.

O primeiro estudo a abordar as condições que devem ser satisfeitas para matar um determinado mutante foi introduzido por [DeMillo e Offutt \(1991\)](#). Neste estudo, estas condições referidas como alcançabilidade, necessidade e suficiência tornaram-se fundamentais para abordagens de geração de dados de teste baseadas em mutações.

A principal contribuição do esquema de *fitness* é o tratamento da condição de suficiência, referida como propagação dos mutantes. Geralmente para matar um mutante, os dados de teste devem: (i) *alcançar* a declaração mutada; (ii) *infectar* o estado do programa; e (iii) *propagar* a infecção para a saída do programa. Nesse contexto, a função de *fitness* proposta visa orientar a busca para as três condições acima mencionadas. Cada um delas é tratada por diferentes partes da função *fitness*: (i) por RD; (ii) por MD; (iii) por ID. Portanto, para matar um mutante, a busca precisa executar incrementalmente RD, MD e ID.

A abordagem proposta utiliza uma nova função de *fitness* que guia incrementalmente o processo para gerar dados de teste que alcançam, infectam e matam os mutantes considerados. A diferença da nova função de *fitness* em relação aos trabalhos anteriores é que ela visa incrementalmente matar os mutantes focando principalmente na propagação, ao invés da infecção nos mutantes. Assim, o esquema de *fitness* proposto mede quão perto estão os dados de teste em ocasionar uma divergência em uma instrução executada em relação ao mutante e ao programa original. Estudos anteriores como [Fraser e Zeller \(2010\)](#), [Harman, Jia e Langdon \(2011\)](#) e [Fraser e Arcuri \(2014\)](#) medem o número de predicados que divergem e não a proximidade para fazer os predicados divergirem, como é abordado na presente tese.

O segundo módulo da RIP-MuT, referente ao método para auxiliar a detecção de mutantes equivalentes foi motivado pelos estudos [Grun, Schuler e Zeller \(2009\)](#) e [Papadakis, Delamaro e Traon \(2014\)](#). O estudo de [Grun, Schuler e Zeller \(2009\)](#) realiza uma análise do impacto na execução dos mutantes e mostra que quanto mais uma mutação altera a execução do programa, maior é a chance do mutante ser não-equivalente. O trabalho de [Papadakis, Delamaro e Traon \(2014\)](#) conduz uma investigação para avaliar a eficiência de uma estratégia de classificação de mutantes baseada em medidas de impacto. A principal diferença do método para identificação de mutantes equivalentes proposto nesta tese, é a análise das condições RIP e o cálculo da medida de impacto, diferentemente dos trabalhos anteriores que analisam somente o impacto. A ideia de incluir o método no processo de geração de dados de teste reduz o número de mutantes analisados, ou seja, é considerado apenas um conjunto menor de mutantes vivos para o método analisar, estudos anteriores analisam os mutantes equivalentes no início do processo do teste de mutação.

Conclusivamente, uma técnica baseada em busca é uma excelente alternativa para identificar dados de teste. No entanto, requer uma função de *fitness* apropriada para guiar a busca em diferentes pontos no espaço de busca como foi focado neste estudo. Uma função de *fitness* adequada pode conduzir às maiores oportunidades para criar conjuntos de dados de teste adequados garantindo um bom escore de mutação e a redução de custos para testes de mutação. Além disso, o modelo RIP indica condições que devem ser satisfeitas para um dado de teste matar um mutante e, de maneira implícita, sugere também as condições para um mutante ser equivalente. É nesta última observação que o método proposto para sugestão de mutantes equivalentes foi fundamentado.

6.2 Limitações

Pela própria natureza da atividade de geração de dados de testes e da linguagem de programação C, algumas limitações podem ser identificadas na abordagem, principalmente pela complexidade da atividade e estruturas dinâmicas da linguagem, como matrizes e ponteiros. Assim, este projeto possui as seguintes limitações:

1. a abordagem para geração de dados utiliza como base o critério de cobertura de ramos para manipular os mutantes. Contudo, só é possível gerar dados para mutações em nós de ramificação (isto é, *if*, *for*, *while*), pois a medida de *branch distance* calcula as proximidades somente para os predicados dos nós de ramificação.
2. a ferramenta AUSTIN manipula somente matrizes com tamanho fixo e não gera entradas muito úteis para *strings*, ponteiros do tipo *void*, bem como construções de união.

6.3 Trabalhos Futuros

Apesar dos resultados encorajadores, ainda há muitos desafios para tornar mais robustas as ferramentas de geração de dados de teste e de teste de mutação para que sejam utilizadas em projetos industriais. A geração de dados de teste e a identificação de mutantes equivalentes são considerados problemas indecidíveis, portanto, não há uma solução 100% adequada para todos os programas. Nesse sentido, um trabalho futuro envolve realizar experimentos adicionais para revalidar estatisticamente os resultados do experimento. Além disso, a condução de experimentos com programas maiores a fim de estabelecer a escalabilidade e aplicabilidade da abordagem em projetos reais.

Outro trabalho futuro seria estender a abordagem para o uso de outros tipos de mutantes além dos lógicos e relacionais. Isso se mostra necessário quando o objetivo do teste está focado em um determinado tipo de defeito. E por fim, um estudo experimental para analisar a influência dos operadores de mutação individualmente na qualidade dos dados de testes, visando investigar principalmente se mutantes com mutações mais complexas e equivalentes podem auxiliar na produção de melhores dados de testes.

