UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

Uma abordagem para identificação de mutantes minimais e equivalentes baseada na estrutura do código fonte

Claudinei Brito Junior

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

ICMC USP
SÃO CARLOS

Claudinei Brito Junior

Uma abordagem para identificação de mutantes minimais e equivalentes baseada na estrutura do código fonte

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências – Ciências de Computação e Matemática Computacional. *EXEMPLAR DE DEFESA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. Márcio Eduardo Delamaro

USP – São Carlos
Fevereiro de 2022

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

Brito Junior, Claudinei
B862a Uma abordagem para identificação de mutantes minimais e equivalentes baseada na estrutura do código fonte / Claudinei Brito Junior; orientador Márcio Eduardo Delamaro. -- São Carlos, 2022.
115 p.

Dissertação (Mestrado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2022.

1. Teste de Software. 2. Teste de Mutação. 3. Mutantes Minimais. 4. Mutantes Equivalentes. I. Delamaro, Márcio Eduardo, orient. II. Título.

Bibliotecários responsáveis pela estrutura de catalogação da publicação de acordo com a AACR2:
Gláucia Maria Saia Cristianini - CRB - 8/4938
Juliana de Souza Moraes - CRB - 8/6176

Claudinei Brito Junior

An approach to identifying minimal and equivalent mutants based on source code structure

Master dissertation submitted to the Instituto de Ciências Matemáticas e de Computação – ICMC-USP, in partial fulfillment of the requirements for the degree of the Master Program in Computer Science and Computational Mathematics. *EXAMINATION BOARD PRESENTATION COPY*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. Márcio Eduardo Delamaro

USP – São Carlos
February 2022

Este trabalho é dedicado à minha esposa Marcela,
ao meu pai Claudinei, à minha mãe Rita e ao meu irmão
Luiz Felipe, que estiveram comigo e me apoiaram em todos
os momentos da minha carreira.

AGRADECIMENTOS

Agradeço primeiramente a Deus que permitiu que eu pudesse trilhar o caminho que hoje vivo e que eu pudesse desenvolver este trabalho. Ele que me guiou durante todo este processo, iluminando o caminho, colocando pessoas maravilhosas ao meu lado e me dando forças quando eu já não tinha.

Quero agradecer ao meu orientador, Dr. Márcio Eduardo Delamaro, que foi a pessoa que me guiou e orientou durante o desenvolvimento deste trabalho. Agradeço pela oportunidade de poder desenvolver este trabalho e pela confiança em mim depositada. Agradeço pela paciência ao longo de todo o período de desenvolvimento deste trabalho.

Agradeço à minha esposa Marcela Vasco Pereira Brito, que antes mesmo de ser minha esposa já me encorajava a me dedicar à este desafio e que em todos os momentos foi minha auxiliadora, mas nos momentos mais difíceis foi meu suporte para que eu pudesse buscar forças e não desanimar. Agradeço por ter me auxiliado em todos os momentos e por ter me dado todo o suporte e coragem que eu necessitei para desenvolver este trabalho.

Quero agradecer aos meus pais, Claudinei Brito e Rita de Cássia Almeida Brito e ao meu irmão, Luiz Felipe Brito, que em momento algum mediram esforços para que eu pudesse me dedicar à graduação e ao mestrado. Agradeço por cada bronca, conselho e suporte que me foi dado durante toda a minha vida e principalmente neste meu momento mais recente de vida. Agradeço às minhas avós, Adelina e Maria, que durante toda a minha vida me ensinaram muito e direcionaram em diversos momentos. Agradeço também ao meu avô Vicente (*in memorian*) que sempre foi lição de amor e estará sempre presente em nossa memória.

Agradeço também a todos os membros do LabES (professores e alunos) que me incentivaram, me guiaram e me ensinaram muito durante o tempo em que estive no laboratório. Agradeço por cada pessoa que direta ou indiretamente contribuiu para que este projeto pudesse ser desenvolvido. Agradeço por terem me ajudado e suportado sem esperar nada em troca e que de forma muito solícita, foram importantíssimos para que eu pudesse desenvolver este trabalho e todos os trabalhos relacionados.

Agradeço a todos os meus amigos, que aqui não foram citados, por todos os momentos compartilhados e opiniões sinceras. Aos professores e funcionários do ICMC agradeço pela dedicação e disposição nos serviços prestados.

Por fim, agradeço o suporte financeiro da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), sob o processo nº PROEX-10838581/M.

“Portanto, quer comais quer bebais,
ou façais outra qualquer coisa, fazei
tudo para glória de Deus.”
(1 Coríntios 10:31)

RESUMO

BRITO JUNIOR, C. **Uma abordagem para identificação de mutantes minimais e equivalentes baseada na estrutura do código fonte**. 2022. 115 p. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2022.

Com o objetivo de garantir que um software esteja sendo construído de acordo com suas especificações estabelecidas, a Engenharia de Software dispõe de uma série de atividades, processos e métodos que coletivamente são chamados de Validação, Verificação e Teste. O processo de teste de software pode ser definido como a execução de um programa com determinadas entradas e verificação da consonância das saídas com as saídas esperadas de acordo com as especificações do programa. Na tentativa de cumprir com seu objetivo, o teste de software pode ser dividido em diversas fases e etapas e conta com diversas técnicas e critérios, no qual cada técnica tem um objetivo específico e todas juntas se complementam. Um dos problemas fundamentais do teste de software é saber na prática ou na teoria quando já se testou o suficiente de um programa. O Teste de Mutação é um critério da técnica de Teste Baseado em Defeitos. É reconhecidamente um critério que pode auxiliar na criação de casos de teste com alta efetividade e capacidade de relevar defeitos. Esse critério mede a adequação de um determinado conjunto de casos de teste e assim fornece uma medida qualitativa da assertividade de um conjunto de teste com relação a um programa. A aplicação do teste de mutação se dá por meio de alterações no programa original, a fim de observar se o caso de teste consegue distinguir o comportamento do programa original e do programa alterado. Cada alteração realizada no programa original cria um novo programa chamado de Programa Mutante. Um dos problemas da aplicação do teste de mutação se dá pois normalmente são gerados muitos programas mutantes, mesmo para pequenos programas, o que eleva muito o custo computacional de geração e análise desses programas mutantes. Sendo assim, existem diversas técnicas que visam reduzir o custo computacional do teste de mutação. Essas técnicas normalmente são divididas em: (i) *Do Fewer*; (ii) *Do Smarter*; e (iii) *Do Faster*. Mutante minimal é um conceito pertencente às técnicas *Do Fewer* que procuram meios para reduzir o número de mutantes gerados. Este projeto tem como objetivo propor uma abordagem para identificação de mutantes minimais com base em suas localizações na estrutura do código-fonte e assim evitar a geração e execução de todos os mutantes. Para o desenvolvimento da abordagem, pretende-se utilizar uma representação de programas utilizadas na técnica de teste estrutural, o Grafo de Fluxo de Controle, e assim relacionar os mutantes minimais com lugares específicos do código, como por exemplo, nós essenciais do grafo de fluxo de controle. Uma vez desenvolvida a abordagem, pretende-se avaliá-la por meio de estudos empíricos com métricas já utilizadas na literatura em estudos prévios que visam reduzir o custo do teste de mutação. Como resultado final deste projeto, espera-se avançar o estado da arte na área de teste de software com uma abordagem mais eficiente para aplicação do critério teste de mutação.

Palavras-chave: Teste de Software, Teste de Mutação, Mutantes Minimais, Mutantes Equivalentes.

ABSTRACT

BRITO JUNIOR, C. **An approach to identifying minimal and equivalent mutants based on source code structure**. 2022. 115 p. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2022.

Aiming ensure that a software has been build in according with established specifications, Software Enginerring has a serie of activities, processes and methods that collectively are called Validation, Verification and Testing. The software testing process can be defined as the program execution with determined inputs and verification of outputs consonances with expected outputs according to program specification. Trying reach his object, software testing can be divided into several phases and steps and has several techniques and criterias, such as each technique has a specified aim and all together complement each other. One of software testing fundamental problem is to know in the practice and theory when have already tested sufficient. Mutation Testing is a criterion of Defect-Based Testing technique. It is recognized as a criterion that can aid in the creation of test cases with high effectiveness and ability to reveal defects. This criterion measures the adequacy of a given set of test cases and thus provides a qualitative measure of the assertiveness of a test set in relation to a program. The application of the mutation testing is done through changes in the original program, in order to observe if the test case can distinguish the behavior of the original program and the altered program. Each change made in the original program creates a new program called Mutant Program. One of the problems of applying the mutation test is that many mutant programs are generated, even for small programs, which greatly increases the computational cost of generating and analyzing these mutant programs. Thus, there are several techniques that aim to reduce the computational cost of the mutation test. These techniques are usually divided into: (i) Do Fewer; (ii) Do Smarter; and (iii) Faster. Minimal mutant is a concept belonging to Do Fewer techniques that seek means to reduce the number of mutants generated. This project aims to propose an approach to identify minimal mutants based on their locations in the source code structure and thus avoid the generation and execution of all mutants. For the development of the approach, we intend to use a representation of programs used in the structural test technique, the Control Flow Graph, and thus to relate the minimal mutants to specific places of the code, such as essential nodes of the control flow graph. Once the approach is developed, it is intended to be evaluated through empirical studies with metrics already used in the literature in previous studies aimed at reducing the cost of mutation testing. As a final result of this project, it is expected to advance the state of the art in the area of software testing with a more efficient approach to applying the mutation testing criterion.

Keywords: Software Testing, Mutation Testing, Minimal Mutants, Equivalent Mutants.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE ALGORITMOS

LISTA DE CÓDIGOS-FONTE

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ES	Engenharia de Software
GBS	*Greedy Backward Selection*
GFC	Grafo de Fluxo de Controle
Q-Q	*Quantile to Quantile*
VV&T	Validação, Verificação e Teste

SUMÁRIO

CAPÍTULO

1

INTRODUÇÃO

1.1 Contextualização e Motivação

A utilização de softwares no cotidiano de pessoas, empresas e indústrias, faz com que o rigor aplicado no desenvolvimento de software seja cada vez maior. O processo de desenvolvimento de software envolve uma série de etapas, normalmente com sujeitos de diversas áreas de conhecimento e muitas das vezes, boa parte do trabalho é feita de forma remota. São inúmeros os fatores que elevam a complexidade do desenvolvimento de software, portanto, a incidência de defeitos acaba se tornando muito propensa. Dessa forma, é necessário que existam processos, métodos e ferramentas que busquem garantir a qualidade do software. A Engenharia de Software (ES) é uma área do conhecimento do contexto de Ciência da Computação que envolve diversos processos, métodos e ferramentas, que visam garantir a qualidade de software (PRESSMAN, 2016).

De forma a tentar garantir que os artefatos e produtos sejam desenvolvidos em conformidade com as especificações estabelecidas, a ES dispõe de uma série de atividades que coletivamente são chamadas de Validação, Verificação e Teste (VV&T). Bertolino (2003) define que o processo de teste de software é executar o programa com determinadas entradas e então, verificar se o comportamento do programa condiz com as especificações. Delamaro, Jino e Maldonado (2017) destacam que a atividade de teste consiste de uma análise dinâmica do produto em construção, tornando-se assim, uma atividade indispensável para a identificação de possíveis defeitos que podem ter sido cometidos durante todo o processo de desenvolvimento de software, desde a especificação de requisitos até a implantação. O objetivo geral do teste de software é fornecer evidências sobre a confiabilidade do produto avaliado (MYERS; SANDLER; BADGETT, 2011). Assim sendo, existem diversas técnicas e critérios que são aplicados em conjunto e complementando-se um ao outro, de forma a garantir uma maior completude de verificação do software, vasculhando todo o produto e garantindo que todo o software foi avaliado e que todos os possíveis defeitos foram reportados.

Além das técnicas e critérios do teste de software, a sistematização do teste também é composta por fases e etapas. As etapas são separadas em:

- Planejamento;
- Projeto de casos de teste;
- Execução; e
- Análise.

Independente da técnica, critério ou fase do teste, essas etapas são cruciais para que o teste seja o mais efetivo possível. As fases do teste de software são:

- Teste de Unidade;
- Teste de Integração;
- Teste de Sistemas;
- Teste de Regressão.

O ciclo de teste de um software é a repetição cíclica e coordenada dessas fases, onde cada uma das fases tem um objetivo específico e visa encontrar um tipo de defeito.

Para cada uma das técnicas de teste, a origem da informação utilizada para a derivação dos requisitos de teste é diferente (DELAMARO; MALDONADO; JINO, 2017).

O Teste Funcional é uma técnica onde os casos de teste são projetados a partir da consideração que o programa é considerado como uma caixa preta (FABBRI; VINCENZI; MALDONADO, 2017), isto é, não é necessário o conhecimento da estrutura interna do programa para derivação dos casos de teste e avaliação. Os critérios de teste da técnica funcional são:

- Particionamento em Classes de Equivalência;
- Análise do Valor Limite;
- Grafo Causa-Efeito; e
- *Error Guessing*.

O Teste Estrutural estabelece os requisitos de teste com base na estrutura interna do programa, como se fosse uma caixa branca (BARBOSA *et al.*, 2017). Nesta técnica, são testados os caminhos lógicos do programa, pondo em prova conjuntos determinados de laços de repetição e estruturas condicionais. Os critérios de teste da técnica estrutural são:

- Critérios baseados na complexidade;
- Critérios baseados em fluxo de controle: Os principais critérios são Todos-nós, Todas-Arestas e Todos-Caminhos; e
- Critérios baseados em fluxo de dados: Os principais critérios são Rapps e Weyuker e Potenciais-usos.

O Teste Baseado em Defeitos deriva os requisitos de teste a partir de defeitos típicos que são cometidos durante o processo de desenvolvimento de software (DELAMARO *et al.*, 2017). O principal critério do teste baseado em defeitos é o Teste de Mutação. O teste baseado em defeitos visa medir a adequação de um conjunto de casos de teste (DEMILLO; OFFUTT, 1991). O teste de mutação se dá por meio da criação de programas mutantes que são criados com base no programa original, e são realizadas alterações sintáticas no programa, de forma a alterar sua semântica. Portanto, conforme Madeyski *et al.* (2014), os casos de teste que conseguem distinguir o programa original do programa mutante, têm maior probabilidade de revelar defeitos, assim, pode ser considerado mais adequado que outro caso de teste que não conseguiria fazer essa distinção.

As principais vantagens da aplicação do teste de mutação, segundo Chekam *et al.* (2017) são: (i) apontamento dos elementos que devem ser testados no momento do desenvolvimento dos casos de teste; (ii) fornecimento de critérios para determinação do término do teste; e (iii) quantificação da adequação e qualidade do conjunto de casos de teste. O segundo problema mencionado soluciona um dos problemas fundamentais do teste de software, que é saber na teoria e na prática quando parar de testar (PAPADAKIS *et al.*, 2019).

Ainda que conforme Chekam *et al.* (2017), o teste de mutação tenha muitas vantagens, Wong e Mathur (1995) mencionam que às vezes o teste de mutação se torna impraticável devido ao alto custo da aplicação da técnica. O alto custo computacional do teste de mutação se dá por possíveis três fatores: (i) Alto número de mutantes gerados: são gerados muitos mutantes, o que eleva o custo computacional para geração, execução e posteriormente, o custo de análise desses mutantes; (ii) Alto número de mutantes redundantes: São gerados muitos mutantes redundantes, que são equivalentes entre si. Tais mutantes redundantes, elevam o custo computacional da aplicação da técnica e não contribuem para a eficácia da técnica; (iii) Geração de mutantes equivalentes: que são mutantes equivalentes ao programa original, isto é, são sintaticamente diferentes, mas é impossível que haja um caso de teste que consiga distinguir o mutante equivalente do programa original.

1.2 Objetivos

Existem várias técnicas e abordagens que visam reduzir o custo do teste de mutação, porém, ainda não há um consenso e muito menos uma definição de uma abordagem ou técnica

que efetivamente reduza o custo do teste de mutação e mantenha sua efetividade.

Objetivos gerais

O objetivo geral deste projeto é utilizar técnicas de aprendizado de máquina para desenvolver modelos preditivos que auxiliem na redução do custo da aplicação do Teste de Mutação. A redução de custo objetificada neste projeto é referente à geração de mutantes redundantes e equivalentes. Fazendo previamente a identificação de mutantes não redundantes e mutantes equivalentes, de forma que sejam executados apenas os não redundantes e não equivalentes, possibilitando uma economia computacional e evitando uma perda de tempo na tentativa de matar um mutante que até então não se saberia se ele era equivalente ou não.

Objetivos específicos

A partir dos objetivos gerais, foram derivados os seguintes objetivos específicos:

1. Desenvolvimento de uma abordagem que permita a detecção dos mutantes pertencentes ao conjunto de mutantes minimais com base nas propriedades e características estruturais desses mutantes;
2. Desenvolvimento de uma abordagem que permita a detecção dos mutantes equivalentes com base nas propriedades e características estruturais desses mutantes;
3. Avaliação das abordagens desenvolvidas por meio de estudos experimentais;

1.3 Organização do trabalho

Esta dissertação está dividida em 7 capítulos. Este primeiro capítulo introdutório aborda a contextualização do projeto, bem como os objetivos traçados para o desenvolvimento deste trabalho. O Capítulo 2 apresenta os principais conceitos de Teste de Mutação que são necessários para compreensão para o desenvolvimento deste trabalho. O referido capítulo inicia com os conceitos acerca do Teste de Mutação, apresentando os pontos positivos em utilizar este critério, bem como os pontos negativos que dificultam sua adoção na indústria de software. Em seguida, são apresentados os conceitos sobre mutantes equivalentes, mutantes redundantes e mutantes minimais.

O Capítulo 3 descreve técnicas e processos existentes para identificação de mutantes minimais e equivalentes. Identificar e classificar cada uma dessas classes de mutantes, ainda é um desafio, e este desafio é um dos principais motivos que elevam o custo computacional e pessoal do Teste de Mutação.

O Capítulo 4 apresenta os conceitos de Aprendizado de Máquina que foram necessários para o desenvolvimento deste trabalho e que são necessários para a compreensão das atividades

realizadas. Na parte inicial do capítulo, são listados os tipos de aprendizado, seguido dos modelos de classificação utilizados aqui neste trabalho e por último são apresentados os conceitos das métricas utilizadas para avaliação de modelos preditivos.

O Capítulo 5 apresenta a abordagem desenvolvida para identificação de mutantes minimais e equivalentes. O capítulo se inicia com uma visão geral acerca da abordagem, seguida pelos procedimentos mais específicos que foram: a geração dos dados dos mutantes e a aplicação do aprendizado de máquina.

O Capítulo 6 faz a avaliação experimental da abordagem desenvolvida neste trabalho. A primeira parte do capítulo define o planejamento para o experimento, seguida da análise dos resultados dos classificadores, depois disso é realizado o teste de hipóteses e por último são listadas as ameaças à validade do experimento.

O último capítulo, o Capítulo 7 apresenta as conclusões deste trabalho. O capítulo é iniciado com uma visão geral do trabalho realizado, seguida das contribuições, dificuldades e limitações e por último, os trabalhos futuros.

CAPÍTULO

2

FUNDAMENTOS SOBRE TESTE DE MUTAÇÃO

2.1 Considerações iniciais

O teste de mutação ou análise de mutantes é um critério de teste da técnica de teste baseado em defeitos. Conforme DeMillo, Lipton e Sayward (1978), esse critério mede a adequação de um dado conjunto de casos de teste. De forma prática, esse critério faz com que o responsável pelo teste interaja com um sistema de mutação automatizado para determinar e melhorar a adequação de um dado conjunto de casos de teste, forçando-o a testar típicos defeitos cometidos na implementação de um software. Papadakis *et al.* (2019) destacam que um dos problemas fundamentais no teste de software é a incapacidade de saber prática ou teoricamente quando se testou suficientemente, enquanto Chekam *et al.* (2017) destacam que existem três vantagens em utilizar o teste de mutação: (i) apontar os elementos que devem ser testados ao desenvolver os casos de teste; (ii) fornecer critérios para o término do teste (quando a cobertura é atingida); e (iii) quantificar a adequação do conjunto de casos de teste.

De acordo com a hipótese do programador competente, os programadores escrevem programas quase perfeitos, e que os erros cometidos são pequenos defeitos sintáticos e que podem ser corrigidos facilmente (DEMILLO; LIPTON; SAYWARD, 1978). Dessa forma, na prática o teste de mutação cria programas denominados programas mutantes ou alternativos, que são cópias do programa original com algumas alterações sintáticas. Aceitando a hipótese do programador competente como verdadeira, compreende-se que as alterações feitas no programa original, não inserem erros sintáticos (que fariam que o programa não fosse mais compilável), mas introduzem defeitos artificiais que alterariam a semântica do programa (PAPADAKIS *et al.*, 2019).

Essas alterações no programa não são feitas de maneira aleatória, mas de forma sistemática aplicando determinados operadores de mutação, que nada mais são que a aplicação

dos erros de implementação mais corriqueiros. Portanto, os casos de teste capazes de identificar a diferença no comportamento do programa original e do programa mutante, podem ser considerados adequados pois um caso de teste capaz de identificar uma pequena alteração no comportamento do programa devido ao defeito inserido, é capaz também de revelar outros tipos de defeitos (MADEYSKI *et al.*, 2014).

Cada paradigma e linguagem de programação, podem ter seus operadores de mutação específicos. O Apêndice A lista os operadores de mutação das linguagens *Fortran-77*, *C* e *Java*. A Figura 1 mostra como seria a aplicação de 2 operadores de mutação da linguagem *C* utilizando parte do Código-fonte 1. A ferramenta "Proteum/IM"[1] desenvolvida por Delamaro, Maldonado e Vincenzi (2001) foi utilizada para gerar programas mutantes e exemplificar seu funcionamento. Tendo como exemplo o Código-fonte 1, ao gerar mutantes para todos os operadores de mutação, foram gerados 166 mutantes.

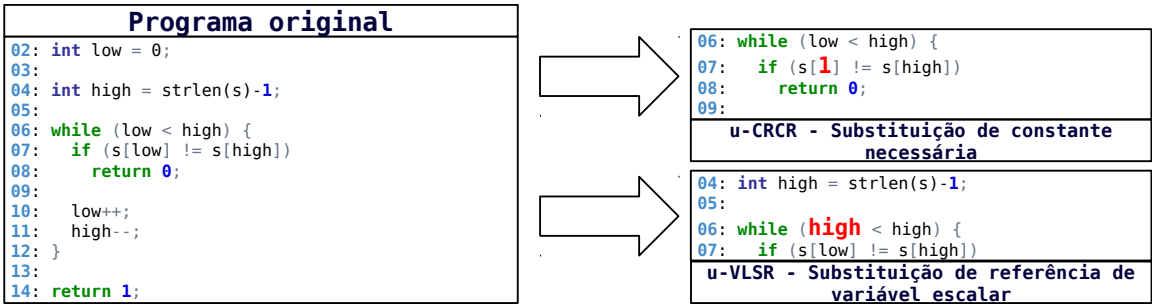
Código-fonte 1 – Função isPalindrome retirado do programa CheckPalindrome

```
 1: int isPalindrome(char s[]) {
 2:   int low = 0;
 3:
 4:   int high = strlen(s)-1;
 5:
 6:   while (low < high) {
 7:     if (s[low] != s[high])
 8:       return 0;
 9:
10:     low++;
11:     high--;
12:   }
13:
14:   return 1;
15: }
```

Após os mutantes terem sido gerados, deve-se analisar o resultado da execução de cada um dos mutantes sob o conjunto de casos de teste (denotado por T), espera-se que o comportamento dos programas mutantes seja diferentes do comportamento do programa original (denotado por P). Se um dado mutante (denotado por M) tem resultado diferente do resultado esperado da execução de P, diz-se que M foi morto, portanto, descartado, porém, se o resultado é o mesmo de P, T não conseguiu diferenciar entre M e P e diz-se que M permanece vivo. A existência de mutantes vivos pode indicar que:

[1] Informações sobre outras ferramentas que implementam a execução e análise de mutantes podem ser obtidas no Seção 2.3

Figura 1 – Aplicação dos operadores de mutaçao u-CRCR e u-VLSR

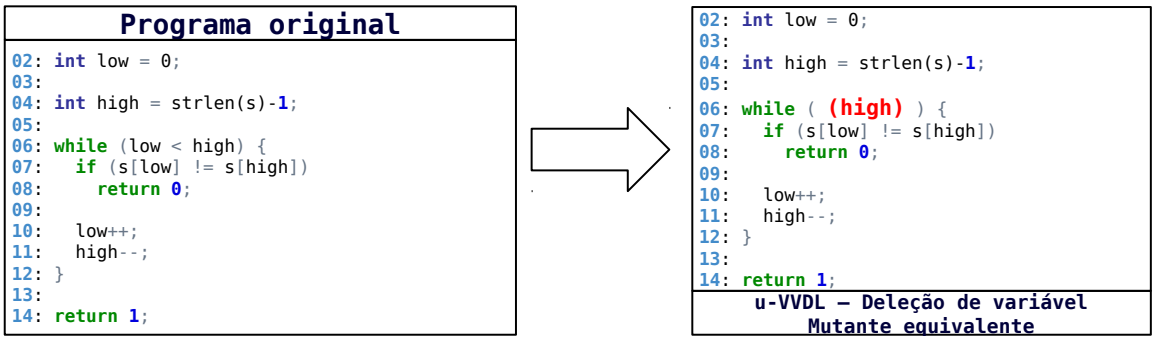


Fonte: Elaborada pelo autor.

- T tem uma vulnerabilidade e pode não conseguir identificar quando um programador cometer um erro igual ao erro introduzido por M; ou

- M é um Mutante Equivalente (denotado por ME), isto é, seu comportamento é igual ao comportamento de P, portanto, M e P são indistinguíveis, fazendo com que seja impossível matar M. Hierons, Merayo e Núñez (2010) destacam que decidir se um mutante é ou não um mutante equivalente, é um problema geralmente indecidível, dessa forma, é necessário que o analista de teste ou testador analise todos os mutantes vivos e assim faça a classificação daqueles que são equivalentes, a fim de não perder tempo tentando matar esse mutante e também incorporar esse número no cálculo do escore de mutação que será tratado logo adiante. Tomando como exemplo o Código-fonte 1, a figura Figura 2 exibe um ME ao P.

Figura 2 – Programa original e mutante equivalente



Fonte: Elaborada pelo autor.

Uma vez que já se tem o número de mutantes mortos, é possível calcular o nível de adequação de um conjunto de teste através do escore de mutação (*mutation score*). Conforme Ammann e Offutt (2016), o escore de mutação pode ser utilizado como uma métrica de adequação do conjunto de casos de teste com relação ao programa. O escore de mutação é um número entre 0 e 1, portanto, quanto mais próximo do 1 for o escore de mutação, maior é o nível de adequação

do conjunto de casos de teste. O cálculo do escore de mutação *ms(P, T)* do programa P com relação ao conjunto de casos de teste T se dá da seguinte forma:

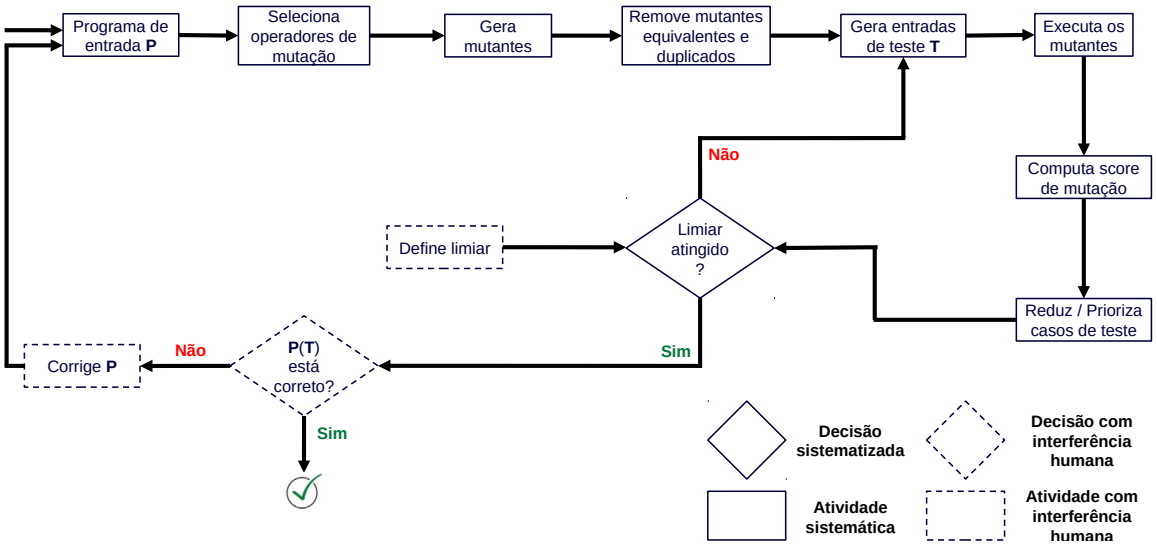
$$ms(P,T) = \frac{MM}{TM - ME}$$

onde:

- ***ms(P, T)***: Escore de mutação do programa P com relação ao conjunto de casos de teste T;
- **MM**: Número de mutantes mortos pelo conjunto de casos de teste;
- **TM**: Número total de mutantes gerados;
- **ME**: Número de mutantes equivalentes.

Offutt e Untch (2001) definem um processo genérico do teste de mutação. Devido aos avanços da área, Papadakis *et al.* (2019) adaptaram aquele processo, para um processo moderno. O processo moderno envolvido no teste de mutação é representado na Figura 3. Os retângulos e losangos com bordas tracejadas, representam atividades que demandam a interação humana, enquanto os retângulos e losangos com bordas contínuas representam atividades que podem ser sistematizadas.

Figura 3 – Processo genérico do teste de mutação



Fonte: Adaptada de Papadakis *et al.* (2019).

2.2 Custo de geração e execução de todos os mutantes

Mesmo sendo reconhecidamente um critério que pode auxiliar na criação de casos de teste com alta efetividade e capacidade de revelar defeitos, o teste de mutação ainda é considerado

muito custoso, isso acontece pois normalmente são gerados muitos mutantes, mesmo para programas simples e pequenos (DELAMARO; CHAIM; MALDONADO, 2018). O problema do alto número de mutantes gerados, normalmente se restringe à: mutantes equivalentes e mutantes redundantes (FERNANDES *et al.*, 2017), dessa forma, eleva-se o custo computacional, na tarefa de executar todos os programas mutantes (incluindo os redundantes) e custo pessoal, na tarefa de identificar quais são os mutantes equivalentes.

2.2.1 Técnicas para redução do custo

Offutt e Untch (2001) relatam que as abordagens para redução do custo do teste de mutação comumente são classificadas entre *Do fewer*, *Do smarter* ou *Do faster*.

Do fewer

Procura meios para reduzir o número de mutantes gerados, mas sem perda de informações. O presente trabalho aplica esta estratégia. Entre as abordagens desta estratégia estão:

- Mutação Restrita ou Seletiva[2];
- Seleção Aleatória ou Mutação por Amostragem[2].

Do smarter

Procura distribuir o custo computacional entre várias máquinas, dividir o custo computacional em diversas execuções, ou, evitar a execução completa de uma só vez. Entre as abordagens desta estratégia destaca-se:

- Mutação Fraca: Conforme Howden (1982 apud OFFUTT; UNTCH, 2001), a mutação fraca é uma técnica de aproximação que compara os estados internos dos programas mutantes e originais imediatamente após a execução da mutação de determinada parte do programa.

Do faster

Procura focar em meios de geração e execução de cada mutante o mais rápido possível. Entre as abordagens desta estratégia está:

- Análise de Mutantes Baseada em Esquema: Um esquema de programa é um modelo. De acordo com Baruch e Katz (1988 apud UNTCH; OFFUTT; HARROLD, 1993a), um esquema de programa parcialmente interpretado se assemelha sintaticamente a um programa, porém, contém identificadores livres, que são chamados de entidades abstratas, utilizados no lugar de variáveis, identificadores de tipo de dado, constantes e instruções

[2] Esta estratégia é detalhada na Subseção 2.2.1.

de programa (UNTCH; OFFUTT; HARROLD, 1993a). Conforme Untch (1992), essa técnica junta todos os mutantes num só programa, denominado "metamutante", e então, esse programa é compilado apenas uma vez sob as mesmas condições de desenvolvimento que o programa original, dessa forma, diminuindo significativamente o custo de análise e execução dos mutantes. O processo de geração de metamutantes começa com a construção de uma árvore sintática abstrata decorada.

Ferrari, Pizzoleto e Offutt (2018) realizaram uma revisão sistemática da literatura, analisando os trabalhos publicados entre 1988 e 2017 na área de teste de mutação, focando em técnicas para redução do custo. No estudo, foram encontrados 146 trabalhos, desses, 110 (75,34%) utilizavam as técnicas *do fewer* como técnica primária do estudo, 22 (15,06%) utilizavam as técnicas *do smarter* como técnica primária do estudo, e 14 (9,58%) utilizavam as técnicas *do faster* como técnica primária do estudo. Dos 110 trabalhos que utilizam a técnica *do fewer*, 20 (18,18%) são referentes à mutação seletiva e 12 (10,90%) utilizam a mutação por amostragem.

2.2.1.1 Redução do número de mutantes - do fewer

Mathur e Wong (1993) utilizam a Seleção Aleatória como mecanismo para reduzir o custo do teste de mutação. A seleção aleatória examina uma pequena porcentagem aleatória de cada operador de mutação, e então, ignora-se o restante dos mutantes, porém, é garantido que todos os operadores de mutação são utilizados.

Wong e Mathur (1995) avaliaram a aplicação da Seleção Aleatória utilizando sete categorias diferentes, cada categoria representava um percentual (10%, 15%, 20%, 25%, 30%, 35% e 40%) de operadores de mutação diferentes. Por utilizar um maior número de mutantes, a categoria que alcançou o maior escore de mutação foi a com 40% de mutantes por operador de mutação, enquanto a que obteve o menor escore de mutação foi a categoria com 10% de mutantes por operador de mutação. O escore de mutação obtido para cada uma das categorias é exibido ao final da seção na Tabela 1.

Uma deficiência da Seleção Aleatória é não levar em consideração as características e a capacidade de revelar defeitos de cada um dos operadores de mutação. Mathur (1991) propôs uma estratégia nomeada Mutação Restrita (*Constrained Mutation*). Offutt, Rothermel e Zapf (1993) adaptaram o termo Mutação Restrita para Mutação Seletiva (*Seletive Mutation*), desde então, alguns autores utilizam o termo Mutação Restrita, enquanto outros utilizam o termo Mutação Seletiva. Devido à pequenas diferenças entre as abordagens, este trabalho baseia-se em Jia e Harman (2011) que compreendem as duas abordagens como Mutação Seletiva[3]. Esta estratégia seleciona apenas alguns operadores de mutação para geração de mutantes e análise, enquanto todos os outros mutantes dos outros operadores de mutação são ignorados.

[3] Para maiores detalhes sobre a diferença entre as abordagens, consultar Ferrari, Pizzoleto e Offutt (2018)

Mathur e Wong (1993), Wong e Mathur (1995) e Wong e Mathur (1995) através de experimentação para analisar os resultados desta estratégia de redução de mutantes, observaram que através da mutação restrita, escolhendo os operadores *abs* [4] e *ror* [5], executando aproximadamente apenas 18% dos mutantes, puderam alcançar um escore de mutação de aproximadamente 92%, isto é, com uma redução de mais de 80% do custo de teste, sacrificou-se apenas 8% do escore de mutação.

Mathur e Wong (1993) relatam que comparando o tamanho, custo e relativo esforço, não existe diferença significativa entre a mutação restrita (utilizando os operadores *abs* e *ror*) e a mutação aleatória (utilizando 10% de cada um dos operadores de mutação). Zhang *et al.* (2010a) relatam que em termos de efetividade ou estabilidade, as técnicas de seleção de mutantes com base nos operadores experimentadas (Offutt *et al.* (1996), Barbosa, Maldonado e Vincenzi (2001) e Namin, Andrews e Murdoch (2008)) não são superiores à mutação aleatória.

Offutt, Rothermel e Zapf (1993) utilizaram a Mutação Seletiva, testando 10 programas escritos em Fortran-77. Em sua primeira execução, nomeada *2-Selective Mutation*, foram excluídos apenas dois operadores de mutação, *ASR* e *SVR*. Posteriormente, foi feita uma execução semelhante, nomeada *4-Selective Mutation*, excluindo quatro operadores de mutação, os dois anteriores, *CSR* e *SCR*. Na terceira execução, nomeada *6-Selective Mutation*, foram excluídos seis operadores de mutação, os quatro anteriores, *ARC* e *ACR*. Os resultados obtidos para cada uma das categorias são exibidos ao final da seção na Tabela 1.

Offutt *et al.* (1996) definiram três categorias para utilização da mutação seletiva com base nas classes de operadores de mutação da ferramenta *Mothra*, *ES-Selective* que utiliza operadores *expression/statement*, *RS-Selective* que utiliza operadores *replacement/statement* e *RE-Selective* que utiliza operadores *replacement/expression*. Os resultados obtidos para cada uma das categorias são exibidos ao final da seção na Tabela 1.

Após analisar os resultados e observar que o escore de mutação obtido pela categoria *RS-Selective* foi menor que o escore das outras duas categorias, os autores propuseram uma nova categoria, denominada *E-Selective*, que faz a combinação das categorias de mutação seletiva *ES* e *RE*. Essa nova categoria reduziu em 77,56% o custo da análise de mutantes, mas mantendo um escore de mutação alto, de 99,51%.

WONG *et al.* (1997) utilizaram onze operadores de mutação para avaliar a aplicabilidade da mutação seletiva, entre esses onze operadores, foram selecionadas seis combinações de mutação seletiva. As combinações de mutação seletiva obtiveram um escore de mutação médio de 77,22%. As combinações de mutação restrita e seus respectivos operadores de mutação são exibidos no Quadro 1. Os resultados obtidos para cada uma das categorias são exibidos ao final da seção na Tabela 1.

[4] O operador de mutação *abs* gera mutantes trocando $abs(x)$ por $-abs(x)$ e $zpush(x)$ onde for possível.
[5] O operador de mutação *ror* gera mutantes trocando cada operador relacional por outro operador relacional.

Quadro 1 – Categorias de mutação seletiva - WONG *et al.* (1997)

Categoria	Operadores de mutação
MUT1	olln, olng, orrn
MUT2	olln, olng, orrn, ocng, orln, olrn, olan, oaln
MUT3	vtdr, vtwd
MUT4	strp
MUT5	olln, olng, orrn, vdtr, vtwd
MUT6	olln, olng, orrn, vdtr, vtwd, strp

Fonte: WONG *et al.* (1997).

Barbosa, Vincenzi e Maldonado (1998) definiram seis categorias de mutação seletiva com base nas quatro classes[6] de operadores de mutação da ferramenta *Proteum*. As categorias de mutação seletiva são descritas no Quadro 2. Os resultados obtidos para cada uma das categorias são exibidos ao final da seção na Tabela 1.

Quadro 2 – Categorias de mutação seletiva - Barbosa, Vincenzi e Maldonado (1998)

Categoria	Descrição
Stat-Oper	Utiliza os operadores das classes *statement* e *operator*
Stat-Var	Utiliza os operadores das classes *statement* e *variable*
Stat-Cons	Utiliza os operadores das classes *statement* e *constant*
Oper-Var	Utiliza os operadores das classes *operator* e *variable*
Oper-Cons	Utiliza os operadores das classes *operator* e *constant*
Var-Cons	Utiliza os operadores das classes *variable* e *constant*

Fonte: Barbosa, Vincenzi e Maldonado (1998).

Por observarem que as três categorias seletivas com maior escore de mutação contém a classe de operadores de mutação de constantes, foi criada a categoria *Cons* que utiliza apenas os operadores da classe de mutação de constantes. Com esta nova categoria de mutação seletiva, foi obtido um escore de mutação de 97,14% com 78,11% de redução de custo da análise de mutantes.

Os autores Barbosa, Vincenzi e Maldonado (1998) ainda trabalharam com uma segunda definição categoria de mutação seletiva, nomeada $MUT6$, composta pelos operadores de mutação OLLN, OLNG, ORRN, VDTR, VTWD e STRP. Com esta categoria seletiva, obtiveram um escore de mutação médio de 98% e uma redução de custo média de 79,70%. Complementarmente à categoria seletiva MUT6, foram incluídos os operadores de mutação OCNG, ORLN, OLRN, OLAN e OALN, obtendo assim um escore de mutação médio de 98,10% e uma redução de custo média de 73,85%.

Barbosa, Maldonado e Vincenzi (2001) definiram um procedimento para a determinação dos operadores suficientes com respeito ao baixo custo e eficiência. Em um primeiro experimento,

[6] Mutação de comandos (*statement mutations*), mutação de operadores (*operator mutations*), mutação de variáveis (*variable mutations*) e mutação de constantes (*constant mutations*).

os autores chegaram nos operadores de mutação SWDD, SMTC, SSDL, OLBN, OASN, ORRN, VTWD, VDTR, Cccr, Ccsr, conjunto denominado *SS-27*. No segundo experimento (dessa vez alterando o conjunto de casos de teste), os operadores de mutação considerados suficientes foram SMTC, SSDL, OEBA, ORRN, VTWD, VDTR, conjunto denominado *SS-5*. Os dados referentes ao escore de mutação e redução do custo obtidos pelos autores pode ser encontrado na Tabela 1.

Zhang *et al.* (2013) propuseram a utilização das estratégias Mutação Seletiva e Seleção aleatória juntas. Os autores definiram oito categorias de Mutação Seletiva. Para cada uma das categorias de mutação seletiva são selecionadores mutantes aleatoriamente com base em um determinado percentual do de determinada origem. As categorias e as origens dos mutantes selecionados são exibidos no Quadro 3. Os dados referentes à média do escore de mutação e da redução do custo médio obtidos pelos autores pode ser encontrado na Tabela 1.

Quadro 3 – Categorias de mutação seletiva - Zhang *et al.* (2013)

Categoria	**Origem**
Base	% do total de mutantes
MOp	% do total de mutantes de cada operador de mutação
Class	% do total de mutantes de uma classe
Meth	% do total de mutantes de um método
Stmt	% do total de mutantes de uma instrução
Class-MOp	% do total de mutantes de cada operador de mutação de uma classe
Meth-MOp	% do total de mutantes de cada operador de mutação de um método
Stmt-MOp	% do total de mutantes de cada operador de mutação de uma instrução

Fonte: Zhang *et al.* (2013).

Delamaro *et al.* (2014) avaliaram a utilização de apenas um operador de mutação, o Operador de Delação de Instrução (SSDL - Statement Deletion Operator). O resultado obtido pode ser encontrado no final da seção na Tabela 1.

Uma visão geral dos resultados obtidos por cada autor relatado aqui nesta seção pode ser observada na Tabela 1.

2.2.2 Mutantes minimais

Para a compreensão do que são mutantes minimais, primeiro, é necessário compreender os conceitos que Ammann, Delamaro e Offutt (2014), Kurtz *et al.* (2014) definiram até chegar nos mutantes minimais. Para facilitar a compreensão, o exemplo retirado de Ammann, Delamaro e Offutt (2014) que está na Tabela 2 será utilizado em todos as seções subsequentes. A Tabela 2 contém cinco casos de teste, $T = \{t1, t2, t3, t4, t5\}$, quatro mutantes, $M = \{m1, m2, m3, m4\}$, escore de mutação de 100% e pode ser lida da seguinte forma:

- O mutante $m1$ é morto pelos casos de teste $t1$, $t3$, $t4$ e $t5$;
- O mutante $m2$ é morto pelos casos de teste $t1$, $t2$, $t4$ e $t5$;

Tabela 1 – Visão geral dos resultados obtidos por cada autor

Autor	**SA**	**MS**	**Categoria**	**EM (%)**	**RC (%)**
Wong e Mathur (1995)	✓		10%	97,56	90,00
	✓		15%	97,77	85,00
	✓		20%	97,95	80,00
	✓		25%	98,68	75,00
	✓		30%	99,10	70,00
	✓		35%	99,26	65,00
	✓		40%	99,39	60,00
Mathur e Wong (1993)		✓	Abs e Ror	92,00	82,00
Offutt, Rothermel e Zapf (1993)		✓	2-SM	99,99	23,98
		✓	4-SM	99,84	41,36
		✓	6-SM	99,71	60,56
Offutt *et al.* (1996)		✓	ES-Selective	99,45	71,52
		✓	RS-Selective	97,31	22,44
		✓	RE-Selective	99,97	6,04
		✓	E-Selective	99,51	77,56
WONG *et al.* (1997)		✓	MUT1	76,67	4,49
		✓	MUT2	70,00	9,05
		✓	MUT3	93,33	20,85
		✓	MUT4	33,33	3,85
		✓	MUT5	93,33	25,34
		✓	MUT6	96,77	29,19
Barbosa, Vincenzi e Maldonado (1998)		✓	Var-Cons	98,82	51,91
		✓	Stat-Cons	98,66	63,28
		✓	Oper-Cons	98,43	41,05
		✓	Stat-Oper	98,40	48,08
		✓	Stat-Var	98,05	58,95
		✓	Oper-Var	97,77	36,73
		✓	Cons	97,14	78,11
Barbosa, Maldonado e Vincenzi (2001)		✓	SS-27	99,66	34,98
		✓	SS-5	99,76	17,95
Zhang *et al.* (2013)	✓	✓	5%	97,77	95,00
	✓	✓	10%	97,95	90,00
	✓	✓	15%	98,68	85,00
	✓	✓	20%	99,10	80,00
	✓	✓	Base	99,74	87,50
	✓	✓	MOp	99,73	87,50
	✓	✓	Class	99,75	87,50
	✓	✓	Meth	99,78	87,50
	✓	✓	Stmt	99,78	87,50
	✓	✓	Class-MOp	99,74	87,50
	✓	✓	Meth-MOp	99,78	87,50
	✓	✓	Stmt-MOp	99,75	87,50
Delamaro *et al.* (2014)		✓	SSDL	96,00	96,74

Legenda – SA: Seleção Aleatória MS: Mutação Seletiva EM: Escore de Mutação RC: Redução de custo

Fonte: Elaborada pelo autor.

Tabela 2 – Escore de mutação de exemplo

	m1	**m2**	**m3**	**m4**
t1	✓	✓		✓
t2		✓	✓	
t3	✓		✓	
t4	✓	✓	✓	✓
t5	✓	✓		

Fonte: Ammann, Delamaro e Offutt (2014).

- O mutante $m3$ é morto pelos casos de teste $t2$, $t3$, e $t4$;
- O mutante $m4$ é morto pelos casos de teste $t1$ e $t4$.

Conjunto adequado de testes

Um conjunto de teste (denotado $\hat{\mathbf{T}}$) é adequado se e somente se para qualquer $\mathbf{t_i} \in \hat{\mathbf{T}}$, que ao retirar $\mathbf{t_i}$ de $\hat{\mathbf{T}}$, o escore de mutação de M com relação a T é alterado, ou seja, o conjunto de teste só é adequado se caso cada um dos casos de teste forem removidos individualmente, o escore de mutação será alterado em cada remoção, portanto, cada caso de teste é importante para que o escore de mutação seja mantido. Seguindo essa definição, a partir do exemplo da Tabela 2, é possível observar que existem três conjuntos adequados de testes:

$$\bar{\mathbf{T}} = \{\{t4\}, \{t1, t2\}, \{t1, t3\}\}$$

Mutantes redundantes

Um mutante m_j é redundante com respeito ao conjunto de mutantes M e ao conjunto de teste T se e somente se $\bar{\mathbf{T}}_{\mathbf{M}} = \bar{\mathbf{T}}_{\mathbf{M - M_j}}$, ou seja, o mutante só é redundante se a remoção deste mutante não faz alteração nenhuma no conjunto de testes adequado. Abaixo é possível visualizar o método para encontrar os mutantes redundantes do exemplo da Tabela 2:

- $\bar{\mathbf{T}}_{\mathbf{M}} = \{\{t4\}, \{t1, t2\}, \{t1, t3\}\}$;
- $\bar{\mathbf{T}}_{\mathbf{M - m1}} = \{\{t4\}, \{t1, t2\}, \{t1, t3\}\}$;
- $\bar{\mathbf{T}}_{\mathbf{M - m2}} = \{\{t4\}, \{t1, t2\}, \{t1, t3\}\}$;
- $\bar{\mathbf{T}}_{\mathbf{M - m3}} = \{\{t1\}, \{t4\}\}$;
- $\bar{\mathbf{T}}_{\mathbf{M - m4}} = \{\{t4\}, \{t1, t2\}, \{t1, t3\}, \{t2, t5\}, \{t3, t5\}\}$.

É possível visualizar que ao retirar os mutantes $m1$ e $m2$, o conjunto adequado de testes $\bar{\mathbf{T}}$ permanece o mesmo, portanto, esses mutantes são redundantes.

Conjunto de mutantes minimais

Um conjunto de mutantes (denotado por M) é minimal se não contém nenhum mutante redundante. Portanto, com base no exemplo e nos passos anteriores (identificação do conjunto de teste minimal e dos mutantes redundantes), a Tabela 3 exibe os mutantes minimais com relação ao escore de mutação da Tabela 2.

Tabela 3 – Escore de mutação de exemplo - Mutantes minimais

	m3	**m4**
t1		✓
t2	✓	
t3	✓	
t4	✓	✓
t5		

Fonte: Ammann, Delamaro e Offutt (2014).

2.3 Ferramentas

Sem a utilização de ferramentas, o teste de mutação (assim como qualquer outro critério de teste) é afetado devido a uma série de passos complexos envolvidos no processo. Delamaro, Jino e Maldonado (2017) apontam várias ferramentas que foram desenvolvidas desde o desenvolvimento da técnica até o desenvolvimento deste trabalho. Jia e Harman (2011) e Papadakis *et al.* (2019) apresentam um catálogo com as principais ferramentas utilizadas no suporte à aplicação do teste de mutação. A Seção 2.3 sumariza as informações exibindo referências, nomes e as linguagens de aplicação das ferramentas. As ferramentas que foram desenvolvidas mas não tiveram um nome definido não foram exibidas no Seção 2.3. É notável que as linguagens *Java* e *C* tem recebido maior atenção dos pesquisadores, com um vasto número de ferramentas para aplicação do critério, 19 (correspondente à 26,76%) e 15 (correspondente à 21,13%) respectivamente. Para a demonstração de alguns exemplos ao longo deste texto e para a execução do experimento, foi utilizada a ferramenta *ProteumIM*.