REFERÊNCIAS

ADAMOPOULOS, K.; HARMAN, M.; HIERONS, R. M. How to overcome the equivalent mutant problem and achieve tailored selective mutation using co-evolution. In: **Proceedings of the Genetic and Evolutionary Computation Conference**. Seattle, Washington, USA: Springer, 2004. v. 3103, p. 1338–1349. Citado na página 44.

ALSMADI, I. Using genetic algorithms for test case generation and selection optimization. In: **Proceedings of the 26th Annual IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)**. Calgary Alberta, Canada: IEEE Computer Society, 2010. p. 1–4. Citado na página 57.

AMMANN, P.; OFFUTT, J. **Introduction to Software Testing**. 1st. ed. New York, NY, USA: Cambridge University Press, 2008. 3-22 p. ISBN 0521880386, 9780521880381. Citado nas páginas 33, 34 e 38.

ARCURI, A.; IQBAL, M.; BRIAND, L. Random testing: Theoretical results and practical implications. **IEEE Trans. Softw. Eng.**, v. 38, n. 2, p. 258–277, 2012. ISSN 0098-5589. Citado na página 35.

AYARI, K.; BOUKTIF, S.; ANTONIOL, G. Automatic mutation test input data generation via ant colony. In: **Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)**. London, England, UK: ACM, 2007. p. 1074–1081. Citado nas páginas 47, 77, 78, 79, 80, 81 e 85.

BALDWIN, D.; SAYWARD, F. **Heuristics for Determining Equivalence of Program Mutations**. [S.l.]: Yale University, 1979. (Department of Computer Science: Research report). Citado na página 44.

BARBOSA, E. F. **Uma Contribuição para a Determinação de um Conjunto Essencial de Operadores de Mutação no Teste de Programas C**. Dissertação (Mestrado) — ICMC-USP, Novembro 1998. Citado nas páginas 43 e 44.

BARBOSA, E. F.; CHAIM, M. L.; VINCENZI, A. M. R.; DELAMARO, M. E.; JINO, M.; MALDONADO, J. C. Introdução ao teste de software. In: DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. (Ed.). 1st. ed. Rio de Janeiro, RJ, Brasil: Elsevier Editora Ltda, 2007. cap. Teste Estrutural, p. 47–74. Citado na página 37.

BASILI, V.; WEISS, D. A methodology for collecting valid software engineering data. **IEEE Transactions on Software Engineering**, v. 10, n. 6, p. 728–738, 1986. Citado na página 110.

BAUDRY, B.; FLRUREY, F.; JEAN-MARC, J.; TRAON, Y. L. From genetic to bacteriological algorithms for mutation-based testing. **Software Testing Verification Reliability**, v. 15, n. 2, p. 73–96, june 2005. ISSN 0960-0833. Citado nas páginas 73 e 74.

BERTOLINO, A. Software testing research: Achievements, challenges, dreams. In: **Proceedings of the Future of Software Engineering (FOSE)**. Washington, DC, USA: IEEE Computer Society, 2007. p. 85–103. Citado na página 32.

- BLONDIN, J. **Particle Swarm Optimization: A Tutorial**. [S.l.], 2009. Citado na página 50.
- BOTTACI, L. A genetic algorithm fitness function for mutation testing. In: **Proceedings of the First International Workshop on Software Engineering using Metaheuristic Innovative Algorithms**. Toronto, Ontario, Canada: IEEE Computer Society, 2001. p. 3–7. Citado nas páginas 28 e 95.
- BOUSQUET, L. du; DELAUNAY, M. Towards mutation analysis for lustre programs. **Electron. Notes Theor. Comput. Sci.**, v. 203, n. 4, p. 35–48, june 2008. ISSN 1571-0661. Citado na página 44.
- BOYER, R. S.; ELSPAS, B.; LEVITT, K. N. Select - a formal system for testing and debugging programs by symbolic execution. In: **Proceedings of the International Conference on Reliable software**. Los Angeles, California, USA: ACM, 1975. p. 234–245. Citado na página 47.
- BRERETON, P.; KITCHENHAM, B. A.; BUDGEN, D.; TURNER, M.; KHALIL, M. Lessons from applying the systematic literature review process within the software engineering domain. **Journal of Systems and Software**, v. 80, n. 4, p. 571–583, apr 2007. ISSN 0164-1212. Citado nas páginas 67 e 81.
- BRIAND, L. C.; LABICHE, Y.; SHOUSHA, M. **Performance Stress Testing of Real-Time Systems Using Genetic Algorithms**. [S.l.], 2004. Citado na página 57.
- BUDD, A. T.; ANGLUIN, D. Two notions of correctness and their relation to testing. **Acta Inf.**, v. 18, p. 31–45, march 1982. Citado na página 44.
- CASTRO, L. N.; TIMMIS, J. **Artificial Immune Systems A New Computational Intelligence Approach**. [S.l.]: 1st ed. Springer, 2002. Citado na página 74.
- CASTRO, L. N.; ZUBEN, F. J. V. The clonal selection algorithm with engineering applications. In: **Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)**. New York, NY, USA: Morgan Kaufmann, 2002. p. 36–37. Citado na página 75.
- CHOI, B. J.; DEMILLO, R.; KRAUSER, E.; MARTIN, R. J.; MATHUR, A.; OFFUTT, A.; PAN, H.; SPAFFORD, E. The mothra tool set (software testing). In: **Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences, Software Track**. [S.l.: s.n.], 1989. v. 2, p. 275–284. Citado na página 75.
- CLARKE, L. A. A system to generate test data and symbolically execute programs. **Journal IEEE Transactions on Software Engineering**, v. 2, n. 3, p. 215–222, sept. 1976. ISSN 0098-5589. Citado na página 47.
- DAHAL, K. P.; REMDE, S.; COWLING, P.; COLLEDGE, N. Improving metaheuristic performance by evolving a variable fitness function. In: **Proceedings of the 8th EvoCOP**. [S.l.]: Springer, 2008. p. 170–181. Citado na página 52.
- DELAMARO, M. E. **Proteum - Um Ambiente de Teste Baseado na Análise de Mutantes**. Tese (Dissertacao de mestrado) — Universidade de Sao Paulo, Outubro 1993. Citado na página 91.
- DELAMARO, M. E.; BARBOSA, E. F.; VINCENZI, A. M. R.; MALDONADO, J. C. Introdução ao teste de software. In: DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. (Ed.). 1st. ed. Rio de Janeiro, RJ, Brasil: Elsevier Editora Ltda, 2007. cap. Teste de Mutação, p. 7–177. Citado nas páginas 35, 40 e 41.