Tabela 4 – Ferramentas que dão suporte à aplicação do teste de mutação

Referência	**Ferramenta**	**Linguagem**
Jabbarvand e Malek (2017)	μDroid	Android apps
Delamare *et al.* (2009) e Delamare, Baudry e Traon (2009)	AjMutator	AspectJ
Mateo, Usaola e Offutt (2010)	Bacterio	Java
Usaola *et al.* (2017)	BacterioWeb	Android apps
Do e Rothermel (2006)	ByteME	Java

Continuação da Tabela 4

Referência	Ferramenta	Linguagem
Kusano e Wang (2013)	CCMUTATOR	C/C++
Acree Jr. (1980) e Hanks (1980)	CMS.1	Cobol
Gligoric *et al.* (2013)	Comutation	Java
Derezinska e Szustek (2008)	Cream	C#
Zip... (2007) e Ellims, Ince e Petre (2007)	CSAW	C
Feng, Marr e O'Callaghan (2008)	ESTP	C/C++
Bradbury, Cordy e Dingel (2006)	ExMan	C/Java
Acree *et al.* (1979), Budd (1981) e Budd, Hess e Sayward (1980)	EXPER	Fortran
Tanaka (1981)	FMS.3	Fortran
Domínguez-Jiménez, Estero-Botaro e Medina-Bulo (2009)	GAmera	WS-BPEL
Zhang *et al.* (2010b)	GenMutants	.Net
Confessions... (2013)	Heckle	Ruby
Omar, Ghosh e Whitley (2014)	HOMAJ	AspectJ/Java
Derezinska e Kowalski (2011)	ILMutator	C#
Parasoft... (2019)	Insure++	C/C++
Schuler e Zeller (2009)	Javalanche	Java
Chevalley e Thévenod-Fosse (2003)	JavaMut	Java
Zhou e Frankl (2009) e Zhou e Frankl (2011)	JDama	SQL/JDBC
Jester (2005)	Jester	Java
Dadeau, Héam e Kheddam (2011)	jMuHLPSL	HLPSL
Madeyski e Radyk (2010)	Judy	Java
Jumble (2015)	Jumble	Java
Parsai, Murgia e Demeyer (2017)	LittleDarwin	Java
Just (2014)	Major	Java
Linares-Vásquez *et al.* (2017)	MDroid+	Android apps
Jia e Harman (2008)	Milu	C
Krenn *et al.* (2015)	MoMut	UML models
DeMillo *et al.* (1987) e DeMillo *et al.* (1988)	Mothra	Fortran
MuBPEL... (2017)	MuBPEL	WS-BPEL
Le *et al.* (2014)	MuCheck	Haskell
Smith e Williams (2007)	MuClipse	Java

Continuação da Tabela 4

Referência	Ferramenta	Linguagem
Delgado-Pérez *et al.* (2017)	MuCPP	C++
Shahriar e Zulkernine (2008b)	MUFORMAT	C
Kim, Harrold e Kwon (2006)	MUGAMMA	Java
Ma, Offutt e Kwon (2005), Ma, Offutt e Kwon (2006) e Offutt, Ma e Kwon (2004)	muJava	Java
Shahriar e Zulkernine (2008a)	Music	SQL
Henard, Papadakis e Traon (2014)	MutaLog	Logic
Mirshokraie, Mesbah e Pattabiraman (2013) e Mirshokraie, Mesbah e Pattabiraman (2015)	Mutandis	JavaScript
Li *et al.* (2015) e GitHub... (2019)	mutant (muRuby)	Ruby
mutatepy... (2017)	mutate	C
GitHub... (2010)	MutateMe	PHP
Andrews e Zhang (2003) e Andrews, Briand e Labiche (2005)	mutgen	C
Gligoric, Jagannath e Marinov (2010)	MutMut	Java
Derezińska e Hałas (2014)	MutPy	Python
Arcaini, Gargantini e Riccobene (2017)	MutRex	Regular expressions
Tokumoto *et al.* (2016)	MuVM	C
Nester... (2013)	Nester	C#
Madiraju e Namin (2011)	Paraμ	Java
Andrés, Merayo e Núñez (2009), Andrés, Merayo e Molinero (2009) e Andrés, Merayo e Núñez (2012)	PASTE	TFSM
Jester (2005)	Pester	Python
Budd *et al.* (1978), Budd e Sayward (1977) e Lipton e Sayward (1978)	PIMS	Fortran
Coles *et al.* (2016)	PIT	Java
PlexTest... (2009)	PlexTest	C/C++
Delamaro, Maldonado e Vincenzi (2001)	Proteum	C
Walsh, McMinn e Kapfhammer (2015)	REDECHECK	HTML/CSS

Continuação da Tabela 4

Referência	Ferramenta	Linguagem
Kapfhammer, McMinn e Wright (2013)	SchemaAnalyst	SQL
Crouzet *et al.* (2006)	SESAME	C/Lustre/Pascal
Dan e Hierons (2012)	SMT-C	C
Gligoric, Badame e Johnson (2011)	SMutant	Smalltalk
Tuya, Suarez-Cabal e Riva (2006)	SQLMutation	SQL
Ghosh, Govindarajan e Mathur (2001)	TDS	CORBA IDL
Polo, Tendero e Piattini (2007)	Testooj	Java
Untch (1992), Untch (1997) e Untch, Offutt e Harrold (1993b)	TUMS	C
Devroey *et al.* (2016) e Kintis, Papadakis e Malevris (2010)	Vibes	Transition systems, Statechart models
Praphamontripong e Offutt (2010)	webMuJava	HTML/JSP
Bertolino *et al.* (2013)	XACMUT	XACML

Fonte: Elaborada pelo autor.

Devido a suas características, a ferramenta escolhida para a execução deste projeto foi a *Proteum* (*PRogram TEsting Using Mutants*). As principais características de ferramenta são destacadas a seguir (MACIEL, 2017):

- Condução do teste por meio de sessões de teste;
- Seleção dos operadores de mutação a serem gerados;
- Visualização da diferente do programa original para o programa mutante;
- Otimização de casos de teste;
- Interface visual.

2.4 Considerações finais

Nesse capítulo foram apresentados e discutidos conceitos fundamentais sobre o Teste de Mutação que é o objeto de estudo deste trabalho, da mesma forma, foram apresentados os problemas principais do Teste de Mutação e também possíveis soluções para os referidos problemas. Resumidamente, o Teste de Mutação é uma técnica comprovadamente com alta

eficácia na revelação de defeitos em um produto de software, e é uma ótima técnica para medir o nível de adequação de um determinado conjunto de casos de teste com relação ao programa. Apesar disso, ainda existem grandes desafios que precisam ser superados para que a técnica possa ser adotada em larga escala. Entre as principais dificuldades está o grande número de mutantes gerados e a dificuldade na identificação de mutantes equivalentes ao programa original, dificultando assim a aplicação da técnica em ambientes reais. No capítulo seguinte são apresentadas técnicas para identificação de mutantes minimais e equivalentes. O principal objetivo desta revisão bibliográfica é fornecer subsídios para os próximos capítulos desta dissertação.

CAPÍTULO

3

IDENTIFICAÇÃO DE MUTANTES MINIMAIS E EQUIVALENTES

3.1 Identificação de mutantes equivalentes

A identificação de mutantes equivalentes é reconhecidamente um problema indecidível, dessa forma, requer análise manual para apontamento dos mutantes que são equivalentes (BUDD; ANGLUIN, 1982). De acordo com Schuler e Zeller (2013) a identificação de um mutante equivalente leva em média 14 minutos e 30 segundos.

Levando em consideração os programas listados na Tabela 9, o percentual médio de mutantes equivalentes relativos ao total de mutantes de um programa, é de 8,25%, o que corresponde a 74 mutantes equivalentes por programa. Com base nestes dados, identificar todos os mutantes equivalentes de um programa com 32 linhas de código em média, pode levar em média 17 horas e 53 minutos. Dessa forma, a existência de mutantes equivalentes, é um dos fatores que agrava a não adoção do teste de mutação na indústria de desenvolvimento de software em programas de larga escala.

3.1.1 Técnicas para identificação de mutantes equivalentes

A identificação de mutantes equivalentes é um problema indecidível e é um fator negativo referente à utilização do Teste de Mutação. Diversas pesquisas têm sido realizadas com o intuito de aplicar técnicas e ferramentas que identifiquem ou facilitem a identificação de mutantes equivalentes de forma automática ou assistida.

Papadakis *et al.* (2015) propõem uma abordagem chamada “Trivial Compiler Equivalence” (TCE) que utiliza o código compilado de programas para identificar a equivalência entre mutantes. De acordo com os autores, pôde descartar aproximadamente 7% de todos os mutantes equivalentes.

Uma abordagem chamada "PredKillable" foi desenvolvida por Chekam *et al.* (2018) e tem o objetivo de predizer os mutantes que são mortais, isto é, aqueles que podem ser mortos através da execução de determinados casos de teste. Dessa forma, ao identificar os mutantes mortais, é possível também selecionar os mutantes que não são mortais e assim, classificá-los como equivalentes. De acordo com os autores, a abordagem obteve 98.7% de acurácia e 22% de revocação.

Através de uma abordagem diferente das abordagens supracitadas, VINCENZI *et al.* (2002) propuseram uma técnica nomeada como BaLBEDeT (Bayesian Learning-Based Equivalent Detection Technique) - Técnica para detecção de equivalentes baseada no aprendizado bayesiano. O aprendizado bayesiano dispõe de uma abordagem probabilística para inferência, sendo assim, essa abordagem foi posta em prática afim de detectar ou ajudar na detecção de mutantes equivalentes e não equivalentes.

Os trabalhos propostos por Grün, Schuler e Zeller (2009) Schuler e Zeller (2010) e Schuler e Zeller (2013) avaliam informações de cobertura de código gerada pela suíte de testes, afim de identificar mutantes equivalentes e não equivalentes. Schuler e Zeller (2010) fazem a análise dos mutantes com base nas informações de cobertura de testes e alteração de dados dos mutantes. A análise que levou em consideração os mutantes que geraram alterações nas informações de cobertura tiveram 75% de precisão e 56% de *recall* em identificar mutantes equivalentes, enquanto a análise dos mutantes que geraram alterações de dados, obteve 67% de precisão e 48% de *recall* em identificar mutantes equivalentes. Uma terceira análise, combinou a análise do impacto nas informações de cobertura de código e o impacto nos dados e assim obteve 70% de precisão e 61% de *recall*.

Schuler e Zeller (2013) evidenciam e confirmam a vasta existência de mutantes equivalentes em um dado conjunto de mutantes. Da mesma forma, o trabalho aplica uma abordagem para identificação de mutantes equivalentes que leva em consideração o percentual de cobertura de testes, isto é, os autores defendem que se uma mutação altera as informações de cobertura de testes, existe 75% de chance daquela mutação ser não equivalente. Sendo assim, os autores argumentam que a grande maioria dos mutantes equivalentes não alteram informações no que se refere à cobertura de código.

De modo semelhante aos trabalhos mencionados anteriormente, Schuler, Dallmeier e Zeller (2009) defendem que mutantes que violam invariações são mais propícios a serem encontrados pelo casos de teste, isto é, estes mutantes muito provavelmente são mutantes não equivalentes. Descrita por Ernst *et al.* (2001) as invariações dinâmicas são alterações no comportamento do programa e podem ser obtidas após a execução da versão original do programa e com todas suas métricas, dados e informações coletadas, compara-se todos estes artefatos com os artefatos gerados pela versão alterada do programa e assim, identifica-se as invariações dinâmicas. Sendo assim, o trabalho proposto por Schuler, Dallmeier e Zeller (2009) busca identificar os mutantes que violam invariações e assim, classifica-os como não equivalentes.

Além dos trabalhos supracitados, o trabalho desenvolvido por Madeyski *et al.* (2014) lista, categoriza e classifica diversas técnicas, abordagens e trabalhos relacionados à identificação de mutantes equivalentes.

3.2 Identificação de mutantes minimais

Kurtz *et al.* (2016) concluem que não existe uma forma de utilizar um conjunto de operadores de mutação e garantir que com aqueles operadores, haverá um baixo custo da análise de mutantes e um alto escore de mutação, portanto, cada programa deve ser analisado individualmente, ou seja, um dado conjunto de operadores de mutação qualquer que é excelente para um dado programa $p1$, não necessariamente será excelente também para um programa $p2$.

A Seção 2.2.2 define o conceito de conjunto de testes minimais e mostra como obter o conjunto de testes minimais de forma manual. O Algoritmo 1 demonstra de forma sistemática, como se pode obter o conjunto de testes minimais.

Algoritmo 1 – Minimização do conjunto adequado de testes

```
1: procedimento MINTESTSET(M, T)        ▷ Usa como entrada o conjunto de mutantes M e o conjunto de teste T
2:     minSet = T
3:     para todo t em minSet faça        ▷ t é selecionado arbitrariamente
4:         se ms(P, minSet - {t}) == ms(P, T) então        ▷ Verifica se ao retirar t o escore de mutação é mantido
5:             minSet = minSet − {t}
6:         fim se
7:     fim para
8:     retorna minSet        ▷ Obtém o conjunto adequado de testes
9: fim procedimento
```

Fonte: Ammann, Delamaro e Offutt (2014).

A Seção 2.2.2 define o conceito de conjunto de mutantes minimais e mostra como obter o conjunto de mutantes minimais de forma manual. O Algoritmo 2 demonstra de forma sistemática, como se pode obter o conjunto de mutantes minimais. Ainda que exista um algoritmo para a obtenção do conjunto mínino de mutantes, Kurtz *et al.* (2014) mencionam que a obtenção do conjunto de mutantes dominantes é um problema indecidível, ou seja, existe um algoritmo que implemente este problema, porém, pode ser que ele entre em um *loop* infinito e nunca pare.

Um conjunto de mutantes dominantes (denotado por D) é um subconjunto mínimo de um conjunto de mutantes (denotado por M), tal que, qualquer conjunto de teste que seja adequado para D, também é adequado para M (JUST; KURTZ; AMMANN, 2017).

Algoritmo 2 – Minimização do conjunto de mutantes

```
1: procedimento MINMUTANTSET(M, T) ▷ Usa como entrada o conjunto de mutantes M e a função de pontuação S
2:     S = S − mutantesVivos
3:     S = S − colunasDuplicadas
4:     minSet = S
5:
6:     dominados = mutantes_dinamicamente_dominados em minSet
7:
8:     retorna (minSet - dominados);          ▷ Obtém o conjunto de mutantes minimais
9: fim procedimento
```

Fonte: Ammann, Delamaro e Offutt (2014).

3.2.1 Técnicas para identificação de mutantes minimais

Diferentemente da identificação de mutantes equivalentes que já tem diversos trabalhos publicados, consolidados e com pesquisa desde o início do século XXI, a identificação de mutantes minimais ainda é um campo com um vasto espaço para exploração tendo em vista que essa técnica por muito tempo foi preterida pelas técnicas *do fewer*, *do smarter* e *do faster*. O trabalho desenvolvido por Pizzoleto *et al.* (2019) é uma revisão sistemática da literatura afim de identificar, mapear e classificar diversas técnicas para redução do custo do teste de mutação. Entre as 21 categorias listadas no trabalho, estava também a técnica de mutação minimal. Entre os trabalhos citados, estão Ammann, Delamaro e Offutt (2014), Kurtz *et al.* (2016) e Just, Kurtz e Ammann (2017).

Os trabalhos desenvolvidos por Ammann, Delamaro e Offutt (2014) e Kurtz *et al.* (2016) definem e teorizam o conceito de mutantes minimais, além de trazer conceitos relacionados à identificação de mutantes minimais, como supramencionado na Subseção 2.2.2.

Just, Kurtz e Ammann (2017) classificam a utilidade de um mutante entre equivalente, trivial ou dominante. Os mutantes equivalentes são os mutantes supracitados no presente trabalho, os mutantes triviais são todos os mutantes que não fazem diferença no escore de mutação e que não afetam o resultado da mutação, enquanto os mutantes dominantes são os mutantes que compreendem todos os mutantes triviais e que são indispensáveis para a manutenção do escore de mutação. Para derivar informações e construir uma base para predizer a classificação dos mutantes, os autores utilizaram diversas informações do contexto do mutante, bem como informações contextuais dos nós anteriores (nó pai) na árvore de sintaxe abstrata, contexto dos nós inferiores (nó filho) e tipo de dado. Dessa forma, a partir de todas essas informações do contexto, a abordagem proposta pelos autores consegue classificar o mutante de acordo com as categorias anteriormente mencionadas e assim executar de forma mais performática o teste de mutação.

CAPÍTULO

4

FUNDAMENTOS SOBRE APRENDIZADO DE MÁQUINA

4.1 Considerações iniciais

A resolução de problemas por meio da aplicação do aprendizado de máquina permite a autoaprendizagem para construção de conhecimento baseado em dados, permitindo a realização de predições de dados ainda desconhecidos com base nos dados históricos. Desta forma, o aprendizado de máquina pode substituir a necessidade de intervenção humana em problemas computacionais indecidíveis, fazendo com que decisões humanas possam ser assistidas pelos algoritmos de aprendizado de máquina e em alguns casos, ainda é possível tomar as decisões com base nas predições realizadas pelos modelos preditivos, tais quais foram criados pelo conhecimento fornecido pelos dados históricos (RASCHKA, 2015).

O primeiro algoritmo de aprendizado de máquina surgiu na década de 1970, entretanto, naquela época a utilização desse algoritmo e de técnicas de aprendizado de máquina, ainda não era algo comum devido ao poder computacional baixo daquela época. Com o passar do tempo e principalmente com o aumento de poder computacional, o aprendizado de máquina têm desenvolvido um importante papel na ciência da computação, permitindo que seja possível utilizar aprendizado de máquina para resolver problemas cada vez mais complexos (LOURIDAS; EBERT, 2016).

O aprendizado de máquina tem sido utilizado em diversos domínios diferentes, entre eles:

- Heurísticas de segurança que distinguem padrões de ataque;
- Análise de imagens para identificação de formas e padrões;

- Aprendizado profundo para derivação de regras para análise de dados e manipulação de grandes massas de dados;
- Reconhecimento de padrões e predições a partir de combinadas saídas de vídeos;
- Reconhecimento de padrões para análise de código e identificação de códigos frágeis e *code smells*.

4.2 Tipos de aprendizado

Essencialmente, os algoritmos de aprendizado de máquina têm comportamentos semelhantes, com distinção apenas nos modelos de predição que utilizam. Entre os tipos de aprendizado de máquina, os mais utilizados são o aprendizado supervisionado e o aprendizado não supervisionado. O presente trabalho aplica o aprendizado supervisionado.

4.2.1 Aprendizado supervisionado

De acordo com Durelli *et al.* (2019), o aprendizado supervisionado é o tipo de aprendizado de máquina mais utilizado para resolver problemas de testes de software. O aprendizado supervisionado tem como objetivo, aprender de acordo com os dados, isto é, realizar predições do futuro (informações ainda desconhecidas) baseada no dados do passado (informações já conhecidas). No aprendizado supervisionado, os dados que servem de entrada para o desenvolvimento dos modelos preditivos contém todas as informações necessárias para predição dos novos dados, tanto as informações básicas, bem como a resposta, que é a informação que os modelos preditivos irão predizer nos novos dados que forem inseridos.

Entre os algoritmos de aprendizado supervisionado, os principais são algoritmos de classificação e regressão (LOURIDAS; EBERT, 2016). Os algoritmos de classificação têm como principal característica receber um determinado conjunto de dados e com esse conjunto de dados, o algoritmo irá identificar padrões e a partir disso, poderá classificar os valores de um atributo de uma nova entidade com base no aprendizado desenvolvido a partir dos dados de treinamento. Os principais algoritmos de classificação são: regressão lógica, árvores de classificação, máquinas de vetores de suporte, florestas aleatórias e redes neurais artificiais. Enquanto os algoritmos de classificação têm como objetivo classificar os valores de um atributo de novas entidades, os algoritmos de regressão têm como objetivo prever novos valores para novas entidades. Os principais algoritmos de regressão são: regressão linear, árvores de decisão, redes bayesianas, classificação *fuzzy* e redes neurais artificiais.

Exemplos:

- Algoritmos de classificação: Análise de e-mails para identificação de spam. Com base em uma determinada massa de dados, no caso, e-mails, o algoritmo poderá identificar os

e-mails que podem ser spam. Exemplos de dados: Sim, não.

- Algoritmos de regressão: Predição da temperatura dos dias seguintes. Com base em uma determinada massa de dados, no caso, informações meteorológicas, os algoritmos podem predizer a temperatura do dia de amanhã. Exemplos de dados: 15º C, 23º C, 30º C, etc.

A Figura 4 exemplifica a utilização de algoritmos de aprendizado supervisionado no contexto deste trabalho.

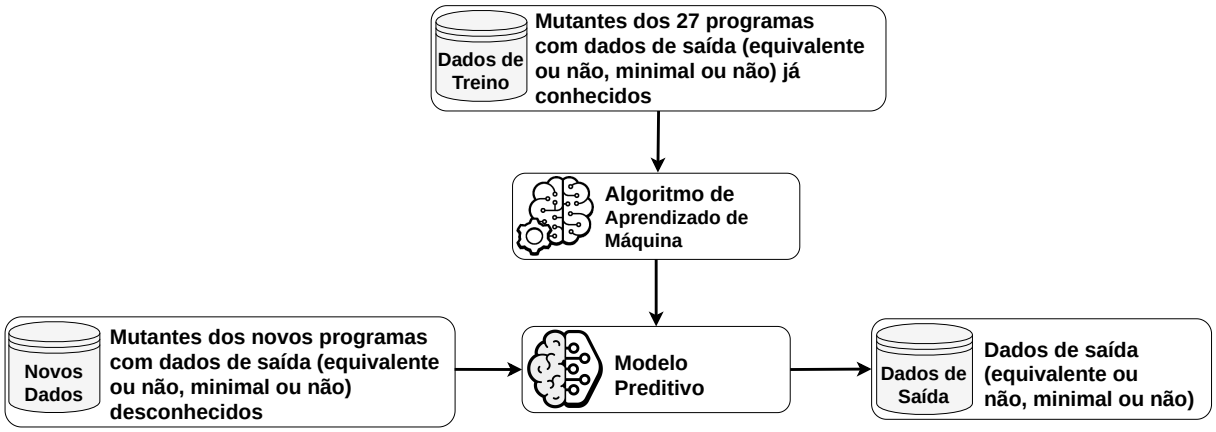


Figura 4 – Aprendizado Supervisionado

4.2.2 Aprendizado não supervisionado

Algoritmos de aprendizado não supervisionado tem a capacidade de aprender e organizar informações, mesmo sem os dados completos para o treinamento. O aprendizado não supervisionado consegue identificar e detectar padrões a partir dos dados e então a partir disso, consegue replicar estes padrões identificados para novas instâncias de dados.

A principal diferença do aprendizado supervisionado e do aprendizado não supervisionado, é que no aprendizado não supervisionado, não são todas as informações que são utilizadas como entrada para desenvolvimento dos modelos preditivos, isto é, no aprendizado não supervisionado, o valor do atributo desejado para as instâncias de dados já conhecidas, não é fornecido. Neste caso, o próprio modelo preditivo tenta encontrar o valor do atributo desejado para os novos dados.

4.3 Modelos de classificação

Nesta subseção, são listados os modelos de classificação (aqui chamados de algoritmos) utilizados neste trabalho. A Tabela 5 lista todos os algoritmos utilizados neste trabalho, bem como as siglas referentes à cada um e que serão citadas adiante e também as parametrizações realizadas em cada um dos algoritmos.

Tabela 5 – Algoritmos classificadores utilizados neste trabalho

Algoritmo		**Parâmetros**
K Nearest Neighbors	KNN	*n_neighbors* = De 1 até 40, pulando de 1 em 1
Decision Tree	DT	*min_samples_split* = De 5 até 95, pulando de 10 em 10
Random Forest	RF	*min_samples_split* = De 5 até 95, pulando de 10 em 10
Support Vector Machine	SVM	
Gaussian Naive Bayes	GNB	
Linear Discriminant Analysis	LDA	
Logistic Regression	LR	

K Nearest Neighbors

Os algoritmos *K Nearest Neighbors* (KNN), ou K vizinhos mais próximos, são considerados os algoritmos mais simples de serem utilizados, dessa forma, comumente são utilizados como ponto de partida ao resolver problemas utilizando aprendizado de máquina. O propósito do algoritmo é memorizar o conjunto de treino e a partir disso, predizer a resposta das novas instâncias de dados com base nas resposta dos vizinhos mais próximos, levando em consideração o conjunto de treinamento (SHALEV-SHWARTZ; BEN-DAVID, 2014). O "K" do algoritmo, denota o número de vizinhos próximos que serão levados em conta para classificar o resultado de novas instâncias. Neste trabalho, o "K" (ou seja, o número de vizinhos) utilizado foi de **1** até **40**.

Para realizar a classificação de uma nova instância de um dado, o KNN parte do princípio que objetos relacionados ao mesmo conceito são semelhantes entre si, dessa forma, ao classificar um novo dado, busca pelos dados existentes mais próximos do novo dado inserido.

Decision Tree

Os algoritmos *Decision Tree* (DT), ou Árvores de decisão, funcionam basicamente como um preditor binário, de forma a predizer o resultado de novas instâncias de dados a partir da navegação do nó origem de uma árvore até a folha, onde cada folha da referida árvore contém uma instância possível do resultado. A navegação do nó inicial até a folha, acontece por meio da escolha binária do ramo que será utilizado para navegar até que se chegue à folha daquela árvore. Para chegar até as folhas da árvore e realizar a predição, este algoritmo leva em consideração um conjunto predefinido de regras de derivação, onde uma dessas regras é o número mínimo de amostras utilizadas para derivar um nó da árvore.

Como o próprio nome sugere, o algoritmo Árvore de Decisão gera um fluxograma na estrutura de uma árvore para estruturar as predições resultantes de uma série de divisões realizadas com base nas propriedades dos dados selecionados, isto é, a partir dos dados de treinamento são derivadas árvores de decisão. A aplicação do algoritmo se inicia com um nó raiz e a partir de cada nó, uma decisão é tomada e assim acontece recursivamente até o nó final da árvore que irá ser o resultado da predição. No presente trabalho, o número mínimo de amostras

para derivar um nó da árvore, foi de **5** até **95**, interpolando de 10 em 10, isto é, os valores foram 5, 15, 25, 35, 45, 55, 65, 75, 85 e 95.

Random Forest

O algoritmo *Random Forest* (RF), ou Floresta aleatória, é uma derivação das árvores de decisão. Utilizada primariamente com o intuito de restringir o tamanho máximo das árvores de decisão e o risco do *overfitting*, para isso, a floresta aleatória cria um conjunto de novas árvores de decisão. Uma floresta aleatória é um classificador que consiste no conjunto de diversas árvores de decisão, onde cada árvore de decisão foi criada a partir da aplicação da um algoritmo no conjunto de dados de treinamento e um vetor aleatório adicional.

O algoritmo Floresta Aleatória é da classe de algoritmos de *Ensemble Learning* (aprendizado conjunto), dessa forma, para a construção de um modelo preditivo com o algoritmo Floresta Aleatória, são combinados diversos outros modelos preditivos visando obter uma eficácia maior na classificação de novos dados. No presente trabalho, os parâmetros utilizados para a criação das florestas aleatórias são os mesmos parâmetros utilizados para a criação das árvores de decisão.

Support Vector Machine

O algoritmo *Support Vector Machine* (SVM), ou Máquina de suporte vetorial é essencialmente utilizado para aprendizado e realização de predições lineares em largos espaços vetoriais (SHALEV-SHWARTZ; BEN-DAVID, 2014), mas também pode ser utilizada para problemas de classificação. O SVM realiza a divisão de um conjunto de dados com diferentes classes, isto é, por meio do conceito de planos de decisão, as novas instâncias de dados de uma determinada resposta, são classificadas separadamente das novas instâncias de dados de outra determinada resposta.

O algoritmo SVM toma como dados de entrada pontos no espaço e inicia-se a classificação separando através de um espaço as instâncias de dados para uma categoria ou outra. Para a classificação de novos dados, verifica-se à qual categoria refere-se o espaço onde aquele novo dado foi inserido. Sendo assim, o SVM define um hiperplano entre instâncias de dados de duas categorias diferentes e procura distanciar o máximo possível os pontos mais próximos de cada uma das categorias. No presente trabalho, este algoritmo foi utilizado com suas configurações e parâmetros padrões.

Gaussian Naive Bayes

O algoritmo *Naïve Bayes* (GNB), é um algoritmo classificador probabilístico que tem sua aplicação e funcionamento baseados no teorema de Bayes. A aplicação deste algoritmo se dá principalmente quando há uma alta dimensionalidade do conjunto de dados, isto é, um vasto número de características e propriedades das instâncias de dados.

Este algoritmo parte do pressuposto que existe uma forte independência entre as propriedades dos dados, isto é, que existe uma baixa correlação entre as características do conjunto de dados. No presente trabalho, este algoritmo foi utilizado com suas configurações e parâmetros padrões.

Linear Discriminant Analysis

O algoritmo *Linear Discriminant Analysis* (LDA), ou Análise Discriminante Linear é um modelo de predição que tem como objetivo principal a redução da dimensionalidade do problema e classificação de novas instâncias de dados com base nas informações de instâncias de dados já conhecidas.

Este modelo pode obter bons resultados na classificação de problemas lineares, porém, não traz bons resultados na classificação de problemas multi-classes. Este algoritmo consiste em buscar uma projeção no hiperplano que minimize a interseção de diferentes categorias e maximizar a distância entre pontos das categorias. No presente trabalho, este algoritmo foi utilizado com suas configurações e parâmetros padrões.

Logistic Regression

O algoritmo *Logistic Regression* (LR), ou Regressão Logística, é uma técnica estatística utilizada com o objetivo primário de predizer resultados categóricos, principalmente, valores binários, isto é, classificar novos valores entre classes binárias. O resultado de uma regressão logística não é apenas a indicação binária de um dado em uma determinada classe, mas também, o percentual de probabilidade daquela classe em detrimento da outra classe analisada. No presente trabalho, este algoritmo foi utilizado com suas configurações e parâmetros padrões.

4.4 Análise dos resultados

A avaliação dos modelos preditivos pode ser realizada pelo emprego de diversas métricas, porém, as métricas que são mais amplamente adotadas e que serão utilizadas neste trabalho são: Acurácia, Precisão, Revocação e F1-Score. O conceito sobre cada uma das métricas é conforme segue abaixo (HULTEN, 2018):

- **Acurácia**: A acurácia é definida como a proporção de todos os resultados classificados corretamente, sejam eles positivos ou negativos;
- **Precisão**: A precisão é definida como a fração de todos os resultados do modelos que foram classificados como positivo e realmente são positivos;
- **Revocação**: A revocação é definida como a proporção de todos os resultados que realmente são positivos e o modelo classifica como positivo;

- **F1-Score**: O *F1-Score* é á média harmônica da precisão e da revocação.

Para entender como funciona o cálculo de cada um das referidas métricas, é necessário entender um outro conceito primeiramente: matriz de confusão. A matriz de confusão separa as predições conforme mostrado abaixo:

- *True Positives* (TP), Verdadeiros Positivos: Predições realizadas como positivas e realmente são positivas;
- *False Negatives* (FN), Falsos Negativos: Predições realizadas como negativas, mas na realidade são positivas;
- *False Positives* (FP), Falsos Positivos: Predições realizadas como positivas, mas na realidade são negativas;
- *True Negatives* (TN), Verdadeiros Negativos: Predições realizadas como negativas e realmente são negativas.

A maneira de se calcular cada uma das métricas é exibida abaixo.

Acurácia =
$$\frac{(TP+TN)}{(FP+FN+TP+TN)}$$

Precisão =
$$\frac{TP}{(TP+FP)}$$

Revocação =
$$\frac{TP}{(FN+TP)}$$

F1-Score =
$$2 \times \frac{(Precisao \times Revocacao)}{(Precisao + Revocacao)}$$

Por englobar duas importantes métricas (precisão e revocação), neste trabalho, foi adotado o *F1-Score* como principal métrica de avaliação dos modelos preditivos.

4.4.1 Exemplos

Métricas coletadas a partir da matriz de confusão:

- Acurácia =
$$\frac{(20+65)}{(5+10+20+65)} = 0,85$$
- Precisão =
$$\frac{20}{(20+5)} = 0,80$$

Tabela 6 – Exemplo descritivo da matriz de confusão

Métrica	**Descrição**
TP - Verdadeiro positivo	Dos **25** mutantes equivalentes, **20** foram classificados corretamente como mutantes equivalentes
FP - Falso positivo	**5** mutantes equivalentes foram classificados incorretamente como não equivalentes
TN - Verdadeiro negativo	Dos **75** mutantes não equivalentes, **65** foram classificados corretamente como não equivalentes
FN - Falso negativo	**10** mutantes não equivalentes foram classificados incorretamente como equivalentes

Tabela 7 – Matriz de confusão

TP	FP
FN	TN

20	5
10	65

- Revocação =
$$\frac{20}{(10+20)} = 0,6666666666666667$$
- F1-Score =
$$2 \times \frac{(0,80 \times 0,66)}{(0,80+0,66)} = 0,7272727273$$

4.5 Considerações finais

Nesta seção, foram descritos os fundamentos básicos de aprendizado de máquina. Foram apresentados os tipos de aprendizado de máquina, sendo eles, aprendizado supervisionado e aprendizado não supervisionado, entretanto, neste trabalho, foi adotado o aprendizado supervisionado. Foram também descritos os modelos preditivos utilizados neste trabalho, sendo eles, o *K Nearest Neighbors*, *Decision Tree*, *Random Forest*, *Support Vector Machine*, *Gaussian Naive Bayes*, *Linear Discriminant Analysis* e *Logistic Regression*. Na parte final dessa seção, foram descritas as principais métricas aqui utilizadas para avaliar os modelos preditivos, bem como a forma de calcular cada uma das métricas. As métricas descritas foram, Acurácia, Precisão, Revocação e F1-Score.

CAPÍTULO

5

UMA ABORDAGEM PARA IDENTIFICAÇÃO DE MUTANTES MINIMAIS E EQUIVALENTES

5.1 Considerações iniciais

A abordagem proposta neste trabalho foi motivada por um estudo anterior proposto por Delamaro, Chaim e Maldonado (2018), o qual investiga a possibilidade de relacionar o local da mutação no Grafo de Fluxo de Controle (GFC) e sua classificação como minimal ou não. A presente abordagem estende o trabalho supracitado em dois pontos principais: (i) foram incluídas outras propriedades dos mutantes que podem influenciar na sua classificação como minimal ou não, e (ii) foi decidido aplicar a mesma abordagem para identificar mutantes equivalentes ou não.

O objetivo deste trabalho é reduzir o custo da aplicação do teste de mutação, e para isso, foram aplicadas técnicas de aprendizado de máquina. A partir dos dados e propriedades dos mutantes de 27 programas (dos quais já se sabe a classificação como minimal ou não, equivalente ou não de todos os mutantes), foram criados modelos preditivos que preveem a classificação de um mutante como minimal ou não, equivalente ou não, sem executar este mutante, apenas com base nas suas propriedades estruturais, ou mais precisamente, pela análise do ponto do programa onde a mutação foi realizada. Neste capítulo, é descrito detalhadamente o processo de desenvolvimento e aplicação da abordagem.

5.2 Planejamento

Esta seção apresenta uma descrição detalhada do experimento, bem como os sujeitos e variáveis utilizadas para executá-lo. O objetivo principal do experimento é verificar a relação entre as propriedades dos mutantes e sua classificação como minimal ou não, equivalente ou não.

5.2.1 Design do Experimento

No experimento, foi utilizado o seguinte:

- **Sujeitos:** 27 programas e 49 funções escritas na linguagem *C* (Todos os programas e suas informações estão disponíveis na Tabela 9);
- **Variáveis independentes:** os operadores de mutação propostos para a linguagem *C* (RICHARD *et al.*, 1989), o conjunto adequado de teste, os algoritmos de aprendizado de máquina e os parâmetros utilizados para os algoritmos;
- **Variáveis dependentes:** As métricas de aprendizado de máquina para cada modelo preditivo.

5.2.2 Questões de Pesquisa

Especificamente, nós pretendemos investigar as seguintes questões de pesquisa:

- **QP$_1$** : *É possível identificar um mutante minimal de acordo com suas características estruturais?*
- **QP$_2$** : *É possível identificar um mutante equivalente de acordo com suas características estruturais?*
- **QP$_3$** : *Existe diferença significativa nas métricas entre os modelos preditivos para classificar mutantes minimais?*
- **QP$_4$** : *Existe diferença significativa nas métricas entre os modelos preditivos para classificar mutantes equivalentes?*

5.2.3 Definição dos objetivos

Foi utilizado o modelo proposto por Wohlin *et al.* (2012) para aplicação do método *Goal/Question/Metric* (GQM) (BASILI; ROMBACH, 1988) . O modelo para aplicação do GQM apresenta o experimento dividido em cinco partes: objeto de estudo, propósito, perspectiva, foco qualitativo e contexto.

- **Objetos de estudo**: Os objetos de estudo são os modelos preditivos desenvolvidos;
- **Propósito**: O propósito deste experimento é avaliar a possibilidade de redução do custo do teste de mutação por meio da aplicação de aprendizado de máquina;
- **Perspectiva**: Este experimento é executado do ponto de vista de um pesquisador;

- **Foco qualitativo**: O foco do experimento é avaliar as métricas de aprendizado de máquina apresentada para cada modelo preditivo;
- **Contexto**: Este experimento foi conduzido utilizando a ferramenta de Teste de Mutação Proteum[1]. Para a classificação dos mutantes e análises, foram utilizados algoritmos de aprendizado de máquina já implementados em Python. Como a implementação é uma prova de conceito acadêmica, os programas utilizados como objetos experimentais deste experimento não possuem uma significância industrial. O código fonte dos programas utilizados foi extraído de livros sobre Engenharia de Software e os resultados aqui apresentados se destinam exclusivamente para ambientes acadêmicos.

Assim, o experimento realizado neste trabalho pode utilizar o modelo proposto por Wohlin *et al.* (2012) e é resumido da seguinte forma:

Analisar algoritmos de aprendizado de máquina

com o propósito de avaliar a possibilidade de reduzir o custo do teste de mutação através do aprendizado de máquina

no que diz respeito à métricas de aprendizado de máquina

do ponto de vista de um pesquisador

no contexto de um conjunto de programas para a linguagem C, extraídos de livros de Engenharia de Software.

5.2.4 Formulação de hipóteses

A seguir, as questões de pesquisa (QP_3 e QP_4) apresentadas no início desta seção são convertidas em hipóteses para que sejam realizados testes estatísticos.

Para a QP_3 foram definidas as seguintes hipóteses:

Hipótese Nula, H_0: Não há diferença significativa nas métricas entre os modelos preditivos para classificar mutantes minimais.

$$KNN \equiv DT \equiv RF \equiv SVM \equiv LDA \equiv LR \equiv GNB$$

Hipótese Alternativa, H_1: Há uma diferença significativa nas métricas entre os modelos preditivos para classificar mutantes minimais.

$$KNN \neq DT \neq RF \neq SVM \neq LDA \neq LR \neq GNB$$

[1] Link para download: <https://github.com/claudineibjr/ClassifyingMutants/blob/master/Proteum/ProteumIM.7z>

Para a QP_4 foram definidas as seguintes hipóteses:

Hipótese Nula, H_0: Não há diferença significativa nas métricas entre os modelos preditivos para classificar mutantes equivalentes.

$$KNN \equiv DT \equiv RF \equiv SVM \equiv LDA \equiv LR \equiv GNB$$

Hipótese Alternativa, H_1: Há uma diferença significativa nas métricas entre os modelos preditivos para classificar mutantes equivalentes.

$$KNN \neq DT \neq RF \neq SVM \neq LDA \neq LR \neq GNB$$

5.3 Geração dos dados dos mutantes

Delamaro, Chaim e Maldonado (2018) sugerem que os mutantes minimais podem estar relacionados à: (i) distância entre o nó do GFC que ocorreu a mutação com os nós iniciais e finais do GFC; e (ii) Pertencimento ou não do nó do GFC aos arcos primitivos definidos por Chusho (1987). A partir desta hipótese, este trabalho estende a ideia de procurar pelos mutantes minimais a partir das características acima mencionadas e também pelos mutantes equivalentes. Além das propriedades mencionadas, a presente abordagem reúne algumas outras propriedades dos mutantes e assim, constroem-se os modelos preditivos. Listadas abaixo, estão todas as propriedades que são utilizadas para construção dos modelos preditivos.

- **OPERATOR**: *String* indicando o operador de mutação do mutante. Os valores possíveis para esta propriedades são todos os operadores de mutação da linguagem C disponíveis na ferramenta Proteum;
- **SOURCE_PRIMITIVE_ARC**: Número (0 ou 1) indicando se o mutante é o nó de origem de um arco primitivo;
- **TARGET_PRIMITIVE_ARC**: Número (0 ou 1) indicando se o mutante é o nó de destino de um arco primitivo;
- **DISTANCE_BEGIN_MIN**: Número indicando a distância mínima do nó onde ocorreu a mutação até o nó inicial do GFC;
- **DISTANCE_BEGIN_MAX**: Número indicando a distância máxima do nó onde ocorreu a mutação até o nó inicial do GFC;
- **DISTANCE_BEGIN_AVG**: Número indicando a distância média do nó onde ocorreu a mutação até o nó inicial do GFC;

- **DISTANCE_END_MIN**: Número indicando a distância mínima do nó onde ocorreu a mutação até o nós finais do GFC;
- **DISTANCE_END_MAX**: Número indicando a distância máxima do nó onde ocorreu a mutação até o nós finais do GFC;
- **DISTANCE_END_AVG**: Número indicando a distância média do nó onde ocorreu a mutação até o nós finais do GFC;
- **COMPLEXITY**: Número indicando o número de mutações que ocorreram no nó do GFC onde a mutação ocorreu;
- **TYPE_STATEMENT**: *String* indicando o tipo de instrução que foi alterado pela mutação. Os valores possíveis são “Block”, “While”, “For”, “If”, “Assignment”, “Return” ou “Function Call”.

Afim de exemplificar os dados gerados pela Proteum e coletados para futura criação dos modelos preditivos,a Tabela 8 mostra as propriedades e valores do mutante ID *163* (mesmo mutante exemplificado na Figura 2) do programa *CheckPalindrome*.

Tabela 8 – Propriedades do mutante #163 do programa CheckPalindrome

Propriedade	**Valor**
OPERATOR	u-VVDL
SOURCE_PRIMITIVE_ARC	1
TARGET_PRIMITIVE_ARC	0
DISTANCE_BEGIN_MIN	1
DISTANCE_BEGIN_MAX	4
DISTANCE_BEGIN_AVG	2.5
DISTANCE_END_MIN	2
DISTANCE_END_MAX	6
DISTANCE_END_AVG	4.0
COMPLEXITY	48
TYPE_STATEMENT	While

A Figura 5 exibe o Mutante *163* do programa CheckPalindrome na ferramenta de teste de mutação “PROTEUM - PROgram TEsting Using Mutants - V 2.1”. O mutante exibido na Figura 5, é o mesmo mutante exibido na Figura 2 e Tabela 8.

5.4 Visão geral do desenvolvimento da abordagem

O objetivo da abordagem é verificar a relação entre as propriedades estruturais dos mutantes e sua classificação como minimal ou não, equivalente ou não. Para o desenvolvimento da abordagem, foram utilizados como entrada 27 programas, 49 funções e o conjunto adequado

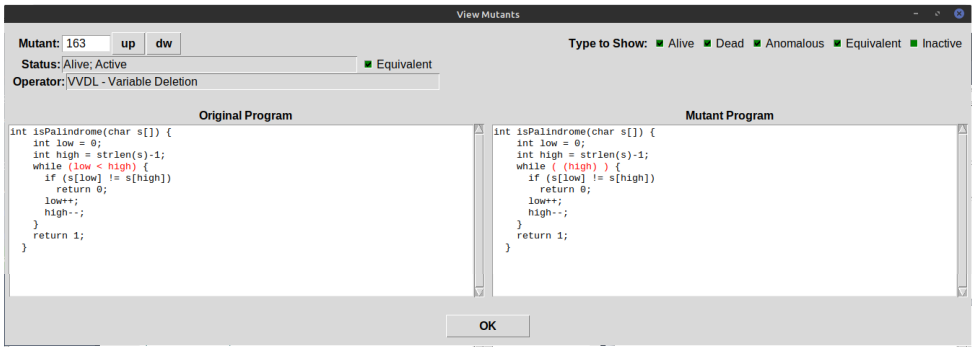


Figura 5 – Visualização do mutante *163* do programa CheckPalindrome na ProteumIM

de teste para cada programa. Para a geração e análise dos mutantes, foi utilizada a ferramenta Proteum (**Pro**gram **Te**sting **U**sing **M**utants) (DELAMARO; MALDONADO, 1996).

A seguir, são descritos os passos executados para o desenvolvimento da abordagem:

1. Para cada função F_1,F_2,...,F_{49}, todos os operadores de mutação da ferramenta Proteum foram utilizados para gerar mutantes;

2. Após a geração dos mutantes, todos eles foram executados contra o conjunto adequado de testes;

3. Para cada função F_i, o conjunto de mutantes minimais foi calculado com respeito ao conjunto adequado de teste;

4. Para cada função F_i, o conjunto de mutantes equivalentes foi identificado (todos os mutantes que sobreviveram à execução do conjunto adequado de testes é identificado como um mutante equivalente);

5. Para cada função F_i, foram gerados os GFCs;

6. Para cada GFC gerado, foram identificados os arcos primitivos (CHUSHO, 1987);

7. Os dados coletados de todos os mutantes de todas as funções foram reunidos em um único conjunto de dados;

8. Foram aplicadas técnicas de aprendizado de máquina para pré processar os dados gerados [2];

9. Foram escolhidos algoritmos de Aprendizado de Máquina (os algoritmos selecionados para este trabalho são listados na Tabela 5) para criação dos modelos preditivos com base nos dados;

[2] Esta etapa é detalhada na Seção 5.5

10. Para cada classificador e cada coluna a ser classificada (minimal ou equivalente), foi construído um modelo preditivo. Os dados gerados nos passos anteriores servem como entrada para a criação dos modelos preditivos;

11. Os modelos preditivos são treinados e depois testados;

12. São calculadas as métricas (Acurácia, Precisão, Revocação e F1-Score) de aprendizado de máquina que são utilizadas para guiar o processo de treino e teste dos modelos preditivos.

A Figura 6 sumariza o processo de desenvolvimento da abordagem.

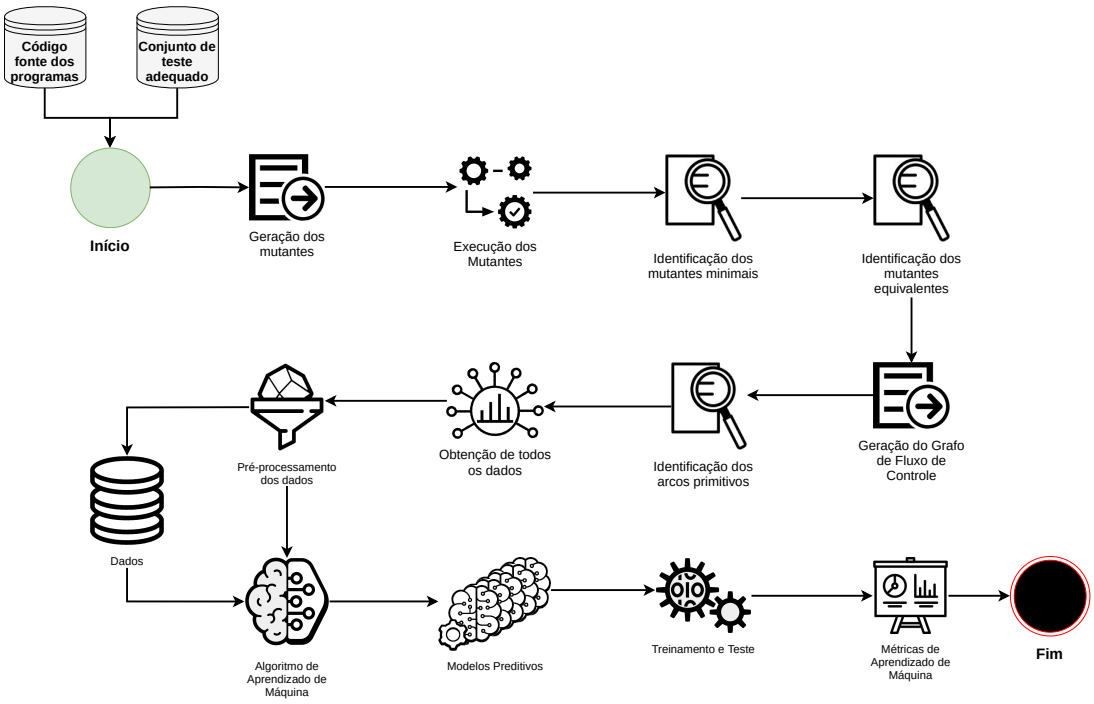


Figura 6 – Passos executados para o desenvolvimento da abordagem

Os 27 programas utilizados para o desenvolvimento da abordagem são exibidos na Tabela 9. Para cada programa, a tabela apresenta o número de funções (Fun), o número de linhas de código (LOC), o número de mutantes (Mut), o número de mutantes minimais (Min) e o percentual dos mutantes minimais referente ao total de mutantes, o número de mutantes equivalentes (Equi) e o percentual dos mutantes equivalentes referente ao total de mutantes e o número de casos de teste (TC).

O código fonte dos programas utilizados no experimento foram fornecidos pelo orientador dessa dissertação. A maioria dos programas já foram utilizados em outros experimentos, por exemplo: Delamaro, Offutt e Ammann (2014), Delamaro *et al.* (2014), Ammann, Delamaro e Offutt (2014) e Andrade *et al.* (2019). Os conjuntos adequados de teste utilizados neste experimento também foram utilizados nos experimentos anteriores.

O funcionamento da abordagem se difere em duas etapas. A primeira etapa, de desenvolvimento da abordagem, que é quando os modelos preditivos são criados, treinados e

Tabela 9 – Programas utilizados no experimento

Program	Fun	LOC	Mut	Min	%	Equi	%	TC
cal	1	22	891	133	14.93	77	8.64	8
Calculation	5	44	928	74	7.97	86	9.27	13
checkIt	1	11	104	32	30.77	3	2.88	9
CheckPalindrome	1	11	166	13	7.83	20	12.05	8
countPositive	1	11	151	8	5.3	9	5.96	6
date-plus	2	141	2412	320	13.27	170	7.05	49
DigitReverser	1	19	496	7	1.41	43	8.67	5
findLast	1	12	198	32	16.16	8	4.04	6
findVal	1	9	190	9	4.74	10	5.26	7
Heap	6	38	931	25	2.69	95	10.2	8
InversePermutation	1	18	576	38	6.6	60	10.42	12
jday-jdate	2	45	2821	117	4.15	93	3.3	27
lastZero	1	11	173	9	5.2	9	5.2	5
LRS	3	45	865	126	14.57	139	16.07	8
MergeSort	3	36	991	28	2.83	33	3.33	18
numZero	1	12	151	6	3.97	10	6.62	5
oddOrPos	1	11	361	8	2.22	70	19.39	7
power	1	13	268	9	3.36	6	2.24	9
printPrimes	2	39	715	23	3.22	72	10.07	7
quicksort	1	24	1026	12	1.17	82	7.99	14
RecursiveSelectionSort	1	18	555	14	2.52	35	6.31	8
sum	1	9	165	10	6.06	11	6.67	6
tcas	3	56	1948	90	4.62	414	21.25	62
testPad	1	26	629	54	8.59	57	9.06	14
trashAndTakeOut	2	23	599	94	15.69	27	4.51	12
twoPred	1	12	246	16	6.5	24	9.76	10
UnixCal	4	142	4855	354	7.29	342	7.04	29
Mínimo	**1**	**9**	**104**	**6**	**1.17**	**3**	**2.24**	**5**
Máximo	**6**	**142**	**4855**	**354**	**30.77**	**414**	**21.25**	**62**
Média	**1.81**	**31.77**	**867.07**	**61.51**	**7.54**	**74.25**	**8.26**	**13.77**
Total	**6**	**142**	**4855**	**354**	**30.77**	**414**	**21.25**	**62**

aperfeiçoados, e a segunda etapa que é quando os modelos preditivos criados na etapa anterior são utilizados para classificar novos mutantes como minimis ou não, equivalentes ou não.

5.5 Aplicação do Aprendizado de Máquina

Após a execução dos passos anteriormente citados, todos os dados dos mutantes já foram gerados, e portanto, é necessário gerar conhecimento a partir daqueles dados e é nesta etapa que aplica-se o aprendizado de máquina. Nas subseções subsequentes, são descritos os passos que são aplicados para obtenção e aplicação do conhecimento adquirido a partir dos dados dos mutantes gerados.

5.5.1 Preparação dos dados

Após os mutantes terem sido gerados e executados na Proteum, todas as propriedades dos mutantes mencionadas na Seção 5.3 são coletadas, porém, para uma melhor eficácia e funcionamento dos modelos preditivos, é necessário uma etapa de pré-processamento dos dados encontrados.

As propriedades *DISTANCE_BEGIN_MIN*, *DISTANCE_BEGIN_MAX*, *DISTANCE_BEGIN_AVG*, *DISTANCE_END_MIN*, *DISTANCE_END_MAX*, *DISTANCE_END_AVG* e *COMPLEXITY* normalmente assumem um número inteiro maior do que 0, e portanto, pode gerar pequenos problemas ao servirem de entrada para os modelos preditivos, pois devido a diversidade dos programas, os valores mínimos e máximos para essas propriedades pode variar, dessa forma, é necessário normalizar esses valores.