- DELAMARO, M. E.; DENG, L.; DURELLI, V. H. S.; LI, N.; OFFUTT, J. Experimental evaluation of sdl and automatic mutation based test data generation. one-op mutation for c. In: **Proceedings of Seventh International Conference on Software Testing, Verification and Validation (ICST)**. [S.l.: s.n.], 2014. p. 203–212. Citado na página [111](#).
- DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. Introdução ao teste de software. In: DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. (Ed.). 1st. ed. Rio de Janeiro, RJ, Brasil: Elsevier Editora Ltda, 2007. cap. Conceitos Básicos, p. 1–7. Citado nas páginas [32](#) e [35](#).
- DEMILLO, R. A.; LIPTON, R. J.; SAYWARD, F. G. Hints on test data selection: Help for the practicing programmer. **Journal Computer**, v. 11, n. 4, p. 34–41, abril 1978. ISSN 0018-9162. Citado nas páginas [26](#) e [39](#).
- DEMILLO, R. A.; OFFUTT, A. J. Constraint-based automatic test data generation. **IEEE Transactions on Software Engineering**, v. 17, n. 9, p. 900–910, sept. 1991. ISSN 0098-5589. Citado nas páginas [26](#), [41](#), [47](#), [48](#), [74](#), [75](#), [77](#), [78](#), [79](#), [80](#), [81](#), [95](#), [134](#) e [136](#).
- DEMILLO, R. A.; SAYWARD, F. G.; LIPTON, R. J. Program mutation: A new approach to program testing. In: **Infotech International State of the Art Report: Program Testing**. [S.l.: s.n.], 1979. p. 107–126. Citado na página [39](#).
- DIJKSTRA, E. Programming considered as a human activity. In: **Book Classics in software engineering**. Upper Saddle River, NJ, USA: Yourdon Press, 1979. p. 1–9. Citado na página [25](#).
- DOMÍNGUEZ-JIMÉNEZ, J. J.; ESTERO-BOTARO, A.; GARCÍA-DOMÍNGUEZ, A.; MEDINA-BULO, I. Evolutionary mutation testing. **Information and Software Technology**, v. 53, n. 10, p. 1108–1123, october 2011. ISSN 0950-5849. Citado na página [28](#).
- DURAN, J. W.; NTAFOSS, S. C. An evaluation of random testing. **Journal IEEE Transactions on Software Engineering**, v. 10, n. 4, p. 438–444, july 1984. ISSN 0098-5589. Citado na página [47](#).
- EBERHART, R. C.; SHI, Y. Comparing inertia weights and constriction factors in particle swarm optimization. In: **Proceedings of the Congress on Evolutionary Computation (CEC)**. San Diego, California, USA: IEEE Computer Society, 2000. p. 84–88. Citado na página [58](#).
- ELFURJANI, S. M.; BOTTACI, L. Efficiency of mutation operators and selective mutation strategies: An empirical study. **Software Testing, Verification and Reliability**, v. 9, n. 4, p. 205–232, december 1999. ISSN 1099-1689. Citado na página [44](#).
- ELLIMS, M.; INCE, D.; PETRE, M. The csaw c mutation tool: Initial results. In: **Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques (TAICPART-MUTATION)**. Washington, DC, USA: IEEE Computer Society, 2007. p. 185–192. Citado na página [44](#).
- FRASER, G.; ARCURI, A. Achieving scalable mutation-based generation of whole test suites. **Empirical Software Engineering**, v. 20, n. 3, p. 783–812, 2014. Citado nas páginas [27](#), [78](#), [79](#), [91](#), [96](#) e [137](#).
- FRASER, G.; ZELLER, A. Mutation-driven generation of unit tests and oracles. In: **Proceedings of the 19th International Symposium on Software Testing and Analysis (ISSTA)**. Trento, Italy: ACM, 2010. p. 147–157. Citado nas páginas [72](#), [77](#), [78](#), [79](#), [80](#), [81](#), [91](#), [96](#), [134](#) e [137](#).

GHATEI, S.; PANAHI, F. T.; HOSSEINZADEH, M.; ROUHI, M.; REZAZADEH, I.; NAEBI, A.; GHATEI, Z.; KHAJEI, R. P. A new hybrid algorithm for optimization using pso and gda. **Journal of Basic and Applied Scientific Research**, p. 2336–2341, jan. 2012. ISSN 2090-4304. Citado na página 60.

GOLDBERG, D. E. **Genetic Algorithms in Search, Optimization and Machine Learning**. 1st. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989. ISBN 0201157675. Citado na página 73.