A segunda parte do pré-processamento necessária para a criação dos modelos preditivos, é a transformação dos valores que são texto (dados categóricos) em números, neste caso, é necessário ajustar as propriedades *OPERATOR* e *TYPE_STATEMENT*. Uma solução comumente utilizada para resolver este problema, é adotar a técnica nomeada *One-Hot Encoding* (RASCHKA, 2015). A ideia por trás desta técnica é criar uma nova propriedade para cada possível valor da propriedade, e indicar com 1 a propriedade que corresponde ao valor correto e indicar com 0 as propriedades que não correspondem ao verdadeiro valor.

Afim de exemplificar o processo de preparação dos dados para criação dos modelos preditivos, a Tabela 10 mostra as propriedades e valores do mutante ID *163* (mesmo mutante exemplificado na Figura 2 e Tabela 8) do programa *CheckPalindrome*.

5.5.2 Seleção dos atributos

Entre as ferramentas de Aprendizado de Máquina disponíveis para aperfeiçoamento dos modelos preditivos, uma delas é a seleção de atributos (*Feature Selection*). Essa ferramenta tem como objetivo a redução da dimensionalidade do conjunto inicial de atributos, fazendo o apontamento dos atributos existentes que não contribuem para a performance do modelo preditivo. Entre as técnicas de *Feature Selection* disponíveis, neste trabalho foi adotada a técnica a *Greedy Backward Selection* (GBS). Essa técnica inicia a avaliação com todos os atributos até então selecionados e em cada nova avaliação elimina individualmente cada um dos atributos e em cada avaliação compara-se as métricas alcançadas com a eliminação de determinado atributo com as métricas alcançadas com todos os atributos selecionados. O objetivo da técnica GBS é encontrar atributos que ao serem removidos do conjunto de atributos, obtenham um F1-Score superior ao F1-Score obtido com todos os atributos disponíveis.

[3] Propriedade correspondente ao processo de *One-Hot Encoding* na propriedade *OPERATOR*
[4] Devido ao grande número de operadores disponíveis, não foi possível colocar todos eles nesta tabela
[5] Propriedade correspondente ao processo de *One-Hot Encoding* na propriedade *TYPE_STATEMENT*

Tabela 10 – Propriedades do mutante #163 do programa CheckPalindrome após a preparação dos dados

Propriedade	**Valor**
u-Cccr[3]	0
u-VVDL[3]	1
u-CRCR[3]	0
u-OCNG[3]	0
...[4]	...
u-Oido[3]	0
u-OODL[3]	0
u-ORAN[3]	0
SOURCE_PRIMITIVE_ARC	1
TARGET_PRIMITIVE_ARC	0
DISTANCE_BEGIN_MIN	0.3333
DISTANCE_BEGIN_MAX	0.6666
DISTANCE_BEGIN_AVG	0.5555
DISTANCE_END_MIN	0.5000
DISTANCE_END_MAX	0.8333
DISTANCE_END_AVG	0.7500
COMPLEXITY	0.5365
Block[5]	0
While[5]	1
For[5]	0
If[5]	0
Assignment[5]	0
Return[5]	0
Function Call[5]	0

Afim de verificar a eficácia dos atributos selecionados, foi aplicada a técnica *Greedy Backward Selection* em todos os classificadores e nos modelos para classificar mutantes minimais e mutantes equivalentes. A Tabela 11 exibe o F1-Score para todos os classificadores e todas as colunas retiradas individualmente para classificar mutantes minimais. Os valores exibidos em negrito, correspondem à atributos que obtiveram F1-Score superior ao F1-Score do classificador com todos os atributos.

Entre todos os atributos, o atributo que obteve o pior desempenho e unicamente obteve um F1-Score maior ao ser removido do que ao ser incluído com relação a todos os classificadores, foi o *DISTANCE_END_MIN*. Seguidamente, os atributos *TARGET_PRIMITIVE_ARC*, *DISTANCE_BEGIN_AVG*, *DISTANCE_END_MAX*, *SOURCE_PRIMITIVE_ARC* e *COMPLEXITY* obtiveram um F1-Score maior ao ser removido em 6 dos 7 classificadores. Os classificadores *LDA* e *RF* tiveram a indicação de que ao remover cada atributo individualmente, alcançaria um F1-Score maior que se tivessem todos os atributos, enquanto todos os outros classificadores tiveram cada um deles 8 atributos que ao serem removidos individualmente obtinham um F1-Score maior que o F1-Score com todos os atributos.

Tabela 11 – Resultados da aplicação do GBS para todos os atributos e classificadores para Mutantes Minimais (F1-Score)

Propriedade	DT	GNB	KNN	LDA	LR	RF	SVM
TODAS	80.27	21.79	73.43	67.46	67.54	82.51	74.75
OPERATOR	80.07	**64.62**	72.73	**69.25**	**69.26**	**83.34**	**74.82**
TYPE_STATEMENT	**80.29**	**65.22**	72.47	**68.41**	67.35	**83.64**	**76.12**
TARGET_PRIMITIVE_ARC	**80.55**	**62.07**	**73.57**	**68.23**	**68.80**	**83.09**	74.69
DISTANCE_BEGIN_MIN	79.28	**62.00**	**74.01**	**70.15**	66.67	**83.37**	**75.41**
DISTANCE_BEGIN_AVG	**80.43**	**59.65**	**73.72**	**69.59**	67.47	**82.91**	**75.31**
DISTANCE_END_MAX	**81.28**	**53.78**	73.09	**70.08**	**67.94**	**83.46**	**75.79**
DISTANCE_BEGIN_MAX	79.73	**54.50**	**74.41**	**68.84**	**68.34**	**83.43**	74.70
DISTANCE_END_MIN	**82.07**	**43.58**	**73.59**	**69.18**	**69.36**	**83.41**	**74.95**
DISTANCE_END_AVG	**80.33**	20.99	**73.61**	**69.63**	**68.56**	**82.90**	73.38
SOURCE_PRIMITIVE_ARC	**81.41**	14.68	**74.05**	**69.47**	**68.51**	**83.18**	**75.63**
COMPLEXITY	**81.12**	13.08	**74.68**	**69.58**	**67.75**	**83.00**	**75.63**

A Tabela 12 exibe o F1-Score para todos os classificadores e todas as colunas retiradas individualmente para classificar mutantes equivalentes. Os valores exibidos em negrito, correspondem à atributos que obtiveram F1-Score superior ao F1-Score do classificador com todos os atributos.

Tabela 12 – Resultados da aplicação do GBS para todos os atributos e classificadores para Mutantes Equivalentes (F1-Score)

Propriedade	DT	GNB	KNN	LDA	LR	RF	SVM
TODAS	77.73	69.39	78.53	74.36	73.69	82.24	78.27
OPERATOR	**79.78**	**69.49**	76.82	74.22	**75.24**	81.70	**78.51**
DISTANCE_BEGIN_MIN	77.68	**69.55**	**79.34**	73.86	73.67	82.08	**78.73**
DISTANCE_END_MAX	**78.05**	69.36	**79.19**	73.77	**74.36**	81.22	**78.79**
TARGET_PRIMITIVE_ARC	**79.15**	69.30	77.95	73.49	73.92	81.72	**79.01**
DISTANCE_BEGIN_MAX	**79.42**	69.34	78.12	**74.98**	73.08	81.52	78.04
DISTANCE_END_MIN	77.16	**69.51**	78.15	74.20	**74.71**	**82.31**	**78.38**
TYPE_STATEMENT	**78.81**	69.39	78.19	73.67	**74.30**	81.10	**78.49**
COMPLEXITY	**78.84**	69.23	78.43	73.91	73.56	80.93	**78.82**
DISTANCE_BEGIN_AVG	**78.36**	**69.64**	78.30	73.57	73.37	80.91	**79.22**
DISTANCE_END_AVG	**78.70**	**69.49**	77.27	73.87	73.95	81.81	77.87
SOURCE_PRIMITIVE_ARC	**78.99**	**69.64**	77.93	73.91	72.26	81.29	**78.58**

Diferentemente da análise para identificação de mutantes minimais, na classificação de mutantes equivalentes, não houve nenhum atributo que obtivesse um F1-Score maior ao ser removido do que incluído em todos os classificadores. Igualmente, quatro atributos obtiveram os piores desempenhos com relação ao número de classificadores em que obtiveram um F1-Score maior ao ser removido do que incluído, foram os atributos *OPERATOR*, *DISTANCE_END_MAX* e *DISTANCE_END_MIN*. Os atributos que obtiveram o melhor desempenho foram o *TARGET_PRIMITIVE_ARC*, *COMPLEXITY* e *DISTANCE_END_AVG* que obtiveram um F1-Score maior

ao ser removido do que incluído em apenas 3 classificadores. Os classificadores *DT* e *SVM* tiveram a indicação de que ao remover cada atributo individualmente, alcançaria um F1-Score maior que se tivessem todos os atributos em 9 dos 11 atributos, em contra partida, o *LDA* obteve essa indicação em apenas 1 atributo e o *KNN* em 2 atributos.

5.5.3 Criação dos modelos preditivos

Depois de todos os dados terem sido coletados e devidamente tratados conforme descrito nas seções anteriores, os dados conhecimentos como dados de treinamento, servem de entrada para a criação dos modelos preditivos. Ao todo, neste trabalho, foram desenvolvidos 128 modelos preditivos, separados da seguinte maneira de acordo com os classificadores listados na Tabela 5:

- 7 algoritmos classificadores: KNN, DT, RF, SVM, LDA, LR e GNB;
- Os primeiros três algoritmos (KNN, DT e RF) tiveram parâmetros customizados para execução, resultando em:
  - KNN: 40 modelos preditivos;
  - DT: 10 modelos preditivos;
  - RF: 10 modelos preditivos;
- Para cada classificador (40 do KNN, 10 do DT, 10 do RF, 1 do SVM, 1 do LDA, 1 do LR e 1 do GNB), é criado um modelo preditivo para classificar Mutantes Minimais e outro Modelo Preditivo para classificar Mutantes Equivalentes, portanto, são 64 modelos preditivos para Mutantes Minimais e 64 modelos preditivos para Mutantes Equivalentes.

A Figura 7 sumariza o processo de criação do modelo preditivo.

Figura 7 – Processo de criação do modelo preditivo

5.5.4 Aplicação da abordagem

O resultado do processo de desenvolvimento da abordagem são os modelos preditivos que posteriormente são utilizados para classificar novos mutantes. A aplicação da abordagem desenvolvida se dá por meio da utilização do código fonte de um programa e um modelo preditivo como entradas e o resultado é a classificação dos mutantes daquele programa (minimal ou não, equivalente ou não). A seguir, são descritos os passos que devem ser executados para aplicar a abordagem em um programa:

1. Todos os operadores de mutação da ferramenta Proteum são utilizados para gerar mutantes para todas as funções existentes no programa utilizado como dado de entrada;

2. São gerados os GFCs para cada função existente no programa;

3. Para cada GFC gerado, são identificados os arcos primitivos;

4. Os dados coletados de todos os mutantes de todas as funções foram reunidos em um único conjunto de dados;

5. Foram aplicadas técnicas de aprendizado de máquina para pré processar os dados gerados;

6. Após os dados terem sido gerados, esses dados são utilizados como entrada no modelo preditivo para classificar os mutantes do programa selecionado.

A Figura 8 sumariza o processo de aplicação da abordagem.

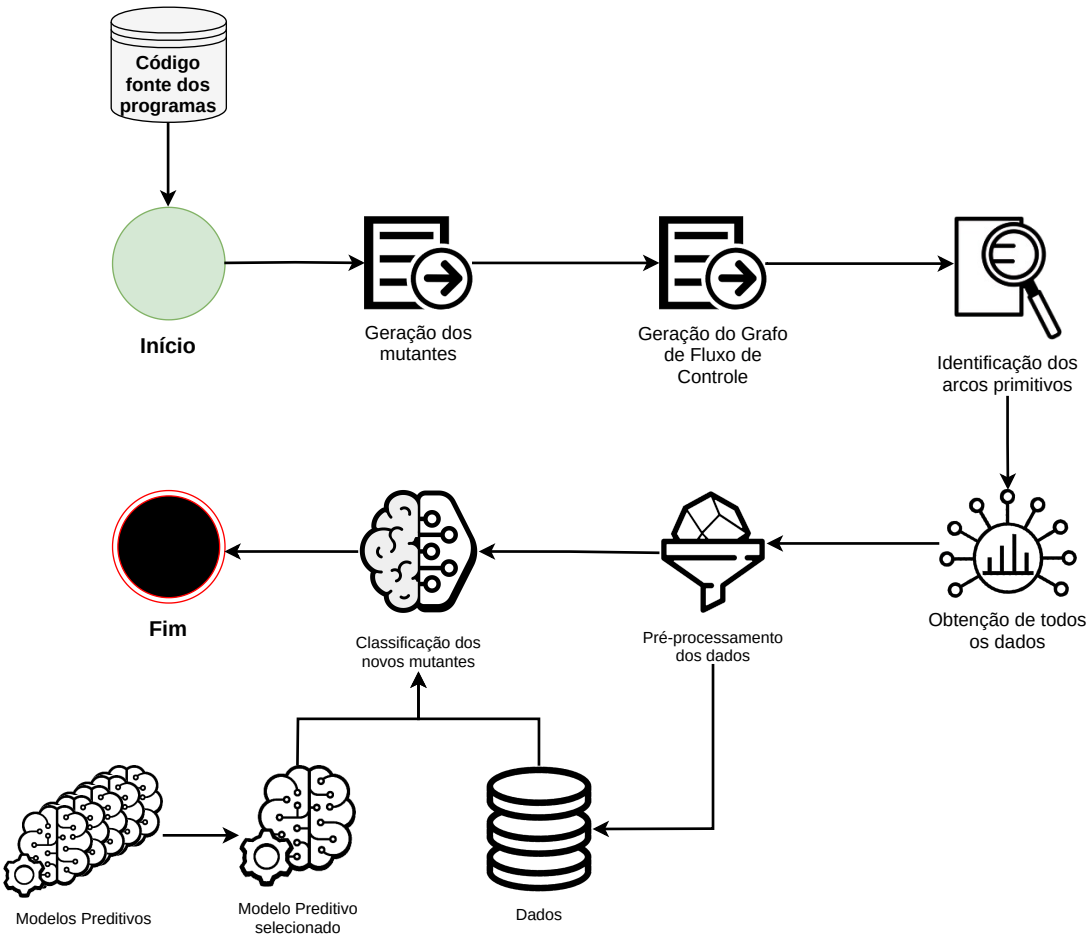


Figura 8 – Passos executados para aplicação da abordagem

5.6 Considerações finais

Conforme discutido no capítulo Capítulo 2, existem algumas desafios que dificultam a ampla utilização do Teste de Mutação e essas dificuldades se resumem ao grande número de mutantes gerados e a dificuldade na identificação de mutantes minimais. Sendo assim, conforme apresentado no Capítulo 4, o Aprendizado de Máquina pode empenhar um importante papel na redução do custo do Teste de Mutação. Neste capítulo foi apresentada uma abordagem para identificação de mutantes minimais e equivalentes. Foram apresentados algoritmos de aprendizados de máquina, propriedades de programas e uma visão geral da abordagem que permite uma replicação deste estudo, independentemente dos programas aqui escolhidos para o desenvolvimento desta abordagem.

CAPÍTULO

6

AVALIAÇÃO EXPERIMENTAL

6.1 Análise dos Resultados

Nesta seção são apresentados os resultados do experimento. Os resultados para os classificadores, programas e modelos preditivos são sumarizados em tabelas e gráficos. As análises são dividas em duas subseções: (i) Modelos Preditivos, onde os modelos preditivos são avaliados com respeito ao conjunto inteiro de dados, e (ii) Programas, onde os modelos preditivos são avaliados com respeito à classificação de programas individualmente, isto é, os modelos preditivos são avaliados utilizando os dados de um programa específico como conjunto de dados.

6.1.1 Modelos Preditivos

A primeira avaliação realizada foi acerca da capacidade geral dos modelos preditivos de classificar mutantes como minimais ou não, equivalentes ou não. Para isso, foram realizada 30 execuções para cada um dos 128 modelos preditivos, e a Tabela 13 exibe a média dos resultados dos 14 modelos preditivos principais.

Tabela 13 – Média dos resultados de 30 execuções dos modelos preditivos

		KNN[1]	DT[2]	RF[3]	SVM	LDA	LR	GNB
Acurácia	**Min**	73.30	80.71	82.12	67.22	68.52	68.23	54.79
	Equi	77.13	78.04	80.42	72.88	73.77	73.52	56.57
Precisão	**Min**	73.20	81.16	82.07	65.81	68.56	68.44	63.91
	Equi	75.16	77.63	79.84	74.93	73.09	73.31	53.57
Revocação	**Min**	73.69	80.19	82.40	72.00	68.60	67.82	41.16
	Equi	81.18	78.95	81.79	68.88	75.35	74.08	98.77
F1	**Min**	73.39	80.58	**82.22**	68.70	68.53	68.07	39.25
	Equi	78.02	78.25	**80.75**	71.71	74.16	73.65	69.46

Para visualizar o resultado da tabela acima de forma gráfica, a Figura 9 exibe na forma de um gráfico de barra o F1-Score obtido para cada um dos 14 modelos preditivos. Acima da barra de cada um dos modelos, é possível visualizar o F1-Score médio para aquele modelo preditivo.

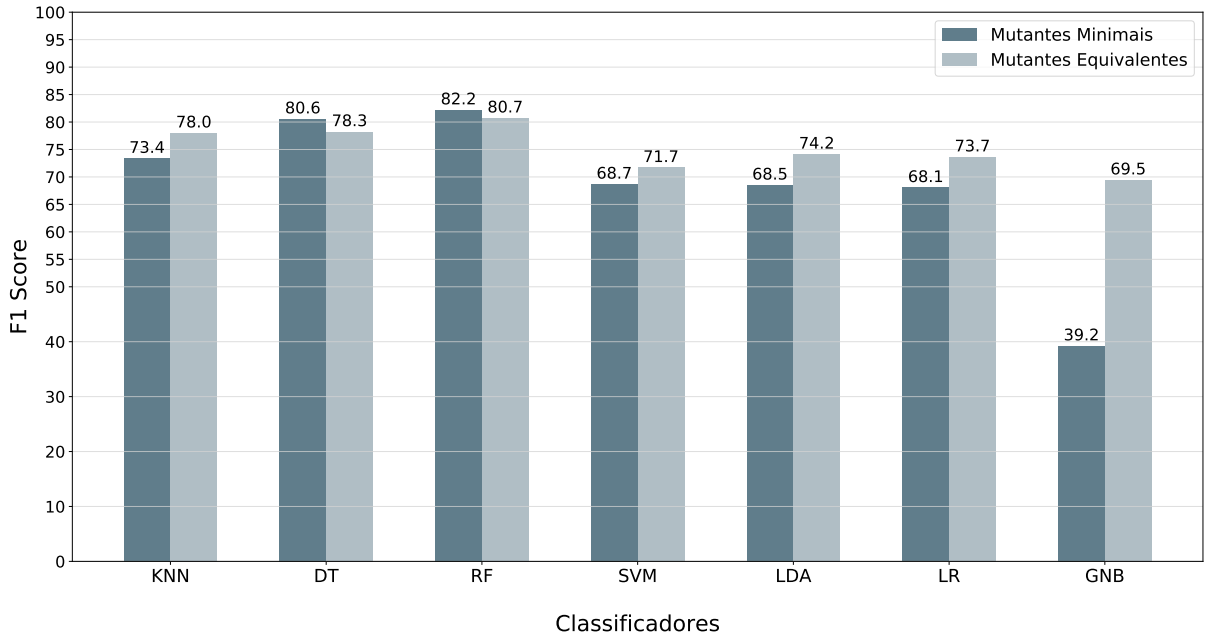


Figura 9 – Gráfico de barra com o F1-Score médio obtido para os 14 melhores modelos preditivos

Para a classificação de mutantes minimais, o classificador que obteve a maior média geral foi o RF que obteve 82,22 de F1-Score, 82,12 de acurácia, 82,07 de precisão e 82,40 de revocação, seguido pelo classificador DT que obteve 80,58 de F1-Score. O classificador que obteve a menor média geral foi o GNB, que obteve 39,25 de F1-Score, 54,79 de acurácia, 63,91 de precisão e 41,16 de revocação.

Para a classificação de mutantes equivalentes, o classificador que obteve a maior média geral foi também o RF que obteve 80,75 de F1-Score, 80,42 de acurácia, 79,84 de precisão e 81,79 de revocação, seguido pelo classificador DT que obteve 78,25 de F1-Score. O classificador que obteve a menor média geral também foi o GNB, que obteve 69,46 de F1-Score, 56,57 de acurácia, 53,57 de precisão e 98,77 de revocação.

Para comparar o resultado de cada uma das 30 execuções dos 14 modelos preditivos acima listados, a Figura 10 exibe um *boxplot* comparando o F1 Score de todos os modelos preditivos. No *boxplot* mencionado, a média para cada modelo preditivo é exibida no topo da figura e também indicado através das linhas pontilhadas que cruzam as caixas. As linhas contínuas que cruzam as caixas, indicam a mediana para cada um dos modelos preditivos.

[1] Os valores exibidos correspondem ao *n_neighbors* = 1 para mutantes minimais e *n_neighbors* = 11 para mutantes equivalentes

[2] Os valores exibidos correspondem ao *min_samples_split* = 15 para mutantes minimais e *min_samples_split* = 35 para mutantes equivalentes

[3] Os valores exibidos correspondem ao *min_samples_split* = 5 para mutantes minimais e *min_samples_split* = 15 para mutantes equivalentes

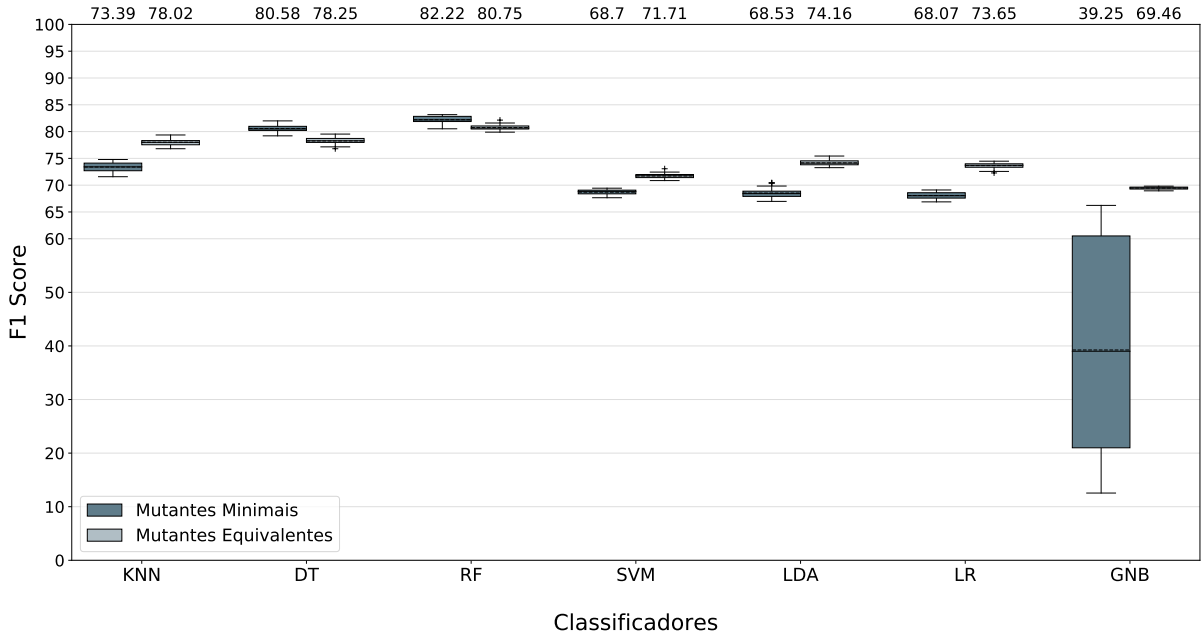


Figura 10 – *Boxplot* com os resultados do F1-Score das 30 execuções para cada classificador

Entre os 14 modelos preditivos citados, 13 deles tiveram os resultados das 30 execuções bastante semelhantes, com um desvio padrão inferior a 1, porém, o GNB para classificar mutantes minimais obteve o F1 Score com uma disparidade alta nas 30 execuções. A Tabela 14 exibe os modelos preditivos e o desvio padrão das 30 execuções para cada modelo preditivo.

Tabela 14 – Desvio padrão do F1 Score das 30 execuções para cada classificador

Classificador	Minimais	Equivalentes
KNN	0.8438	0.6131
DT	0.6366	0.6642
RF	0.6506	0.4968
SVM	0.4924	0.4720
LDA	0.8890	0.5825
LR	0.6199	0.5208
GNB	19.2085	0.2285

Afim de demonstrar os valores obtidos para todos os classificadores, inclusive daqueles que têm parâmetros customizados, abaixo são exibidas tabelas com os valores médios das 30 execuções para todos os parâmetros customizados dos algoritmos classificadores KNN, DT e RF. A Tabela 15 exibe os valores para cada um dos 80 modelos preditivos (40 para classificar mutantes minimais e 40 para classificar mutantes equivalentes) que utilizam o algoritmo KNN, bem como o parâmetro customizado, que no caso do KNN é o *n_neighbors* (número de vizinhos utilizados para classificar o dado novo).