GRUN, B. J. M.; SCHULER, D.; ZELLER, A. The impact of equivalent mutants. In: **Proceedings of the 2nd International Conference on Software Testing, Verification, and Validation, Workshops (ICSTW)**. Denver, Colorado, USA: IEEE Computer Society, 2009. p. 192–199. Citado nas páginas 45 e 137.

GUPTA, N.; MATHUR, A. P.; SOFIA, M. L. Automated test data generation using an iterative relaxation method. In: **Proceedings of the 6th International Symposium on Foundations of Software Engineering (SIGSOFT)**. Orlando, FLorida, USA: ACM, 1998. Citado na página 76.

HAGA, H.; SUEHIRO, A. Automatic test case generation based on genetic algorithm and mutation analysis. In: **Proceedings of the International Conference on Control System, Computing and Engineering (ICCSCE)**. Penang, Malaysia: IEEE Computer Society, 2012. p. 119–123. Citado nas páginas 72, 77, 78, 79, 80 e 81.

HAMLET, D.; TAYLOR, R. Partition testing does not inspire confidence (program testing). **Journal IEEE Transactions on Software Engineering**, v. 16, n. 12, p. 1402–1411, december 1990. ISSN 0098-5589. Citado nas páginas 31 e 47.

HAMLET, R. G. Testing programs with the aid of a compiler. **IEEE Transactions on Software Engineering**, SE-3, n. 4, p. 279–290, july 1977. ISSN 0098-5589. Citado na página 39.

HARMAN, M.; BRYAN, F. J. Search-based software engineering. **Information and Software Technology**, v. 43, n. 14, p. 833–839, December 2001. Citado na página 44.

HARMAN, M.; JIA, Y.; LANGDON, W. B. Strong higher order mutation-based test data generation. In: **Symposium on the Foundations of Software Engineering (SIGSOFT)**. Szeged, Hungary: ACM, 2011. p. 212–222. Citado nas páginas 25, 27, 72, 77, 78, 79, 80, 81, 91, 96, 134 e 137.

HARMAN, M.; MCMINN, P. A theoretical & empirical analysis of evolutionary testing and hill climbing for structural test data generation. In: **Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)**. London, United Kingdom: ACM, 2007. p. 73–83. Citado na página 91.

_____. A theoretical and empirical study of search-based testing: Local, global, and hybrid search. **IEEE Transactions on Software Engineering**, v. 36, n. 2, p. 226–247, 2010. Citado nas páginas 92 e 93.

HIERONS, R.; HARMAN, M.; DANICIC, S. Using program slicing to assist in the detection of equivalent mutants. **Software Testing, Verification and Reliability**, v. 9, n. 4, p. 233–262, december 1999. Citado na página 44.

HOLLAND, J. **Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence**. Cambridge, MA, USA: MIT Press, 1992. ISBN 0262082136. Citado na página 55.

HOLLAND, J. H. **Adaptation in Natural and Artificial Systems**. Ann Arbor, MI: University of Michigan Press, 1975. Citado na página 53.

HORST, R.; PARDALOS, P. M.; THOAI, N. V. **Introduction to global optimization**. 2nd. ed. [S.l.]: Springer, 2000. ISBN 0-7923-6574-7. Citado na página 57.

HOWDEN, W. E. Symbolic testing and the dissect symbolic evaluation system. **Journal IEEE Transactions on Software Engineering**, v. 3, n. 4, p. 266–278, July 1977. ISSN 0098-5589. Citado nas páginas 31 e 47.

_____. Weak mutation testing and completeness of test sets. **IEEE Trans. Softw. Eng.**, v. 8, n. 4, p. 371–379, jul. 1982. ISSN 0098-5589. Citado na página 43.

IEEE. **IEEE Standard Glossary of Software Engineering Terminology**. 1990. IEEE Standard. Citado nas páginas 32 e 36.

JENKINS, N. **A Software Testing Primer: An Introduction to Software Testing**. San Francisco, California: Creative Commons, 2008. 43 p. Citado na página 32.

Ji, C.; CHEN, Z.; XU, B.; WANG, Z. A new mutation analysis method for testing java exception handling. In: **Proceedings of the 33rd Annual IEEE International Computer Software and Applications Conference**. Seattle, Washington, USA: IEEE Computer Society, 2009. p. 556–561. Citado na página 44.

Ji, C.; CHEN, Z.; XU, B.; ZHAO, Z. A novel method of mutation clustering based on domain analysis. In: **Proceedings of the 21st International Conference on Software Engineering & Knowledge Engineering (SEKE)**. Boston, Massachusetts, USA: Knowledge Systems Institute Graduate School, 2009. p. 422–425. Citado nas páginas 43 e 44.

JIA, Y.; HARMAN, M. Constructing subtle faults using higher order mutation testing. **Source Code Analysis and Manipulation, IEEE International Workshop on**, v. 0, p. 249–258, september 2008. Citado na página 80.

_____. Higher order mutation testing. **Information and Software Technology**, v. 51, n. 10, p. 1379–1393, 2009. Citado na página 44.

_____. An analysis and survey of the development of mutation testing. **IEEE Transactions on Software Engineering**, v. 37, n. 5, p. 649–678, sept. 2011. ISSN 0098-5589. Citado nas páginas 39 e 43.

JUANG, C.-F. A hybrid of genetic algorithm and particle swarm optimization for recurrent network design. **Trans. Sys. Man Cyber. Part B**, v. 34, n. 2, p. 997–1006, april 2004. ISSN 1083-4419. Citado na página 59.