A Tabela 16 exibe os valores para cada um dos 20 modelos preditivos (10 para classificar mutantes minimais e 10 para classificar mutantes equivalentes) que utilizam o algoritmo DT,

Tabela 15 – Média dos resultados de 30 execuções para todos os modelos preditivos com parâmetros customizados do classificador KNN

Parâmetro	Mutantes Minimais				Mutantes Equivalentes			
	Acurácia	**Precisão**	**Revocação**	**F1**	**Acurácia**	**Precisão**	**Revocação**	**F1**
1	73.30	73.20	73.69	**73.39**	74.34	75.66	71.87	73.67
2	70.66	78.25	57.34	66.09	70.49	79.17	55.69	65.30
3	71.72	70.75	74.22	72.40	76.13	75.03	78.44	76.66
4	70.78	74.78	62.82	68.20	75.04	78.22	69.52	73.56
5	70.57	69.71	72.91	71.23	77.00	75.12	80.85	77.84
6	69.96	72.83	63.80	67.95	76.01	77.28	73.77	75.44
7	70.03	69.30	72.04	70.60	76.84	74.76	81.17	77.80
8	70.32	72.44	65.72	68.84	76.51	77.05	75.62	76.29
9	70.41	69.55	72.75	71.06	76.78	74.73	81.06	77.73
10	70.47	71.91	67.34	69.48	76.89	77.00	76.79	76.85
11	70.60	69.60	73.33	71.36	77.13	75.16	81.18	**78.02**
12	70.88	71.85	68.82	70.24	77.11	76.95	77.54	77.20
13	71.00	69.80	74.18	71.88	77.15	75.25	81.05	78.00
14	70.73	71.37	69.37	70.29	77.02	76.63	77.86	77.20
15	70.77	69.57	74.00	71.66	77.00	75.15	80.80	77.84
16	70.33	70.98	68.93	69.88	76.79	76.39	77.67	76.99
17	70.59	69.55	73.42	71.38	76.80	75.10	80.33	77.58
18	70.47	71.11	69.12	70.03	76.65	76.23	77.57	76.85
19	70.63	69.67	73.27	71.37	76.57	74.95	79.97	77.34
20	70.75	71.36	69.50	70.35	76.69	76.20	77.75	76.92
21	70.67	69.82	73.00	71.31	76.54	74.92	79.92	77.30
22	70.60	71.17	69.41	70.21	76.69	76.06	78.03	76.99
23	70.61	69.84	72.73	71.20	76.56	74.84	80.16	77.37
24	70.77	71.31	69.65	70.40	76.59	75.72	78.39	76.99
25	70.73	70.02	72.69	71.27	76.46	74.66	80.25	77.32
26	70.90	71.33	70.05	70.62	76.44	75.44	78.51	76.91
27	70.89	70.17	72.85	71.43	76.36	74.43	80.44	77.28
28	71.16	71.50	70.56	70.96	76.29	75.19	78.61	76.82
29	70.97	70.20	73.05	71.54	76.27	74.29	80.48	77.22
30	71.10	71.32	70.73	70.96	76.27	75.11	78.70	76.82
31	70.95	70.17	73.04	71.52	76.17	74.16	80.47	77.15
32	70.89	71.11	70.52	70.76	76.15	74.91	78.75	76.74
33	70.70	69.97	72.68	71.24	76.10	74.03	80.53	77.11
34	70.66	70.82	70.43	70.57	76.04	74.83	78.59	76.62
35	70.52	69.85	72.39	71.04	76.04	74.03	80.33	77.01
36	70.53	70.65	70.41	70.47	76.01	74.80	78.58	76.60
37	70.40	69.63	72.52	70.99	75.86	73.91	80.09	76.83
38	70.51	70.50	70.72	70.55	75.91	74.67	78.55	76.52
39	70.42	69.60	72.71	71.06	75.86	73.89	80.11	76.84
40	70.59	70.40	71.25	70.76	75.93	74.61	78.74	76.58

bem como o parâmetro customizado, que no caso do DT é o *min_samples_split* (número mínimo de amostras para divisão do nó interno).

A Tabela 17 exibe os valores para cada um dos 20 modelos preditivos (10 para classificar mutantes minimais e 10 para classificar mutantes equivalentes) que utilizam o algoritmo RF, bem como o parâmetro customizado, que no caso do RF é o *min_samples_split* (número mínimo

Tabela 16 – Média dos resultados de 30 execuções para todos os modelos preditivos com parâmetros customizados do classificador DT

Parâmetro	Mutantes Minimais				Mutantes Equivalentes			
	Acurácia	**Precisão**	**Revocação**	**F1**	**Acurácia**	**Precisão**	**Revocação**	**F1**
5	80.51	81.30	79.41	80.29	77.93	78.50	76.98	77.71
15	80.71	81.16	80.19	**80.58**	78.18	78.27	78.20	78.19
25	80.47	80.64	80.33	80.43	78.07	77.86	78.50	78.14
35	80.16	80.14	80.35	80.15	78.04	77.63	78.95	**78.25**
45	79.88	79.90	80.07	79.94	77.85	77.28	79.00	78.12
55	79.63	79.51	80.03	79.69	77.55	77.03	78.71	77.82
65	79.39	79.24	79.81	79.47	77.29	76.87	78.26	77.50
75	79.12	78.99	79.56	79.21	76.97	76.52	78.02	77.21
85	78.78	78.49	79.45	78.92	76.65	76.28	77.55	76.86
95	78.55	78.45	78.95	78.60	76.52	76.19	77.36	76.70

de amostras para divisão do nó interno).

Tabela 17 – Média dos resultados de 30 execuções para todos os modelos preditivos com parâmetros customizados do classificador RF

Parâmetro	Mutantes Minimais				Mutantes Equivalentes			
	Acurácia	**Precisão**	**Revocação**	**F1**	**Acurácia**	**Precisão**	**Revocação**	**F1**
5	82.12	82.07	82.40	**82.22**	80.26	79.54	81.63	80.53
15	81.87	82.44	81.42	81.81	80.42	79.84	81.79	**80.75**
25	81.44	82.30	80.03	81.12	80.28	79.77	80.95	80.33
35	80.98	82.24	78.74	80.33	79.89	79.77	80.30	80.00
45	80.23	82.13	77.47	79.82	79.60	79.64	79.79	79.66
55	79.89	82.12	76.48	79.34	79.32	79.60	79.18	79.35
65	79.54	81.94	75.87	78.74	79.18	79.38	78.65	78.87
75	79.26	81.60	75.08	78.33	78.89	79.26	78.20	78.70
85	78.88	81.57	74.70	78.05	78.57	79.11	77.73	78.44
95	78.45	81.21	74.56	77.65	78.40	78.94	77.50	78.09

6.1.2 Comparação do desempenho dos modelos preditivos gerados para classificação dos programas individualmente

Os programas utilizados neste trabalho são diferentes entre si em diversas propriedades e essas diferenças podem impactar no resultado da classificação dos modelos, isto é, os mutantes de um programa podem ser corretamente classificados utilizando um determinado algoritmo, porém, para os mutantes de outro programa, este mesmo algoritmo pode não ser o melhor. Para isso, nesta subseção nós investigamos individualmente cada programa e cada modelo preditivo.

Avaliação 1 - Classificação dos mutantes dos programas de forma isolada

Nesta etapa, nós avaliamos cada programa individualmente e utilizamos tanto para treino (criação do modelo preditivo) como para teste, unicamente os dados dos mutantes do programa sob avaliação. Desta forma, nós dividimos os dados em dados de treino e dados de teste: Os

dados de treino são utilizados para a criação dos modelos preditivos, enquanto os dados de teste são utilizados para avaliar a efetividade do modelo criado. A Figura 11 esquematiza o processo conduzido para criar e avaliar cada modelo preditivo gerado individualmente para cada programa.

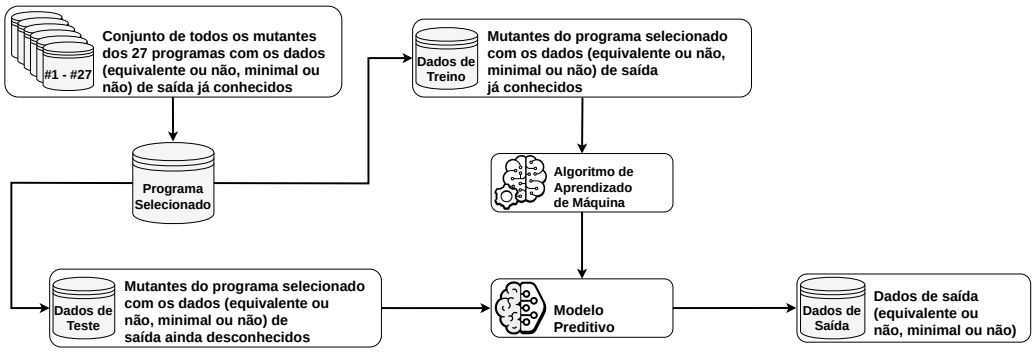


Figura 11 – Passos executados para obtenção dos dados da Tabela 18 - Avaliação 1

Os dados resultantes dessa avaliação são exibidos na Tabela 18. Essa tabela exibe o ID de cada programa e o F1-Score obtido para cada programa utilizando os classificadores KNN, DT, RF, SVM, LDA, LR e GNB, além disso, a tabela também faz a distinção dos valores obtidos para classificar mutantes minimais e mutantes equivalentes. Na parte inferior da tabela, é exibido o F1-Score mínimo, máximo e médio para os 27 programas, tanto para mutantes minimais como equivalentes, além de indicar o número de programas em que cada classificador foi considerado o melhor.

Para comparar o resultado de cada um dos 27 programas dos 14 modelos preditivos utilizados, a Figura 12 exibe um *boxplot* comparando o F1 Score de todos os modelos preditivos. No *boxplot* mencionado, a média para cada modelo preditivo é exibida no topo da figura e também é indicada através das linhas pontilhadas que cruzam as caixas. As linhas contínua que cruzam as caixas, indicam a mediana para cada um dos modelos preditivos.

Para os classificadores KNN, DT e RF, são exibidos os valores apenas para os melhores parâmetros encontrados para tais programas. Os dados sobre os melhores parâmetros (independente do programa) para estes classificadores pode ser encontrado nas Tabelas 15, 16 e 17, enquanto o resultado para o melhor parâmetro para cada um dos classificadores e programas pode ser exibido na Tabela 19.

Para a classificação de mutantes minimais, os classificadores RF e LDA se destacaram frente aos demais, sendo considerado os melhores classificadores para 7 programas cada um. Os algoritmos KNN e DT obtiveram 0 de F1-Score para 6 programas diferentes, programa #7 e #17 para o KNN e #5, #7, #16, #17 e #22 para o DT. O algoritmo que obteve a maior média foi o LR, que obteve 66,94 de média, 92,87 como F1 máximo e 27,78 como F1 mínimo. O algoritmo KNN obteve 53,69 de média, a menor média entre os classificadores, obtendo também 84,61 como F1 máximo e 0,00 como F1 mínimo. O maior F1 entre os 27 programas e os 7 classificadores, foi o

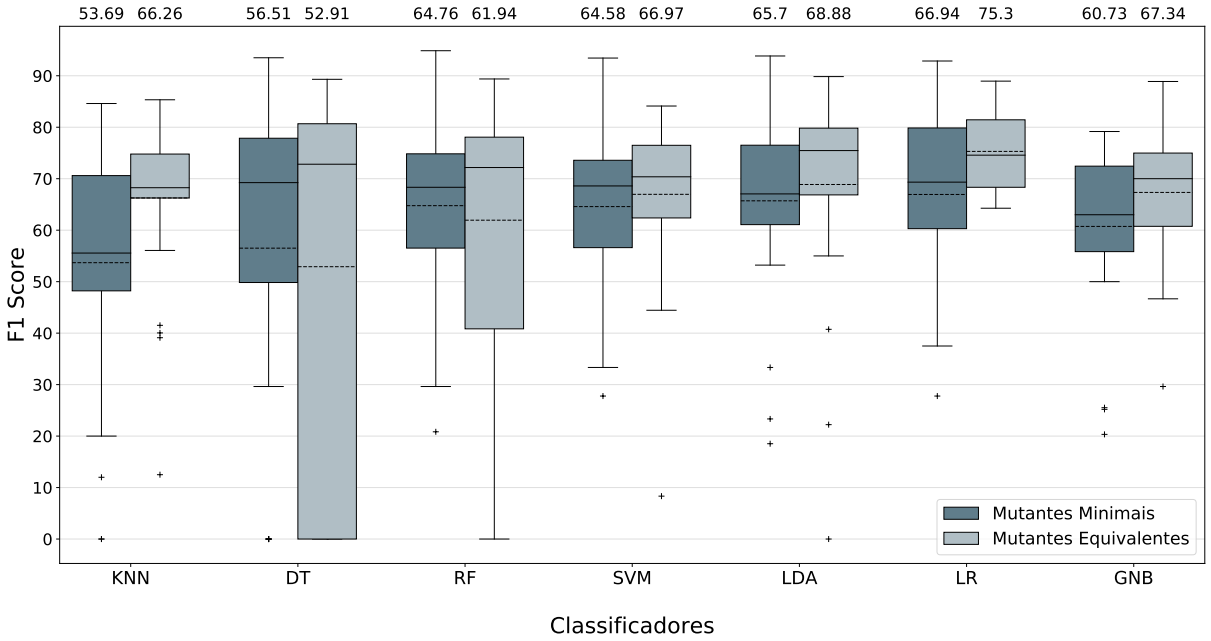


Figura 12 – *Boxplot* com os resultados do F1-Score para cada um dos 14 modelos classificadores

valor obtido pelo programa #27 no RF, que foi 94,86.

Para a classificação de mutantes equivalentes, os classificadores DT e LDA se destacaram frente aos demais, sendo considerado os melhores classificadores para 6 programas cada um. Os algoritmos DT, RF e LDA obtiveram 0 de F1-Score para 8 programas diferentes, programa #3, #5, #8, #9, #13, #16, #18 e #22 para DT e #3 para RF e LDA. O algoritmo que obteve a maior média foi o LR, que obteve 75,30 de média, 88,96 como F1 máximo e 64,27 como F1 mínimo. O algoritmo DT obteve 52,91 de média, a menor média entre os classificadores, obtendo também 89,31 como F1 máximo e 0,00 como F1 mínimo. O maior F1 entre os 27 programas e os 7 classificadores, foi o valor obtido pelo programa #1 no LDA, que foi 89,84.

Avaliação 2 - Classificação dos mutantes dos programas de forma conjunta aos mutantes dos outros programas

Nesta etapa, foram avaliados cada programa individualmente, porém, diferentemente da etapa anterior, nesta etapa os dados dos mutantes do programa sob avaliação são utilizados apenas para a fase de teste do modelo, enquanto para a fase de treino (criação do modelo preditivo), são utilizados os dados dos mutantes dos outros 26 programas. A Figura 13 esquematiza o processo conduzido para criar e avaliar cada modelo preditivo gerado individualmente para cada programa.

Os dados resultantes dessa avaliação são exibidos na Tabela 20. Essa tabela exibe o ID de cada programa, o melhor classificador para aquele programa (e também o parâmetro para os classificadores que são parametrizados), bem como a Acurácia, Precisão, Revocação e F1-Score para aquele classificador naquele programa, além disso, a tabela também faz a distinção dos

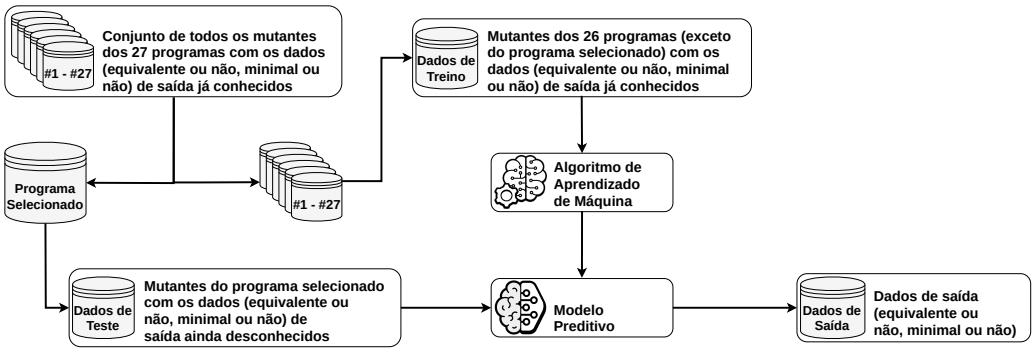


Figura 13 – Passos executados para obtenção dos dados da Tabela 20 - Avaliação 2

valores obtidos para classificar mutantes minimais e mutantes equivalentes.

Os dados exibidos na Tabela 20 são exibidos na forma de gráficos a seguir, exibindo o F1-Score e a Acurácia para cada um dos programas. Os programas são listados de forma a colocar os programas com maior F1-Score à esquerda e seguindo com os programas com menor F1-Score à direita. A Figura 14 exibe o gráfico com os valores para classificar mutantes minimais e a Figura 15 exibe o gráfico com os valores para classificar mutantes equivalentes.

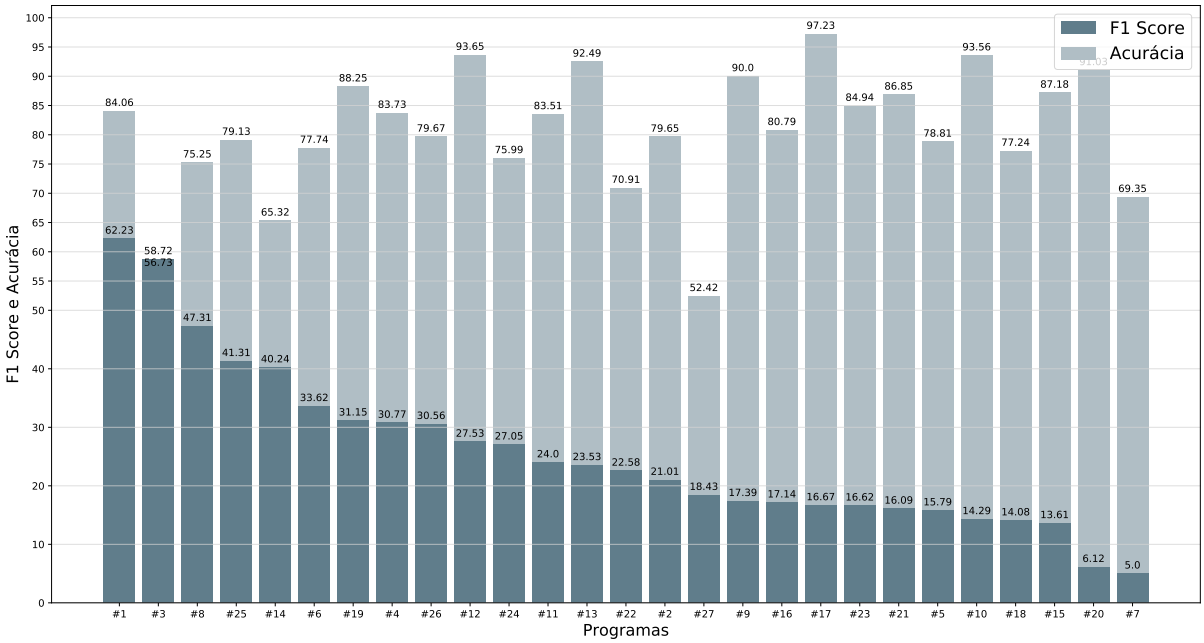


Figura 14 – Métricas para a classificação de cada programa com o melhor classificador - Mutantes Minimais

Para a classificação de mutantes minimais, a média geral do F1-Score de todos os programas com os melhores classificadores foi de 25,66. Os classificadores RF e DT se destacaram frente aos demais, sendo considerados os melhores classificadores para 9 e 8 programas, respectivamente. Em contrapartida, os classificadores LDA, LR e GNB foram os piores classificadores entre os programas, sendo considerados os melhores classificadores apenas para 1 programa cada um deles. Os maiores índices para o F1 foram obtidos pelos programas #1 (RF), #3 (DT) e #8 (KNN), alcançando 62,23, 58,72 e 47,31, respectivamente. Os piores índices para o F1 foram

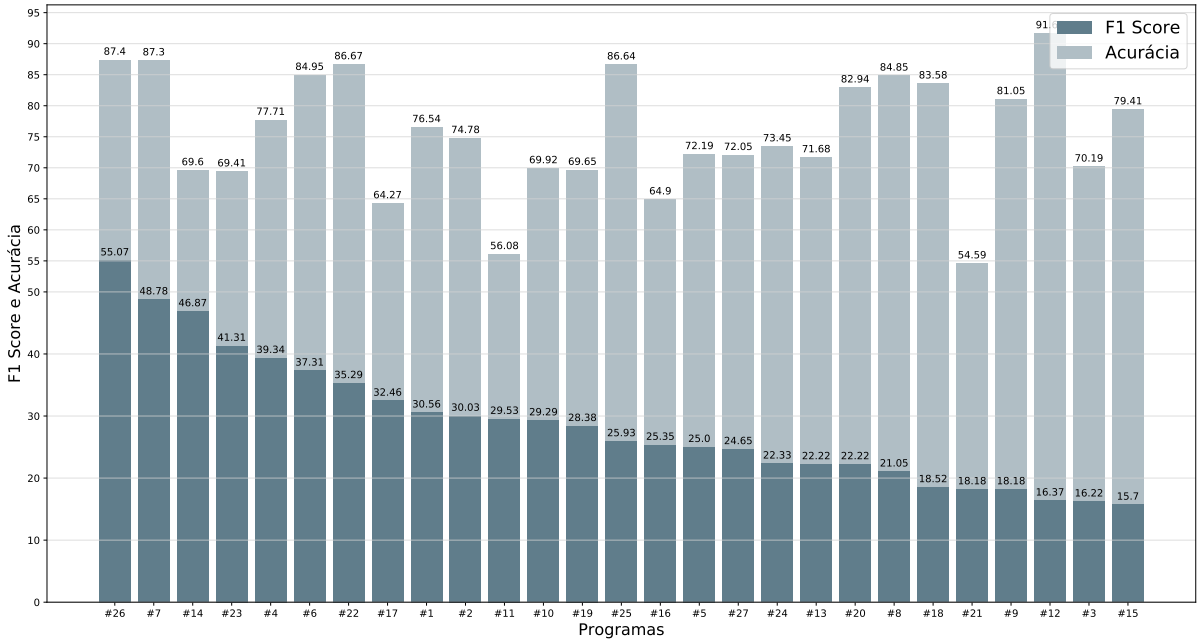


Figura 15 – Métricas para a classificação de cada programa com o melhor classificador - Mutantes Equivalentes

obtidos pelos programas #7 (KNN), #20 (RF) e #15 (DT), com os valores 5,00, 6,12 e 13,61, respectivamente.

Para a classificação de mutantes equivalentes, a média geral do F1-Score de todos os programas com os melhores classificadores foi de 28,75. Os classificadores KNN e 8 se destacaram frente aos demais, sendo considerados os melhores classificadores para 8 programas cada um. Em contrapartida, os classificadores LR e GNB foram os piores classificadores entre os programas, sendo considerados os melhores classificadores apenas para nenhum e para 1 programa, respectivamente. Os maiores índices para o F1 foram obtidos pelos programas #26 (DT), #7 (SVM) e #14 (KNN), alcançando 55,07, 48,78 e 46,87, respectivamente. Os piores índices para o F1 foram obtidos pelos programas #15 (KNN), #3 (GNB) e #12 (DT), com os valores 15,70, 16,22 e 16,37, respectivamente.

6.2 Teste de hipóteses

Antes da execução dos testes estatísticos, primeiramente faz-se necessário verificar a distribuição dos dados. Para cada um dos 7 classificadores dos 2 modelos preditivos, deve-se avaliar se os resultados encontrados seguem uma distribuição paramétrica (normal) ou não paramétrica. Para mutantes minimais e equivalentes, o F1-Score de todos os 7 classificadores foram apresentados em gráficos *Quantile to Quantile* (Q-Q) (DODGE, 2008), que podem ser visualizados nas Figuras 16 e 17.

Além da representação por meio dos gráficos, a normalidade dos dados foi medida por meio do teste de *Shapiro-Wilk*. O teste *Shapiro-Wilk* analisa todo o conjunto de dados e

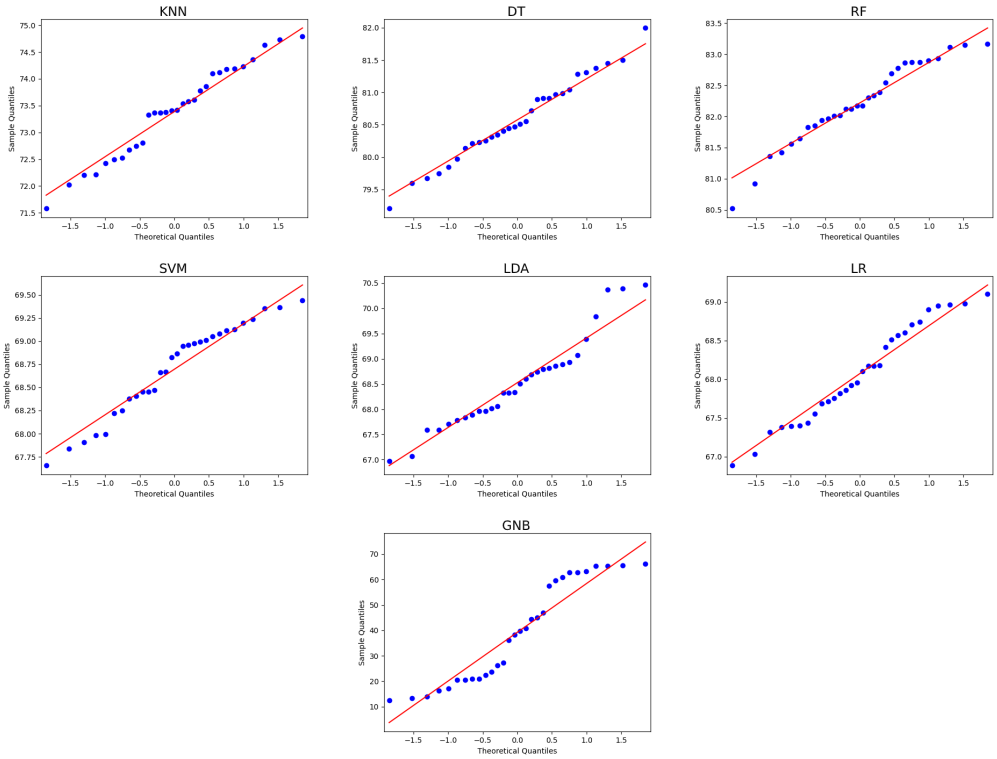


Figura 16 – *Quantile to Quantile* do F1-Score para classificação de Mutantes Minimais

verifica se este conjunto é diferente estatisticamente de uma distribuição normal. Como hipótese nula, define-se que os dados analisados assumem uma distribuição normal, enquanto a hipótese alternativa define que os dados assumem uma distribuição não normal. A Tabela 21 exibe o *p-value* e o resultado do teste de *Shapiro-Wilk* para todos os classificadores.