KAMINSKI, G.; MASON, G.; AMMANN, P. Using a fault hierarchy to improve the efficiency of dnf logic mutation testing. In: **Proceedings of the Second International Conference on Software Testing, Verification and Validation (ICST)**. Denver, Colorado: IEEE Computer Society, 2009. p. 386–395. Citado na página 44.

- KENNEDY, J.; EBERHART, R. C. Particle swarm optimization. In: **Proceedings of the IEEE International Conference on Neural Networks**. Perth, Western Australia: IEEE Computer Society, 1995. p. 1942–1948. Citado na página 57.
- KINTIS, M.; PAPADAKIS, M.; MALEVRIS, N. Evaluating mutation testing alternatives: A collateral experiment. In: **Proceedings of the 17th Asia Pacific Software Engineering Conference (ASPEC)**. Sydney, Australia: IEEE Computer Society, 2010. p. 300–309. Citado na página 44.
- KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. Optimization by simulated annealing. **Science**, v. 220, n. 4598, p. 671–680, may 1983. Citado na página 59.
- KITCHENHAM, B.; BUDGEN, D.; BRERETON, P. **Evidence-Based Software Engineering and Systematic Reviews**. [S.l.]: Chapman and Hall/CRC, 2015. 433 p. Citado na página 65.
- KITCHENHAM, B.; PRETORIUS, R.; BUDGEN, D.; BRERETON, P.; TURNER, M.; NIAZI, M.; LINKMAN, S. Systematic literature reviews in software engineering - a tertiary study. **Inf. Softw. Technol.**, Butterworth-Heinemann, Newton, MA, USA, p. 792–805, 2010. Citado nas páginas 65, 66 e 82.
- KOREL, B. Automated software test data generation. **IEEE Transactions on Software Engineering**, v. 16, n. 8, p. 870–879, aug. 1990. ISSN 0098-5589. Citado nas páginas 45, 46, 47, 48, 63, 72, 91 e 94.
- _____. Computation of dynamic program slices for unstructured programs. **Software Engineering, IEEE Transactions on**, v. 23, n. 1, p. 11–33, jan, 1997. ISSN 0098-5589. Citado na página 76.
- KRUSKAL, W. Use of ranks in one-criterion variance analysis. **Journal of the American Statistical Association**, v. 47, n. 260, p. 583–621, 1947. Citado na página 117.
- LAKHOTIA, K.; HARMAN, M.; GROSS, H. Austin: A tool for search based software testing for the c language and its evaluation on deployed automotive systems. In: **Proceedings of the 2th SSBSE**. [S.l.: s.n.], 2010. p. 101–110. Citado na página 91.
- LETICIA, M. P. **Estratégias de Seleção de Caminhos no Contexto de Critérios Estruturais de Teste de Software**. Dissertação (Mestrado) — UNICAMP, 1999. Citado na página 75.
- LIPTON, R. **Fault Diagnosis of Computer Programs**. [S.l.], 1971. Citado na página 39.
- LIU, M.-H.; GAO, Y.-F.; SHAN, J.-H.; LIU, J.-H.; ZHANG, L.; SUN, J.-S. An approach to test data generation for killing multiple mutants. In: **Proceedings of the 22nd IEEE International Conference on Software Maintenance (ICSM)**. Philadelphia, Pennsylvania, USA: IEEE Computer Society, 2006. p. 113–122. Citado nas páginas 76, 78, 79, 80, 81 e 134.
- LOUZADA, J.; CAMILO-JUNIOR, C. G.; VINCENZI, A.; RODRIGUES, C. An elitist evolutionary algorithm for automatically generating test data. In: **Proceedings of the World Congress on Computational Intelligence (WCCI)**. Brisbane, Australia: IEEE Computational Intelligence Society, 2012. p. 1–8. Citado nas páginas 72, 77, 78, 79, 80, 81 e 85.
- M., D.; CARO, G. D. Handbook of metaheuristics. In: GLOVER, F. W.; KOCHENBERGER, G. A. (Ed.). New York, NY, USA: Springer US, 1999. cap. The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances, p. 250–285. Citado na página 72.

- MANN, H. B.; WHITNEY, D. R. On a test of whether one of two random variables is stochastically larger than the other. **Annals of Mathematical Statistics**, v. 18, n. 1, p. 50–60, 1947. Citado na página [123](#).
- MANNING, C. D.; RAGHAVAN, P.; SCHUTZE, H. **Introduction to Information Retrieval**. [S.l.]: Cambridge University Press, 2008. 482 p. Citado na página [109](#).
- MARTIN, E.; XIE, T. A fault model and mutation testing of access control policies. In: **Proceedings of the 16th International Conference on World Wide Web (IW3C2)**. Banff, Alberta, Canada: ACM, 2007. p. 667–676. Citado na página [44](#).
- MASUD, M.; NAYAK, A.; ZAMAN, M.; BANSAL, N. Strategy for mutation testing using genetic algorithms. In: **Proceedings of the Canadian Conference on Electrical and Computer Engineering**. [S.l.: s.n.], 2005. p. 1049–1052. Citado nas páginas [72](#), [77](#), [78](#) e [79](#).
- MATHUR, A.; WONG, W. Reducing the cost of mutation testing: An empirical study. **The Journal of Systems and Software**, v. 31, n. 3, p. 185–196, dec 1995. ISSN 0164-1212. Citado nas páginas [43](#) e [44](#).
- MATHUR, A. P.; WONG, W. E. An empirical comparison of data flow and mutation-based test adequacy criteria. **Software Testing, Verification and Reliability**, v. 4, n. 1, p. 9–31, oct. 1994. Citado na página [43](#).
- MAY, P.; MANDER, K.; TIMMIS, J. Software vaccination: An artificial immune system approach to mutation testing. In: **Proceedings of the Artificial Immune Systems, Second International Conference (ICARIS)**. Edinburgh, UK: Springer, 2003. p. 81–92. Citado na página [74](#).
- MAY, P.; TIMMIS, J.; MANDER, K. Immune and evolutionary approaches to software mutation testing. In: **Proceedings of the 6th International Conference on Artificial Immune Systems (ICARIS)**. Santos, Brazil: Springer-Verlag, 2007. p. 336–347. Citado nas páginas [75](#), [77](#), [78](#), [79](#), [80](#) e [81](#).
- MCGREGOR, J. D.; SYKES, D. A. **A practical guide to testing object-oriented software**. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001. Citado na página [32](#).
- MCMINN, P. Search-based software test data generation: A survey. **Software Testing, Verification & Reliability**, v. 14, n. 2, p. 105–156, june 2004. ISSN 0960-0833. Citado nas páginas [27](#), [28](#) e [48](#).
- _____. Search-based software testing: Past, present and future. In: **Proceedings of the IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)**. Berlin, Germany: IEEE Computer Society, 2011. p. 153–163. Citado na página [52](#).
- MCMINN, P.; HOLCOMBE, M. Evolutionary testing using an extended chaining approach. **Evolutionary Computation**, v. 14, n. 1, p. 41–64, 2006. Citado na página [94](#).
- MICHAEL, C. C.; MCGRAW, G.; SCHATZ, M. Generating software test data by evolution. **IEEE Transactions on Software Engineering**, v. 27, n. 12, p. 1085–1110, december 2001. ISSN 0098-5589. Citado nas páginas [47](#) e [57](#).
- MILLER, W.; SPOONER, D. L. Automatic generation of floating-point test data. **IEEE Trans. Softw. Eng.**, v. 2, n. 3, p. 223–226, maio 1976. ISSN 0098-5589. Citado na página [48](#).

- MISHRA, K. K.; TIWARI, S.; KUMAR, A.; MISRA, A. An approach for mutation testing using elitist genetic algorithm. In: **Proceedings of the International Conference on Computer Science and Information Technology (ICCSIT)**. Chengdu, China: IEEE, 2010. p. 426–429. Citado nas páginas 72, 78, 79 e 85.
- MITCHELL, M. **An Introduction to Genetic Algorithms**. Cambridge, MA, USA: MIT Press, 1998. ISBN 0262631857. Citado nas páginas 54 e 55.
- MYERS, G. J.; SANDLER, C. **The Art of Software Testing**. [S.l.]: John Wiley & Sons, 2004. ISBN 0471469122. Citado nas páginas 25, 31 e 37.
- NAMIN, A. S.; XUE, X.; ROSAS, O.; SHARMA, P. Muranker: A mutant ranking tool. **Softw. Test. Verif. Reliab.**, John Wiley and Sons Ltd., v. 25, n. 5-7, p. 572–604, ago. 2015. ISSN 0960-0833. Citado na página 100.
- NICA, S. On the improvement of the mutation score using distinguishing test cases. In: **International Conference on Software Testing, Verification and Validation Workshops (ICSTW)**. Berlin, Germany: IEEE Computer Society, 2011. p. 423–426. Citado nas páginas 75, 78, 79, 80 e 81.
- OFFUTT, A. J. An integrated system for automatically generating test data. In: **Proceedings of the first international conference on Systems Integration**. Piscataway, NJ, USA: IEEE Press, 1990. p. 694–695. Citado nas páginas 47, 78 e 79.
- _____. Investigations of the software testing coupling effect. **ACM Transactions on Software Engineering and Methodology**, v. 1, n. 1, p. 5–20, jan. 1992. ISSN 1049-331X. Citado nas páginas 39 e 44.
- OFFUTT, A. J.; J., P. Automatically detecting equivalent mutants and infeasible paths. **Software Testing, Verification, and Reliability**, v. 7, n. 3, p. 165–192, september 1997. Citado nas páginas 44 e 45.
- OFFUTT, A. J.; LEE, A.; ROTHERMEL, G.; UNTCH, R. H.; ZAPF, C. An experimental determination of sufficient mutant operators. **ACM Trans. Softw. Eng. Methodol.**, v. 5, n. 2, p. 99–118, april 1996. ISSN 1049-331X. Citado nas páginas 40 e 43.
- OFFUTT, A. J.; PAN, J. Detecting equivalent mutants and the feasible path problem. In: **Proceedings of the Eleventh Annual Conference on Computer Assurance : Systems Integrity, Software Safety, Process Security (COMPASS)**. Gaithersburg, Maryland: IEEE Computer Society Press, 1996. p. 224–236. Citado nas páginas 44 e 45.
- OFFUTT, A. J.; ROTHERMEL, G.; ZAPF, C. An experimental evaluation of selective mutation. In: **Proceedings of the 15th international conference on Software Engineering**. Los Alamitos, CA, USA: IEEE Computer Society Press, 1993. p. 100–107. Citado na página 43.
- OFFUTT, J.; MA, Y.-S.; KWON, Y.-R. The class-level mutants of mujava. In: **Proceedings of the International Workshop on Automation of Software Test (AST)**. Shanghai, China: ACM, 2006. p. 78–84. Citado na página 44.
- OFFUTT, J. A.; CRAFT, M. W. Using compiler optimization techniques to detect equivalent mutants. **IEEE**, v. 4, n. 3, p. 1–26, december 1994. Citado na página 44.
- OFFUTT, J. A.; UNTCH, R. H. **Mutation 2000: Uninting the Orthogonal**. [S.l.]: Testing in the Twentieth and the Twenty First Centuries, 2000. 45-55 p. Citado nas páginas 42 e 45.