De acordo com o teste *Shapiro-Wilk*, um grau de significância $p < 0.05$ indica que os dados não seguem uma distribuição normal, enquanto um valor de $p > 0.05$ indica que os dados seguem uma distribuição normal (JOHNSON; BHATTACHARYYA, 2019). Entre todos os classificadores e modelos preditivos, o único classificador que foi classificado como não paramétrico foi o GNB para classificar mutantes minimais.

Para verificar se há diferença estatística entre o *F1-Score* de todos os classificadores, deve ser aplicado o Teste T Pareado para os dados que são paramétricos e o Teste de Wilcoxon para dados não paramétricos. Para mutantes minimais e equivalentes, cada classificador foi comparado com os outros 6 classificadores afim de verificar se há diferença estatística no resultado entre os classificadores. Para os testes estatísticos citados, um grau de significância $p < 0.05$ indica que existe diferença estatística entre os resultados dos classificadores, enquanto um valor de $p > 0.05$ indica que não há evidências para indicar diferença estatística entre os resultados dos classificadores.

A Tabela 22 exibe o valor do *p-value* entre o *F1-Score* obtido pelos classificadores ao classificar mutantes minimais.

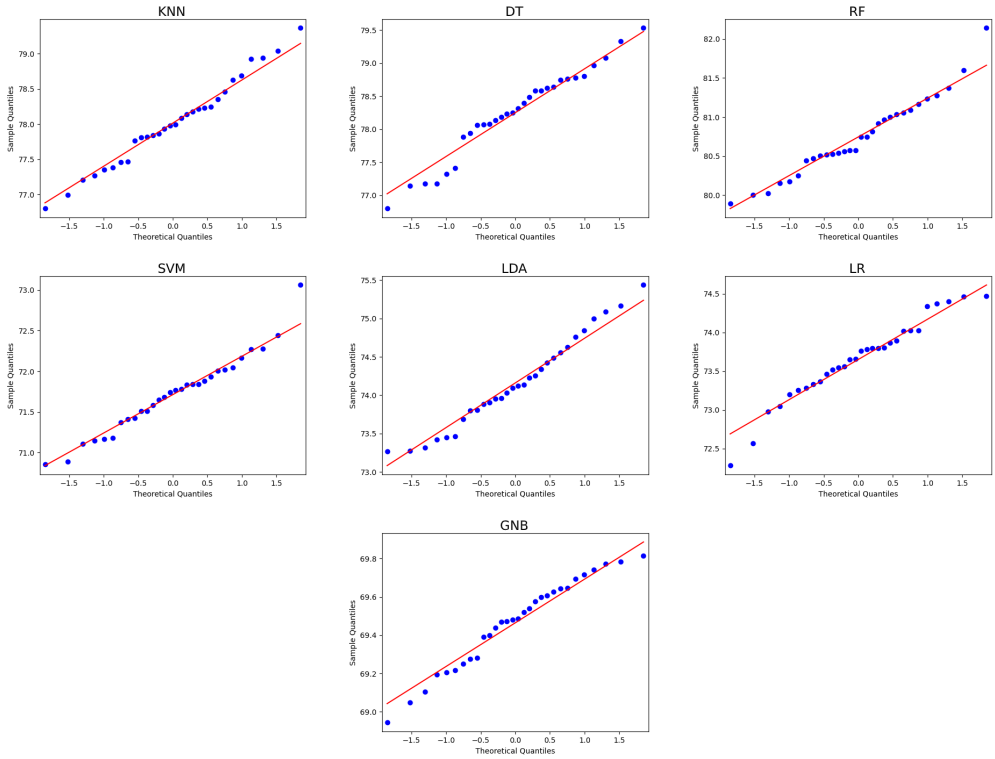


Figura 17 – *Quantile to Quantile* do F1-Score para classificação de Mutantes Equivalentes

Através da tabela supracitada, é possível identificar que em apenas 1 relação não é possível evidenciar diferença estatística, que é o caso dos classificadores SVM e LDA que obtiveram um $p = 0.3575$. Os resultado mais próximos de atingir o limiar de $p > 0.05$ foram os classificadores LDA e LR que obtiveram um $p = 0.0490$, e SVM e LR que obtiveram um $p = 0.0013$, contudo, ainda é possível afirmar que há diferença estatística entre estes classificadores.

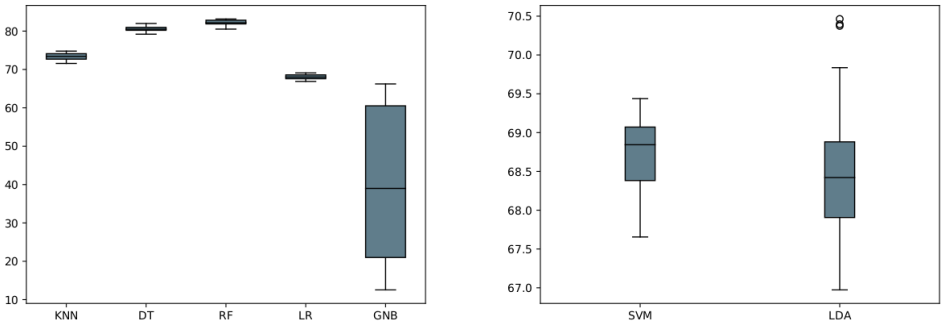


Figura 18 – *Box-plot* do F1-Score para classificação de Mutantes Minimais

A Tabela 23 exibe o valor do *p-value* entre o *F1-Score* obtido pelos classificadores ao classificar mutantes equivalentes.

Através da tabela supracitada, é possível identificar que em apenas 1 relação não é possível evidenciar diferença estatística, que é o caso dos classificadores KNN e DT, que

obtiveram um $p = 0.1752$. Os resultados mais próximos de atingir o limiar de $p < 0.05$ foram os classificadores LDA e LR que obtiveram um $p = 0.0052$, contudo, ainda é possível afirmar que há diferença estatística entre estes classificadores.

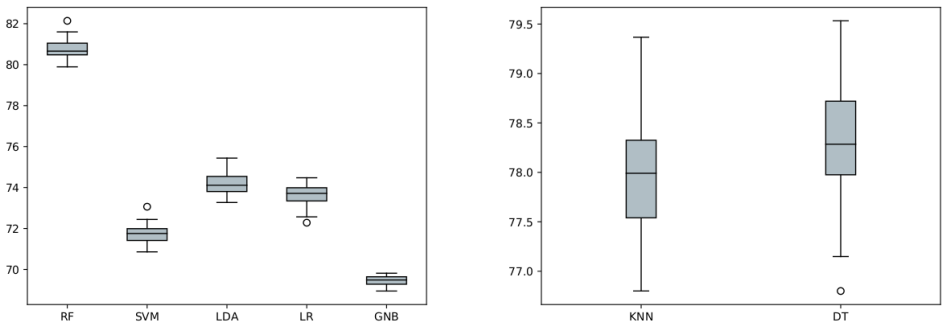


Figura 19 – *Box-plot* do F1-Score para classificação de Mutantes Equivalentes

6.3 Ameaças à validade

A primeira ameaça à validade externa refere-se a generalização dos resultados obtidos, em decorrência das amostras coletadas para desenvolvimento da abordagem e avaliação experimental. Para o desenvolvimento da abordagem, era necessário o conjunto adequado de teste do programa a ser incluído na abordagem, portanto, conforme mencionado na Seção 3.2, identificar o conjunto adequado de testes de um programa, é um problema indecidível, sendo assim, os programas selecionados para o desenvolvimento da abordagem não foram a partir de uma seleção aleatória de um conjunto maior de programas, isto é, os programas selecionados são programas que já se conhecia o conjunto adequado de teste e já haviam sido utilizados em estudos anteriores. Dessa forma, não é possível generalizar os resultados do experimento realizado neste trabalho, uma vez que não é possível garantir que os 27 programas selecionados são representativos. Afim de mitigar este problema, nós decidimos utilizar a técnica *k-Fold Cross Validation*, que tem como objetivo principal, evitar o *over-fetching*, que é o super ajuste do modelo preditivo aos dados de treinamento.

Além disso, os programas utilizados neste trabalho foram escritos na linguagem C, desta forma, não é possível garantir a generalização dos resultados para outras linguagens de programação, como Java ou Python, por exemplo. A utilização da linguagem é outra ameaça à validade pois cada linguagem de programação tem suas especificidades referentes aos operadores de mutação.

Outra ameaçada à validade externa, ainda referente aos programas utilizados, é no que tange ao tamanho dos programas utilizados neste trabalho. Os programas não são programas do mundo real (isto é, programas de escala industrial), portanto, é impossível averiguar a efetividade da abordagem em programas reais que normalmente são maiores que os programas utilizados

para o desenvolvimento da abordagem. Para mitigar essa ameaça à validade e todas outras que referem-se aos programas selecionados para a avaliação experimental, foram utilizados programas que já haviam sido utilizados em experimentos anteriores Delamaro, Offutt e Ammann (2014), Delamaro *et al.* (2014), Ammann, Delamaro e Offutt (2014) e Andrade *et al.* (2019).

Uma ameaça à validade interna se refere aos algoritmos de aprendizado de máquina utilizados no desenvolvimento da abordagem e a parametrização realizada para a criação dos modelos preditivos. Para mitigar a ameaça à validade referente aos algoritmos de aprendizado de máquina selecionados, levou-se em consideração estudos que mapearam o estado da arte referente à aplicação de aprendizado de máquina para resolver problemas de teste de software Durelli *et al.* (2019). Para mitigar a ameaça referente aos parâmetros dos algoritmos, foi procurado utilizar os parâmetros básicos dos algoritmos que assim permitiam, dessa forma, os algoritmos KNN, DT e RF tiveram variações nas definições de seus hiper-parâmetros e os resultados para cada um dos parâmetros foi relatado nas Tabelas 15, 16 e 17.

Outra ameaça à validade interna se refere às propriedades utilizadas para a criação dos modelos preditivos. Para tentar mitigar essa ameaça, foram levadas em consideração as hipóteses formuladas por Delamaro, Chaim e Maldonado (2018), que indica que a existência de mutantes minimais está relacionada com as informações do mutante no grafo de fluxo de controle. Depois disso, foi aplicada a técnica *Greedy Backward Selection* para verificar se as propriedades selecionadas são relevantes e necessárias para a efetividade dos modelos preditivos e os resultados indicaram que todas as propriedades eram essenciais para a precisão dos modelos preditivos num espectro geral, sem levar em consideração as especifidades de cada algoritmo classificador e cada programa.

6.4 Considerações finais

Neste capítulo, foi descrito em detalhes o experimento realizado para avaliar a abordagem, bem como os resultados dessa avaliação experimental. No início do capítulo, foi apresentado o design do experimento, bem como as questões de pesquisa que nortearam toda essa avaliação experimental, os objetivos do experimento no modelo *Goal/Question/Metric* (GQM) e a formulação de hipóteses que no final serviria para fornecer os subsídios necessários para responder às questões de pesquisa. A parte central deste capítulo, apresenta os resultados obtidos pelo experimento de avaliação da abordagem proposta neste trabalho e ainda que sem diferença significativa, os resultados apontam que o classificador que obteve melhor resultado tanto para classificar mutantes minimais como equivalentes, foi o *Random Forest* (RF). Após a apresentação e análise inicial dos resultados, foi apresentado o teste de hipóteses que avalia os resultados apresentados, responde às hipóteses formuladas e fornece subsídios para respostas às questões de pesquisa. Por último no capítulo, são apresentadas as ameças à validade deste projeto.

Tabela 18 – Média do F1 Score para classificação individual para cada programa e classificador

ID do Programa		**KNN**	**DT**	**RF**	**SVM**	**LDA**	**LR**	**GNB**
#1	Min	70.85	80.50	82.58	**87.00**	82.01	86.41	73.57
	Equi	68.25	83.94	**89.38**	84.12	89.84	88.96	86.41
#2	Min	67.33	69.24	**69.96**	63.89	65.39	64.34	62.22
	Equi	63.79	69.17	**75.56**	52.59	75.46	71.54	70.00
#3	Min	50.67	**83.95**	68.71	71.99	71.33	76.98	72.42
	Equi	**66.67**	0.00	0.00	66.67	0.00	**66.67**	55.56
#4	Min	12.00	**80.00**	54.67	76.33	61.67	61.33	53.33
	Equi	85.33	50.00	52.67	73.33	55.00	**86.33**	50.67
#5	Min	70.83	0.00	45.83	79.17	58.33	**87.50**	79.17
	Equi	**66.67**	0.00	29.63	70.37	40.74	**66.67**	59.26
#6	Min	69.55	77.35	**78.19**	71.74	72.72	67.18	72.46
	Equi	81.92	**83.74**	82.46	79.89	80.13	80.97	74.45
#7	Min	0.00	0.00	52.38	61.90	**66.67**	42.86	57.14
	Equi	83.14	87.30	79.37	64.10	**88.14**	83.90	75.58
#8	Min	49.10	**76.81**	70.74	72.52	53.21	61.76	55.00
	Equi	12.50	0.00	41.67	8.33	62.50	**66.67**	58.33
#9	Min	55.56	66.67	29.63	51.85	33.33	**85.19**	59.26
	Equi	73.33	0.00	40.00	60.00	**76.67**	66.67	46.67
#10	Min	53.67	73.17	68.14	43.05	**80.07**	47.17	64.57
	Equi	72.05	80.51	80.49	63.39	80.60	**83.26**	75.51
#11	Min	51.78	55.87	61.29	**73.27**	61.05	69.77	20.33
	Equi	56.06	66.99	72.67	62.45	77.37	73.53	**78.43**
#12	Min	70.37	78.38	**80.86**	73.89	78.83	80.21	25.19
	Equi	39.11	**76.44**	72.17	62.30	74.21	74.57	75.79
#13	Min	51.85	55.56	59.26	59.26	18.52	44.44	**70.37**
	Equi	66.67	0.00	29.63	44.44	22.22	**70.37**	29.63
#14	Min	74.18	80.81	**80.91**	66.16	79.16	79.51	67.24
	Equi	71.95	72.81	78.04	72.11	**79.63**	76.31	67.89
#15	Min	70.86	**76.40**	70.24	68.60	67.05	74.29	73.36
	Equi	79.57	**89.31**	74.62	72.33	72.10	81.55	79.27
#16	Min	44.44	0.00	50.00	27.78	**61.11**	27.78	50.00
	Equi	73.33	0.00	33.33	83.33	**86.67**	70.00	70.00
#17	Min	0.00	0.00	20.83	33.33	**66.67**	37.50	58.33
	Equi	41.52	53.46	59.64	**65.82**	55.07	64.27	63.73
#18	Min	29.63	29.63	**62.96**	33.33	59.26	59.26	**62.96**
	Equi	77.78	0.00	33.33	61.11	72.22	72.22	**88.89**
#19	Min	69.24	71.33	84.24	72.29	**87.81**	84.67	73.67
	Equi	66.71	**79.98**	78.11	77.65	79.73	66.21	73.85
#20	Min	70.00	48.00	64.67	68.00	61.33	**71.67**	51.67
	Equi	67.85	73.52	71.48	73.56	**77.33**	72.93	70.29
#21	Min	47.33	54.67	68.33	66.33	**71.67**	69.33	63.00
	Equi	65.83	**84.90**	76.17	78.69	72.99	80.14	72.71
#22	Min	20.00	0.00	40.00	46.67	23.33	53.33	**56.67**
	Equi	70.00	0.00	40.00	**84.00**	66.67	83.33	54.67
#23	Min	58.96	70.43	71.47	71.66	74.18	**74.87**	70.78
	Equi	67.30	76.26	**77.44**	75.33	76.75	77.15	71.59
#24	Min	54.57	63.70	58.37	53.96	**72.61**	61.96	68.03
	Equi	40.04	**74.35**	66.32	53.73	67.04	66.05	62.24
#25	Min	80.46	88.11	**88.48**	79.59	83.19	82.23	25.52
	Equi	73.07	61.33	70.57	63.57	70.19	**82.05**	66.07
#26	Min	71.67	51.67	71.00	76.67	69.67	63.00	**77.33**
	Equi	82.33	83.57	**84.00**	82.24	80.50	79.57	68.38
#27	Min	84.61	93.50	**94.86**	93.44	93.84	92.87	76.21
	Equi	76.25	80.85	**83.68**	72.62	79.92	81.34	72.20
Mínimo	**Min**	0.00	0.00	20.83	27.78	18.52	27.78	20.33
	Equi	12.50	0.00	0.00	8.33	0.00	64.27	29.63
Máximo	**Min**	84.61	93.50	94.86	93.44	93.84	92.87	79.17
	Equi	85.33	89.31	89.38	84.12	89.84	88.96	88.89
Média	**Min**	53.69	56.51	64.76	64.58	65.70	66.94	60.73
	Equi	66.26	52.91	61.94	66.97	68.88	75.30	67.34
Melhor classificador	**Min**	0	4	7	2	7	4	3
(em programas)	**Equi**	1	6	4	3	6	5	2

Tabela 19 – Melhores parâmetros dos classificadores para cada programa

ID do Programa		**KNN**	**DT**	**RF**
#1	Min	1	15	5
	Equi	11	35	15
#2	Min	1	15	5
	Equi	11	35	15
#3	Min	1	15	5
	Equi	3	35	15
#4	Min	1	15	5
	Equi	11	35	15
#5	Min	8	15	5
	Equi	9	35	15
#6	Min	1	15	5
	Equi	11	35	15
#7	Min	7	15	5
	Equi	11	35	15
#8	Min	1	15	5
	Equi	8	35	15
#9	Min	9	15	5
	Equi	10	35	15
#10	Min	1	15	5
	Equi	11	35	15
#11	Min	1	15	5
	Equi	11	35	15
#12	Min	1	15	5
	Equi	11	35	15
#13	Min	9	15	5
	Equi	9	35	15
#14	Min	1	15	5
	Equi	11	35	15
#15	Min	1	15	5
	Equi	11	35	15
#16	Min	6	15	5
	Equi	10	35	15
#17	Min	8	15	5
	Equi	11	35	15
#18	Min	9	15	5
	Equi	6	35	15
#19	Min	1	15	5
	Equi	11	35	15
#20	Min	1	15	5
	Equi	11	35	15
#21	Min	1	15	5
	Equi	11	0	15
#22	Min	10	15	5
	Equi	11	35	15
#23	Min	1	15	5
	Equi	11	35	15
#24	Min	1	15	5
	Equi	11	35	15
#25	Min	1	15	5
	Equi	11	35	15
#26	Min	1	15	5
	Equi	11	35	15
#27	Min	1	15	5
	Equi	11	35	15

Tabela 20 – Melhores classificadores para classificação de cada um dos programas e as respectivas métricas

ID do Programa		**Classificador**	**Acurácia**	**Precisão**	**Revocação**	**F1**
#1	Min	RF - 25	84.06	48.15	87.97	62.23
	Equi	RF - 65	76.54	20.54	59.74	30.56
#2	Min	DT - 85	79.65	15.24	33.78	21.01
	Equi	RF - 15	74.78	20.24	58.14	30.03
#3	Min	DT - 15	56.73	41.56	100.00	58.72
	Equi	GNB	70.19	8.82	100.00	16.22
#4	Min	RF - 5	83.73	23.08	46.15	30.77
	Equi	LDA	77.71	29.27	60.00	39.34
#5	Min	KNN - 12	78.81	10.00	37.50	15.79
	Equi	DT - 55	72.19	14.89	77.78	25.00
#6	Min	DT - 35	77.74	27.81	42.50	33.62
	Equi	SVM	84.95	26.41	63.53	37.31
#7	Min	KNN - 9	69.35	2.61	57.14	5.00
	Equi	SVM	87.30	37.50	69.77	48.78
#8	Min	KNN - 4	75.25	36.07	68.75	47.31
	Equi	KNN - 2	84.85	13.33	50.00	21.05
#9	Min	RF - 25	90.00	14.29	22.22	17.39
	Equi	DT - 75	81.05	11.76	40.00	18.18
#10	Min	RF - 65	93.56	11.11	20.00	14.29
	Equi	KNN - 23	69.92	19.27	61.05	29.29
#11	Min	DT - 15	83.51	17.24	39.47	24.00
	Equi	RF - 65	56.08	17.73	88.33	29.53
#12	Min	RF - 5	93.65	26.15	29.06	27.53
	Equi	DT - 35	91.67	12.23	24.73	16.37
#13	Min	DT - 25	92.49	25.00	22.22	23.53
	Equi	DT - 85	71.68	12.96	77.78	22.22
#14	Min	SVM	65.32	26.86	80.16	40.24
	Equi	KNN - 36	69.60	32.58	83.45	46.87
#15	Min	DT - 75	87.18	8.40	35.71	13.61
	Equi	KNN - 1	79.41	9.09	57.58	15.70
#16	Min	GNB	80.79	10.34	50.00	17.14
	Equi	KNN - 19	64.90	14.75	90.00	25.35
#17	Min	RF - 15	97.23	25.00	12.50	16.67
	Equi	KNN - 34	64.27	25.62	44.29	32.46
#18	Min	KNN - 14	77.24	8.06	55.56	14.08
	Equi	RF - 45	83.58	10.42	83.33	18.52
#19	Min	DT - 75	88.25	19.19	82.61	31.15
	Equi	KNN - 13	69.65	18.61	59.72	28.38
#20	Min	RF - 85	91.03	3.49	25.00	6.12
	Equi	RF - 35	82.94	17.48	30.49	22.22
#21	Min	DT - 15	86.85	9.59	50.00	16.09
	Equi	KNN - 17	54.59	10.26	80.00	18.18
#22	Min	LR	70.91	13.46	70.00	22.58
	Equi	DT - 15	86.67	26.09	54.55	35.29
#23	Min	RF - 95	84.94	11.20	32.22	16.62
	Equi	KNN - 7	69.41	34.96	50.49	41.31
#24	Min	SVM	75.99	18.30	51.85	27.05
	Equi	DT - 25	73.45	15.19	42.11	22.33
#25	Min	KNN - 40	79.13	36.97	46.81	41.31
	Equi	DT - 95	86.64	17.28	51.85	25.93
#26	Min	RF - 95	79.67	19.64	68.75	30.56
	Equi	DT - 45	87.40	42.22	79.17	55.07
#27	Min	LDA	52.42	10.53	73.73	18.43
	Equi	LDA	72.05	15.22	64.91	24.65
Minimum	**Min**		52.42	2.61	12.50	5.00
	Equi		54.59	8.82	24.73	15.70
Maximum	**Min**		97.23	48.15	100.00	62.23
	Equi		91.67	42.22	100.00	55.07
Mean	**Min**		80.57	19.24	49.69	25.66
	Equi		75.68	19.81	63.07	28.75

Tabela 21 – Valor de significância *p-value* para o teste de *Shapiro-Wilk*

Teste de Shapiro-Wilk para normalidade (p-value)			
Modelo Preditivo	**Classificador**	**p-Value**	**É paramétrico?**
Mutantes Minimais	KNN	0.4199	Sim
	DT	0.9853	Sim
	RF	0.2213	Sim
	SVM	0.1169	Sim
	LDA	0.1117	Sim
	LR	0.2992	Sim
	GNB	0.0028	**Não**
Mutantes Equivalentes	KNN	0.9416	Sim
	DT	0.3738	Sim
	RF	0.4981	Sim
	SVM	0.6204	Sim
	LDA	0.4726	Sim
	LR	0.3298	Sim
	GNB	0.3309	Sim

Tabela 22 – Valor de significância *p-value* para diferença entre classificadores - Mutantes Minimais

	KNN	**DT**	**RF**	**SVM**	**LDA**	**LR**	**GNB**
KNN	—	—	—	—	—	—	—
DT	0.0000	—	—	—	—	—	—
RF	0.0000	0.0000	—	—	—	—	—
SVM	0.0000	0.0000	0.0000	—	—	—	—
LDA	0.0000	0.0000	0.0000	**0.3575**	—	—	—
LR	0.0000	0.0000	0.0000	0.0013	0.0490	—	—
GNB	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	—

Tabela 23 – Valor de significância *p-value* para diferença entre classificadores - Mutantes Equivalentes

	KNN	**DT**	**RF**	**SVM**	**LDA**	**LR**	**GNB**
KNN	—	—	—	—	—	—	—
DT	**0.1752**	—	—	—	—	—	—
RF	0.0000	0.0000	—	—	—	—	—
SVM	0.0000	0.0000	0.0000	—	—	—	—
LDA	0.0000	0.0000	0.0000	0.0000	—	—	—
LR	0.0000	0.0000	0.0000	0.0000	0.0052	—	—
GNB	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	—

CAPÍTULO

7

CONCLUSÕES

7.1 Visão Geral

Este capítulo faz uma revisão e uma apresentação final sobre todo o trabalho desenvolvido neste projeto.

O Teste de Mutação é um critério de teste da técnica de teste baseado em defeitos. Tem como principal objetivo medir a adequação de um dado conjunto de casos de teste. O apontamento dos elementos que devem ser testados ao desenvolver casos de teste, o fornecimento de critérios para o término do teste e a quantificação da adequação do conjunto de casos de teste são as principais vantagens da utilização do Teste de Mutação (CHEKAM *et al.*, 2017). Ainda que seja uma técnica altamente eficaz para a melhoria dos casos de teste e consequente identificação de defeitos, o teste de mutação ainda sofre com desafios importantes que dificultam e até impedem sua utilização em larga escala. A identificação de mutantes equivalentes e a geração de um grande número de mutantes, são os principais desafios concernentes ao Teste de Mutação. Identificar mutantes equivalentes é reconhecidamente um problema indecidível (BUDD; ANGLUIN, 1982) e portanto, requer intervenção manual, entretanto, leva-se muito tempo para identificar um mutante equivalente, aproximadamente 14 minutos e 30 segundos por mutante (SCHULER; ZELLER, 2013). Outro problema fundamental do teste de mutação é o grande número de mutantes gerados, ainda que para programas simples e pequenos (DELAMARO; CHAIM; MALDONADO, 2018). Muitos dos mutantes que são gerados são redundantes, e portanto, não contribuem para a efetividade do Teste de Mutação e apenas inflam o score de mutação, elevam o custo computacional de executar todos esses mutantes e dificultam a análise do testador que deverá gerar casos de teste que matem todos esses mutantes. O conjunto de mutantes minimais é um conjunto de mutantes que não contém nenhum mutante redundante, isto é, todos os mutantes deste conjunto são essenciais para a manutenção do escore de mutação. De acordo com Delamaro, Chaim e Maldonado (2018), o conjunto de mutantes minimais é significativamente menor que o conjunto total de mutantes e pode corresponder apenas a 1% do conjunto total de mutantes.