ORZESZYNA, W. **Solutions to the equivalent mutants problem: A systematic review and comparative experiment**. Tese (Master Thesis) — School of Computing Blekinge Institute of Technology, June 2011. Citado nas páginas 44 e 45.

PAPADAKIS, M.; DELAMARO, M.; TRAON, Y. L. Mitigating the effects of equivalent mutants with mutant classification strategies. **Sci. Comput. Program.**, Elsevier North-Holland, Inc., v. 95, n. P3, p. 298–319, dez. 2014. Citado nas páginas 27 e 137.

PAPADAKIS, M.; MALEVRIS, N. An effective path selection strategy for mutation testing. In: **Proceedings of the 16th Asia-Pacific Software Engineering Conference (ASPEC)**. Penang, Malaysia: IEEE, 2009. p. 422–429. Citado nas páginas 75, 78, 79, 80 e 81.

_____. Automatic mutation test case generation via dynamic symbolic execution. In: **Proceedings of the 21st International Symposium on Software Reliability Engineering (ISSRE)**. San Jose, CA, USA: IEEE Computer Society, 2010. p. 121–130. Citado nas páginas 78, 79, 80, 81 e 134.

_____. An empirical evaluation of the first and second order mutation testing strategies. In: **Proceedings of the 3rd International Conference on Software Testing, Verification, and Validation Workshops (ICSTW)**. Paris, France: IEEE Computer Society, 2010. p. 90–99. Citado na página 44.

_____. Automatic mutation based test data generation. In: **Proceedings of the 13th annual Conference Companion on Genetic and Evolutionary Computation (GECCO)**. Dublin, Ireland: ACM, 2011. p. 247–248. Citado na página 85.

_____. Mutation based test case generation via a path selection strategy. **Journal Information and Software Technology**, v. 54, n. 9, p. 915–932, september 2012. ISSN 0950-5849. Citado nas páginas 27, 78, 79, 80 e 81.

_____. Searching and generating test inputs for mutation testing. **Journal SpringerPlus**, v. 2, n. 1, p. 1–12, march 2013. ISSN 2193-1801. Citado nas páginas 27, 77, 78, 79, 80, 81, 86, 95 e 136.

PAPADAKIS, M.; MALEVRIS, N.; KALLIA, M. Towards automating the generation of mutation tests. In: **Proceedings of the 5th Workshop on Automation of Software Test (AST)**. Cape Town, South Africa: ACM, 2010. p. 111–118. Citado nas páginas 78, 79, 80 e 81.

PARK, J.-B.; LEE, K.-S.; SHIN, J.-R.; LEE, K. A particle swarm optimization for economic dispatch with nonsmooth cost functions. **IEEE Transactions on Power Systems**, v. 20, n. 1, p. 34–42, feb. 2005. ISSN 0885-8950. Citado na página 58.

PARSOPOULOS, K. E.; VRAHATIS, M. N. Particle swarm optimization method in multiobjective problems. In: **Proceedings of the ACM Symposium on Applied Computing (SAC)**. Madrid, Spain: ACM, 2002. p. 603–607. Citado na página 58.

PEZZE, M.; YOUNG, M. **Teste e Análise de Software: Processos, Princípios e Técnicas**. [S.l.]: Editora: Artmed, 2008. 39-50 p. Citado nas páginas 35 e 38.

PIANKA, E. R. **Evolutionary Ecology**. 6th. ed. [S.l.]: Benjamin-Cummings Publishing Company, 1999. Citado na página 73.

PRESSMAN, R. S. **Software engineering: A practitioner's approach**. 7th. ed. [S.l.]: McGraw-Hill, 2010. 928 p. Citado nas páginas 25, 35 e 36.

- RAD, M.; AKBARI, F.; BAKHT, A. Implementation of common genetic and bacteriological algorithms in optimizing testing data in mutation testing. In: **Proceedings of the International Conference on Computational Intelligence and Software Engineering (CiSE)**. Wuhan, China: IEEE Computer Society, 2010. p. 1–6. Citado nas páginas 72, 77, 78, 79, 80 e 81.
- RAMAMOORTHY, C. V.; HO, S. B. F.; CHEN, W. T. On the automated generation of program test data. **Journal IEEE Transactions on Software Engineering**, SE-2, n. 4, p. 293–300, december 1976. ISSN 0098-5589. Citado na página 47.
- RAYWARD-SMITH, V.; OSMAN, I. H.; REEVES, C. R.; D., S. G. **Modern heuristic search methods**. 1. ed. New York: John Wiley, 1996. ISBN 9780471962809. Citado na página 49.
- RUSSELL, S. J.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. 2. ed. [S.l.]: Pearson Education, 2003. Chapter 4. ISBN 0137903952. Citado nas páginas 57, 61, 62 e 74.
- SCHULER, D.; DALLMEIER, V.; ZELLER, A. Efficient mutation testing by checking invariant violations. In: **Proceedings of the 8th International Symposium on Software Testing and Analysis, (ISSTA)**. Chicago, Illinois, USA: ACM, 2009. p. 69–80. Citado na página 45.
- SCHULER, D.; ZELLER, A. Un-covering equivalent mutants. In: **Proceedings of the 3rd International Conference on Software Testing Verification and Validation (ICST)**. Paris, France: IEEE Computer Society, 2010. p. 45–54. Citado na página 45.
- SEN, K.; MARINOV, D.; AGHA, G. Cute: a concolic unit testing engine for c. In: **Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering**. Lisbon, Portugal: ACM, 2005. p. 263–272. Citado nas páginas 76 e 77.
- SHAN, J. H.; WANG, J.; QI, Z. C.; WU, J. P. Improved method to generate path-wise test data. **Journal of Computer Science and Technology**, v. 18, n. 2, p. 235–240, march 2003. ISSN 1000-9000. Citado na página 76.
- SINGH, K.; R., K. Optimization of functional testing using genetic algorithms. **International Journal of Innovation, Management and Technology**, v. 1, n. 1, p. 43–46, april 2010. ISSN 2010-0248. Citado na página 57.
- SOININEN, T.; GELLE, E. **Dynamic Constraint Satisfaction in Configuration**. [S.l.], 1999. Citado na página 75.
- SOUZA, F. C. M.; PAPADAKIS, M.; TRAON, Y. L.; DELAMARO, M. E. Strong mutation-based test data generation using hill climbing. In: **Proceedings of the 9th International Workshop on Search-Based Software Testing**. [S.l.]: ACM, 2016. (SBST '16), p. 45–54. ISBN 978-1-4503-4166-0. Citado nas páginas 77, 78, 79, 80, 81 e 111.
- SRIDHARAN, M.; NAMIN, A. S. Prioritizing mutation operators based on importance sampling. In: **Proceedings of the IEEE 21st International Symposium on Software Reliability Engineering (ISSRE)**. San Jose, CA, USA: IEEE Computer Society, 2010. p. 378–387. Citado na página 28.
- UNTCH, R. H.; OFFUTT, A. J.; HARROLD, M. J. Mutation analysis using mutant schemata. In: **Proceedings of the 1993 ACM SIGSOFT international symposium on Software testing and analysis (ISSTA)**. Cambridge, Massachusetts, USA: ACM, 1993. p. 139–148. Citado na página 81.

- VERGILIO, S. R.; MALDONADO, J. C.; JINO, M. Introdução ao teste de software. In: DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. (Ed.). 1st. ed. Rio de Janeiro, RJ, Brasil: Elsevier Editora Ltda, 2007. cap. Geração de Dados de Teste, p. 269–291. Citado nas páginas 45 e 48.
- VINCENZI, A. M. R.; NAKAGAWA, E. Y.; MALDONADO, J. C.; DELAMARO, M. E.; ROMERO, R. A. F. Bayesian-learning based guidelines to determine equivalent mutants. **International Journal of Software Engineering and Knowledge Engineering**, v. 12, n. 6, p. 675–690, 2002. ISSN 1793-6403. Citado na página 45.
- WALCOTT, K. R.; SOFFA, M. L.; KAPFHAMMER, G. M.; ROOS, R. S. Time aware test suite prioritization. In: **Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)**. Portland, Maine, USA: ACM, 2006. p. 1–12. Citado na página 57.
- WEGENER, J.; BARESEL, A.; STHAMER, H. Evolutionary test environment for automatic structural testing. **Information and Software Technology**, v. 43, n. 14, p. 841–854, december 2001. ISSN 0950-5849. Citado nas páginas 47 e 93.
- WEISE, T. **Global Optimization Algorithms: Theory and Application**. 2nd. ed. [S.l.]: Self-Published, 2009. Disponível em: <<http://www.it-weise.de/projects/book.pdf>>. Acessado em: 28/08/2013. Citado nas páginas 49, 50 e 55.
- WHITE, L. J.; COHEN, E. I. A domain strategy for computer program testing. **IEEE Trans. Softw. Eng.**, SE-6, n. 3, p. 247–257, may 1980. ISSN 0098-5589. Citado na página 47.
- WOHLIN, C.; RUNESON, P.; HÖST, M. C.; OHLSSON, M. C.; REGNELL, B.; WESSLÉN, A. **Experimentation in Software Engineering**. [S.l.]: Springer, 2012. ISBN 978-3-642-29043-5. Citado na página 111.
- WOODWARD, M. R.; K., H. From weak to strong, dead or alive? an analysis of some mutation testing issues. In: **Software Testing, Verification, and Analysis, 1988., Proceedings of the Second Workshop on**. [S.l.: s.n.], 1988. p. 152–158. Citado na página 43.
- WYHE, J. V. **Charles Darwin: gentleman naturalist**. 2002. Disponível em: <<http://darwin-online.org.uk/darwin.html>>. Acessado em: 22/08/2013. Citado na página 49.
- YAO, X.; HARMAN, M.; JIA, Y. A study of equivalent and stubborn mutation operators using human analysis of equivalence. In: **Proceedings of the 36th International Conference on Software Engineering**. [S.l.]: ACM, 2014. (ICSE 2014), p. 919–930. ISBN 978-1-4503-2756-5. Citado na página 101.
- ZBIGNIEW, M.; FOGEL, D. **How to Solve It: Modern Heuristics**. [S.l.]: Springer; Enlarged 2nd, 2004. Citado na página 49.
- ZHAN, Y.; CLARK, J. A. Search-based mutation testing for simulink models. In: **Proceedings of the Conference on Genetic and Evolutionary Computation (GECCO)**. Washington, D.C, USA: ACM, 2005. p. 1061–1068. Citado na página 28.
- ZHANG, L.; XIE, T.; TILLMANN, N.; HALLEUX, J.; MEI, H. Test generation via dynamic symbolic execution for mutation testing. In: **Proceedings of IEEE International Conference on Software Maintenance (ICSM)**. [S.l.]: IEEE, 2010. p. 1–10. Citado nas páginas 78, 79, 80 e 81.