7.2 Contribuições

Diante dos problemas supracitados referentes ao Teste de Mutação, este trabalho de mestrado visou reduzir o custo do Teste de Mutação através do emprego de técnicas de Aprendizado de Máquina que poderiam identificar mutantes minimais e mutantes equivalentes. Dessa forma, este trabalho tem diversas contribuições para a literatura em Teste de Software e as principais contribuições são:

- Diminuição do custo do teste de mutação através da criação de modelos preditivos que identifiquem mutantes minimais e equivalentes. Subseção 7.2.1;
- Apontamento de um conjunto de algoritmos classificadores que podem ser utilizados para criação de modelos preditivos que identifiquem mutantes minimais e equivalentes. Subseção 7.2.2;
- Apontamento de um conjunto de características dos programas que podem ser utilizadas na criação de modelos preditivos que identifiquem mutantes minimais e equivalentes. Subseção 7.2.3.

7.2.1 Redução do custo do teste de mutação utilizando aprendizado de máquina

Como pôde ser visto na Seção 6.1, alguns modelos preditivos desenvolvidos neste trabalho, obtiveram um desempenho satisfatório em classificar mutantes minimais e equivalentes.

Ao analisar os modelos preditivos criados com base nos mutantes de todos os programas, os classificadores RF, DT e KNN obtiveram os melhores resultados em classificar mutantes minimais e equivalentes. O classificador RF obteve 82.22 de F1-Score para classificar mutantes minimais e 80.75 em classificar mutantes equivalentes. O DT alcançou 80.58 e 78.25 de F1-Score ao classificar mutantes minimais e equivalentes, respectivamente. O terceiro melhor algoritmo, o KNN, alcançou 73.39 em classificar mutantes minimais e 78.02 em classificar mutantes equivalentes.

Ao construir modelos preditivos criados com base apenas nos mutantes dos programas de forma individual, os valores máximos de F1-Score para classificar mutantes minimais, foi de 62.23 com o RF no programa **cal** (#1) e 55.07 para classificar mutantes equivalentes, com o classificador DT no programa **twoPred** (#26).

7.2.2 Conjunto de classificadores que podem ser usados para classificar mutantes minimais e equivalentes

Os algoritmos apontados na Seção 4.3 podem servir de um ponto de partida para novos pesquisadores ao procurar resolver o problema do custo do teste de mutação com algoritmos de

aprendizado de máquina. Entre os 7 algoritmos utilizados neste trabalho, existiram aqueles que obtiveram um resultado satisfatório e com a utilização de hiperparâmetros poderiam alcançar resultados ainda melhores que os apresentados neste trabalho e avançar o estado da arte nesta área de pesquisa.

7.2.3 Conjunto de propriedades e características estruturais podem ser usados para classificar mutantes minimais e equivalentes

A seleção das propriedades utilizadas para a construção dos modelos preditivos é uma tarefa importante no desenvolvimento de modelos preditivos, e as propriedades listadas e exemplificadas na Seção 5.3 podem servir como um ponto de partida em novas pesquisas nesta área do conhecimento. As propriedades utilizadas neste trabalho, derivaram de um trabalho anterior (DELAMARO; CHAIM; MALDONADO, 2018), e é possível que as 11 propriedades aqui utilizadas possam servir de base para a criação de novos estudos.

7.3 Dificuldades e limitações

A seguir, são apresentadas algumas dificuldades e limitações identificadas na condução e execução do presente trabalho:

- **Limitações referentes ao conjunto de programas utilizados**: Para o desenvolvimento e análise da abordagem proposta neste trabalho, era necessário obter um conjunto de programas com o conjunto adequado de testes para posterior obtenção do conjunto de mutantes minimais e do conjunto de mutantes equivalentes. Dessa forma, não foi possível utilizar quaisquer programas para o desenvolvimento da abordagem, mas sim, apenas aqueles em que era conhecido o conjunto adequado de testes.
- **Limitações referentes à ferramenta Proteum**: Como a abordagem proposta foi instanciada utilizando programas escritos na linguagem C e a ferramenta utilizada para o suporte ao teste de mutação foi a Proteum (DELAMARO; MALDONADO; VINCENZI, 2001), todas as limitações inerentes à Proteum também foram incorporadas à abordagem desenvolvida.

7.4 Considerações finais

Esta dissertação apresentou uma abordagem para redução do custo de aplicação do teste de mutação. Alguns dos problemas fundamentais do teste de mutação referem-se à geração de muitos mutantes redundantes e diversos mutantes equivalentes. Os dois problemas mencionados, dificultam a aplicação do teste de mutação em larga escala na indústria de desenvolvimento de software. Dessa forma, a abordagem desenvolvida nesta dissertação tem como objetivos principais, identificar o conjunto de mutantes minimais, isto é, o conjunto de mutantes únicos,

sem mutantes redundantes e também, identificar o conjunto de mutantes equivalentes de um conjunto de mutantes.

Para atingir o objetivo especificado, a abordagem desenvolvida utiliza algoritmos de aprendizado de máquina para identificar os mutantes minimais e equivalentes. Os algoritmos de aprendizado de máquina mencionados, têm o intuito de aprender com as informações já conhecidas (dados do passado) de um determinado conjunto de dados, para predizer informações ainda desconhecidas (dados do futuro) de um outro determinado conjunto de dados, dessa forma, a partir do desenvolvimento da abordagem, onde foram inseridas diversas informações já conhecidas de um conjunto de dados é possível predizer algumas informações desconhecidas de um outro determinado conjunto de dados.

Espera-se que os resultados apresentados nesta dissertação instiguem outros pesquisadores a dedicar esforços em aprimorar e estender a abordagem aqui desenvolvida, para que a área de Teste e Validação de Software, mais especificamente, a área de Teste de Mutação possa ser utiliza mais amplamente em projetos de software.

REFERÊNCIAS

ACREE, A. T.; BUDD, T. A.; DEMILLO, R. A.; LIPTON, R. J.; SAYWARD, F. G. **Mutation Analysis**. [S.l.], 1979. Citado na página 47.

ACREE JR., A. T. **On Mutation**. Tese (Doutorado) — Georgia Institute of Technology, Atlanta, GA, USA, 1980. AAI8107280. Citado na página 47.

AMMANN, P.; DELAMARO, M.; OFFUTT, J. Establishing theoretical minimal sets of mutants. In: **ICST**. [s.n.], 2014. p. 21–30. Cited By 72. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84903164798&doi=10.1109%2fICST.2014.13&partnerID=40&md5=fa08a86d598c7e4410cc0ccb9208cbc8>. Citado nas páginas 43, 45, 46, 53, 54, 69 e 89.

AMMANN, P.; OFFUTT, J. **Introduction to software testing**. [S.l.]: Cambridge University Press, 2016. Citado na página 37.

ANDRADE, S. a. A.; BRITO JUNIOR, C.; JúNIOR, M.; MARCIEL, A. C.; ABDALLA, G.; DELAMARO, M. E. Analyzing the effectiveness of one-op mutation against the minimal set of mutants. In: **Proceedings of the IV Brazilian Symposium on Systematic and Automated Software Testing**. New York, NY, USA: ACM, 2019. (SAST 2019), p. 22–31. ISBN 978-1-4503-7648-8. Disponível em: <http://doi.acm.org/10.1145/3356317.3356321>. Citado nas páginas 69 e 89.

ANDRÉS, C.; MERAYO, M. G.; MOLINERO, C. Advantages of mutation in passive testing: An empirical study. In: IEEE. **2009 International Conference on Software Testing, Verification, and Validation Workshops**. [S.l.], 2009. p. 230–239. Citado na página 48.

ANDRÉS, C.; MERAYO, M. G.; NÚÑEZ, M. Passive testing of stochastic timed systems. In: IEEE. **2009 International Conference on Software Testing Verification and Validation**. [S.l.], 2009. p. 71–80. Citado na página 48.

____. Formal passive testing of timed systems: theory and tools. **Software Testing, Verification and Reliability**, Wiley Online Library, v. 22, n. 6, p. 365–405, 2012. Citado na página 48.

ANDREWS, J. H.; BRIAND, L. C.; LABICHE, Y. Is mutation an appropriate tool for testing experiments? In: ACM. **Proceedings of the 27th international conference on Software engineering**. [S.l.], 2005. p. 402–411. Citado na página 48.

ANDREWS, J. H.; ZHANG, Y. General test result checking with log file analysis. **IEEE Transactions on Software Engineering**, IEEE, v. 29, n. 7, p. 634–648, 2003. Citado na página 48.

ARCAINI, P.; GARGANTINI, A.; RICCOBENE, E. Mutrex: A mutation-based generator of fault detecting strings for regular expressions. In: IEEE. **2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)**. [S.l.], 2017. p. 87–96. Citado na página 48.

BARBOSA, E. F.; CHAIM, M. L.; VINCENZI, A. M. R.; DELAMARO, M. E.; JINO, M.; MALDONADO, J. C. Introdução ao Teste de Software – Capítulo 4 - Teste Estrutural. In: **Introdução ao Teste de Software**. [S.l.]: Elsevier Editora Ltda., 2017. p. 47–76. ISBN 9788535283525. Citado na página 30.

BARBOSA, E. F.; MALDONADO, J. C.; VINCENZI, A. M. R. Toward the determination of sufficient mutant operators for c. **Software Testing, Verification and Reliability**, Wiley Online Library, v. 11, n. 2, p. 113–136, 2001. Citado nas páginas 41, 42 e 44.

BARBOSA, E. F.; VINCENZI, A. M.; MALDONADO, J. C. Uma contribuição para a determinação de um conjunto essencial de operadores de mutação no teste de programas c. **XII Simpósio Brasileiro de Engenharia de Software (SBES 98)**, p. 103–120, 1998. Citado nas páginas 17, 42 e 44.

BARUCH, O.; KATZ, S. Partially interpreted schemas for csp programming. **Science of Computer Programming**, Elsevier, v. 10, n. 1, p. 1–18, 1988. Citado na página 39.

BASILI, V. R.; ROMBACH, H. D. The TAME project: Towards improvement-oriented software environments. **IEEE Transactions on software engineering**, IEEE, v. 14, n. 6, p. 758–773, 1988. Citado na página 64.

BERTOLINO, A. Software Testing Research and Practice. In: BÖRGER, E.; GARGANTINI, A.; RICCOBENE, E. (Ed.). **Abstract State Machines 2003**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. p. 1–21. ISBN 978-3-540-36498-6. Citado na página 29.

BERTOLINO, A.; DAOUDAGH, S.; LONETTI, F.; MARCHETTI, E. Xacmut: Xacml 2.0 mutants generator. In: IEEE. **2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops**. [S.l.], 2013. p. 28–33. Citado na página 49.

BRADBURY, J. S.; CORDY, J. R.; DINGEL, J. Exman: A generic and customizable framework for experimental mutation analysis. In: IEEE. **Second Workshop on Mutation Analysis (Mutation 2006-ISSRE Workshops 2006)**. [S.l.], 2006. p. 4–4. Citado na página 47.

BUDD, T.; HESS, R.; SAYWARD, F. Exper implementor's guide. **Yale University, New Haven, Connecticut, Technique Report**, 1980. Citado na página 47.

BUDD, T.; SAYWARD, F. Users guide to the pilot mutation system. **Yale University, New Haven, Connecticut, Technique Report**, v. 114, 1977. Citado na página 48.

BUDD, T. A. Mutation analysis of program test data. 1981. Citado na página 47.

BUDD, T. A.; ANGLUIN, D. Two notions of correctness and their relation to testing. **Acta Informatica**, Springer, v. 18, n. 1, p. 31–45, 1982. Citado nas páginas 51 e 95.

BUDD, T. A.; DEMILLO, R. A.; LIPTON, R. J.; SAYWARD, F. G. The design of a prototype mutation system for program testing. In: **Proceedings of the AFIPS National Computer Conference**. [S.l.: s.n.], 1978. v. 74, p. 623–627. Citado na página 48.

CHEKAM, T. T.; PAPADAKIS, M.; BISSYANDÉ, T.; TRAON, Y. L.; SEN, K. Selecting fault revealing mutants. **arXiv preprint arXiv:1803.07901**, 2018. Citado na página 52.

CHEKAM, T. T.; PAPADAKIS, M.; TRAON, Y. L.; HARMAN, M. An empirical study on mutation, statement and branch coverage fault revelation that avoids the unreliable clean program assumption. In: IEEE. **2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)**. [S.l.], 2017. p. 597–608. Citado nas páginas 31, 35 e 95.

CHEVALLEY, P.; THÉVENOD-FOSSE, P. A mutation analysis tool for java programs. **International journal on software tools for technology transfer**, Springer, v. 5, n. 1, p. 90–103, 2003. Citado na página 47.

CHUSHO, T. Test data selection and quality estimation based on the concept of essential branches for path testing. **IEEE Transactions on Software Engineering**, IEEE, n. 5, p. 509–517, 1987. Citado nas páginas 66 e 68.

COLES, H.; LAURENT, T.; HENARD, C.; PAPADAKIS, M.; VENTRESQUE, A. Pit: a practical mutation testing tool for java. In: ACM. **Proceedings of the 25th International Symposium on Software Testing and Analysis**. [S.l.], 2016. p. 449–452. Citado na página 48.

CONFESSIONS of a Ruby Sadist sudo gem install heckle. 2013. <http://ruby.sadi.st/Heckle.html>. (Accessed on 03/07/2019). Citado na página 47.

CROUZET, Y.; WAESELYNCK, H.; LUSSIER, B.; POWELL, D. The sesame experience: from assembly languages to declarative models. In: IEEE. **Second Workshop on Mutation Analysis (Mutation 2006-ISSRE Workshops 2006)**. [S.l.], 2006. p. 7–7. Citado na página 49.

DADEAU, F.; HÉAM, P.-C.; KHEDDAM, R. Mutation-based test generation from security protocols in hlpsl. In: IEEE. **2011 Fourth IEEE International Conference on Software Testing, Verification and Validation**. [S.l.], 2011. p. 240–248. Citado na página 47.

DAN, H.; HIERONS, R. M. Smt-c: A semantic mutation testing tools for c. In: IEEE. **2012 IEEE Fifth International Conference on Software Testing, Verification and Validation**. [S.l.], 2012. p. 654–663. Citado na página 49.

DELAMARE, R.; BAUDRY, B.; GHOSH, S.; TRAON, Y. L. A test-driven approach to developing pointcut descriptors in aspectj. In: IEEE. **2009 International Conference on Software Testing Verification and Validation**. [S.l.], 2009. p. 376–385. Citado na página 46.

DELAMARE, R.; BAUDRY, B.; TRAON, Y. L. Ajmutator: A tool for the mutation analysis of aspectj pointcut descriptors. In: IEEE. **2009 International Conference on Software Testing, Verification, and Validation Workshops**. [S.l.], 2009. p. 200–204. Citado na página 46.

DELAMARO, M.; CHAIM, M. L.; MALDONADO, J. C. Where are the minimal mutants? In: **Proceedings of the XXXII Brazilian Symposium on Software Engineering**. New York, NY, USA: ACM, 2018. (SBES '18), p. 190–195. ISBN 978-1-4503-6503-1. Disponível em: <http://doi.acm.org/10.1145/3266237.3266241>. Citado nas páginas 39, 63, 66, 89, 95 e 97.

DELAMARO, M.; JINO, M.; MALDONADO, J. **Introdução ao Teste de Software**. [S.l.]: Elsevier Editora Ltda., 2017. ISBN 9788535283525. Citado nas páginas 29 e 46.

DELAMARO, M. E.; BARBOSA, E. F.; VINCENZI, A. M. R.; MALDONADO, J. C. Introdução ao Teste de Software – Capítulo 5 - Teste de Mutação. In: **Introdução ao Teste de Software**. [S.l.]: Elsevier Editora Ltda., 2017. p. 77–116. ISBN 9788535283525. Citado na página 31.

DELAMARO, M. E.; DENG, L.; DURELLI, V. H. S.; LI, N.; OFFUTT, J. Experimental evaluation of sdl and one-op mutation for c. In: IEEE. **2014 IEEE Seventh International Conference on Software Testing, Verification and Validation**. [S.l.], 2014. p. 203–212. Citado nas páginas 43, 44, 69 e 89.

DELAMARO, M. E.; MALDONADO, J. C. Proteum-a tool for the assessment of test adequacy for c programs. In: **Proceedings of the Conference on Performability in Computing Systems (PCS'96)**. [S.l.: s.n.], 1996. p. 79–95. Citado na página 68.

DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. Introdução ao Teste de Software – Capítulo 1 - Conceitos Básicos. In: **Introdução ao Teste de Software**. [S.l.]: Elsevier Editora Ltda., 2017. p. 1–8. ISBN 9788535283525. Citado na página 30.

DELAMARO, M. E.; MALDONADO, J. C.; VINCENZI, A. M. R. Proteum/im 2.0: An integrated mutation testing environment. In: **Mutation testing for the new century**. [S.l.]: Springer, 2001. p. 91–101. Citado nas páginas 36, 48 e 97.

DELAMARO, M. E.; OFFUTT, J.; AMMANN, P. Designing deletion mutation operators. In: IEEE. **2014 IEEE Seventh International Conference on Software Testing, Verification and Validation**. [S.l.], 2014. p. 11–20. Citado nas páginas 69 e 89.

DELGADO-PÉREZ, P.; MEDINA-BULO, I.; PALOMO-LOZANO, F.; GARCÍA-DOMÍNGUEZ, A.; DOMÍNGUEZ-JIMÉNEZ, J. J. Assessment of class mutation operators for c++ with the mucpp mutation system. **Information and Software Technology**, Elsevier, v. 81, p. 169–184, 2017. Citado na página 48.

DEMILLO, R.; GUINDI, D.; KING, K.; MCCRACKEN, W.; MATHUR, A.; OFFUTT, A. An overview of the mothra software testing environment. **Purdue University, West Lafayette, Indiana, Technique Report SERC-TR-3-P**, 1987. Citado na página 47.

DEMILLO, R. A.; GUINDI, D. S.; MCCRACKEN, W.; OFFUTT, A. J.; KING, K. An extended overview of the mothra software testing environment. In: IEEE. **[1988] Proceedings. Second Workshop on Software Testing, Verification, and Analysis**. [S.l.], 1988. p. 142–151. Citado na página 47.

DEMILLO, R. A.; LIPTON, R. J.; SAYWARD, F. G. Hints on test data selection: Help for the practicing programmer. **Computer**, v. 11, n. 4, p. 34–41, April 1978. ISSN 0018-9162. Citado na página 35.

DEMILLO, R. A.; OFFUTT, A. J. Constraint-based automatic test data generation. **IEEE Transactions on Software Engineering**, v. 17, n. 9, p. 900–910, Sep. 1991. ISSN 0098-5589. Citado na página 31.

DEREZIŃSKA, A.; HAŁAS, K. Analysis of mutation operators for the python language. In: SPRINGER. **Proceedings of the Ninth International Conference on Dependability and Complex Systems DepCoS-RELCOMEX. June 30–July 4, 2014, Brunów, Poland**. [S.l.], 2014. p. 155–164. Citado na página 48.

DEREZINSKA, A.; KOWALSKI, K. Object-oriented mutation applied in common intermediate language programs originated from c. In: IEEE. **2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops**. [S.l.], 2011. p. 342–350. Citado na página 47.

DEREZINSKA, A.; SZUSTEK, A. Tool-supported advanced mutation approach for verification of c# programs. In: IEEE. **2008 Third International Conference on Dependability of Computer Systems DepCoS-RELCOMEX**. [S.l.], 2008. p. 261–268. Citado na página 47.

DEVROEY, X.; PERROUIN, G.; PAPADAKIS, M.; LEGAY, A.; SCHOBBENS, P.-Y.; HEYMANS, P. Featured model-based mutation analysis. In: ACM. **Proceedings of the 38th International Conference on Software Engineering**. [S.l.], 2016. p. 655–666. Citado na página 49.

DO, H.; ROTHERMEL, G. On the use of mutation faults in empirical assessments of test case prioritization techniques. **IEEE Transactions on Software Engineering**, IEEE, v. 32, n. 9, p. 733–752, 2006. Citado na página 46.

DODGE, Y. **The concise encyclopedia of statistics**. [S.l.]: Springer Science & Business Media, 2008. Citado na página 85.

DOMÍNGUEZ-JIMÉNEZ, J. J.; ESTERO-BOTARO, A.; MEDINA-BULO, I. A framework for mutant genetic generation for ws-bpel. In: SPRINGER. **International Conference on Current Trends in Theory and Practice of Computer Science**. [S.l.], 2009. p. 229–240. Citado na página 47.

DURELLI, V. H.; DURELLI, R. S.; BORGES, S. S.; ENDO, A. T.; ELER, M. M.; DIAS, D. R.; GUIMARÃES, M. P. Machine Learning Applied to Software Testing: A Systematic Mapping Study. **IEEE Transactions on Reliability**, v. 68, n. 3, p. 1189–1212, 2019. Citado nas páginas 56 e 89.

ELLIMS, M.; INCE, D.; PETRE, M. The csaw c mutation tool: Initial results. In: IEEE. **Testing: Academic and Industrial Conference Practice and Research Techniques-MUTATION (TAICPART-MUTATION 2007)**. [S.l.], 2007. p. 185–192. Citado na página 47.

ERNST, M. D.; COCKRELL, J.; GRISWOLD, W. G.; NOTKIN, D. Dynamically discovering likely program invariants to support program evolution. **IEEE Transactions on Software Engineering**, v. 27, n. 2, p. 99–123, 2001. Citado na página 52.

FABBRI, S. C. P. F.; VINCENZI, A. M. R.; MALDONADO, J. C. Introdução ao Teste de Software – Capítulo 2 - Teste Funcional. In: **Introdução ao Teste de Software**. [S.l.]: Elsevier Editora Ltda., 2017. p. 9–26. ISBN 9788535283525. Citado na página 30.

FENG, X.; MARR, S.; O'CALLAGHAN, T. Estp: an experimental software testing platform. In: IEEE. **Testing: Academic & Industrial Conference-Practice and Research Techniques (taic part 2008)**. [S.l.], 2008. p. 59–63. Citado na página 47.

FERNANDES, L.; RIBEIRO, M.; CARVALHO, L.; GHEYI, R.; MONGIOVI, M.; SANTOS, A.; CAVALCANTI, A.; FERRARI, F.; MALDONADO, J. C. Avoiding useless mutants. In: **Proceedings of the 16th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences - GPCE 2017**. New York, New York, USA: ACM Press, 2017. p. 187–198. ISBN 9781450355247. Citado na página 39.

FERRARI, F. C.; PIZZOLETO, A. V.; OFFUTT, J. A systematic review of cost reduction techniques for mutation testing: Preliminary results. In: IEEE. **2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)**. [S.l.], 2018. p. 1–10. Citado na página 40.

GHOSH, S.; GOVINDARAJAN, P.; MATHUR, A. P. Tds: a tool for testing distributed component-based applications. In: **Mutation testing for the new century**. [S.l.]: Springer, 2001. p. 103–112. Citado na página 49.

GITHUB - manuelpichler/mutateme: A PHP 5.3+ Mutation Testing framework. 2010. <https://github.com/manuelpichler/mutateme>. (Accessed on 02/28/2019). Citado na página 48.

GITHUB - mbj/mutant: Mutation testing for Ruby. 2019. <https://github.com/mbj/mutant>. (Accessed on 02/28/2019). Citado na página 48.

GLIGORIC, M.; BADAME, S.; JOHNSON, R. Smutant: a tool for type-sensitive mutation testing in a dynamic language. In: ACM. **Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering**. [S.l.], 2011. p. 424–427. Citado na página 49.

GLIGORIC, M.; JAGANNATH, V.; MARINOV, D. Mutmut: Efficient exploration for mutation testing of multithreaded code. In: IEEE. **2010 Third International Conference on Software Testing, Verification and Validation**. [S.l.], 2010. p. 55–64. Citado na página 48.

GLIGORIC, M.; ZHANG, L.; PEREIRA, C.; POKAM, G. Selective mutation testing for concurrent code. In: ACM. **Proceedings of the 2013 International Symposium on Software Testing and Analysis**. [S.l.], 2013. p. 224–234. Citado na página 47.

GRÜN, B. J. M.; SCHULER, D.; ZELLER, A. The Impact of Equivalent Mutants. In: **2009 International Conference on Software Testing, Verification, and Validation Workshops**. [S.l.: s.n.], 2009. p. 192–199. Citado na página 52.

HANKS, J. M. **Testing COBOL programs by mutation**. Tese (Doutorado) — Georgia Institute of Technology, 1980. Citado na página 47.

HENARD, C.; PAPADAKIS, M.; TRAON, Y. L. Mutalog: A tool for mutating logic formulas. In: IEEE. **2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops**. [S.l.], 2014. p. 399–404. Citado na página 48.

HIERONS, R. M.; MERAYO, M. G.; NÚÑEZ, M. Mutation testing. In: **Encyclopedia of Software Engineering Three-Volume Set (Print)**. [S.l.]: Auerbach Publications, 2010. p. 594–602. Citado na página 37.

HOWDEN, W. E. Weak mutation testing and completeness of test sets. **IEEE Transactions on Software Engineering**, SE-8, n. 4, p. 371–379, July 1982. ISSN 0098-5589. Citado na página 39.

HULTEN, G. Building Intelligent Systems. Springer, 2018. Citado na página 60.

JABBARVAND, R.; MALEK, S. μdroid: an energy-aware mutation testing framework for android. In: ACM. **Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering**. [S.l.], 2017. p. 208–219. Citado na página 46.

JESTER. 2005. <http://jester.sourceforge.net/>. (Accessed on 02/28/2019). Citado nas páginas 47 e 48.

JIA, Y.; HARMAN, M. Milu: A customizable, runtime-optimized higher order mutation testing tool for the full c language. In: IEEE. **Testing: Academic & Industrial Conference-Practice and Research Techniques (taic part 2008)**. [S.l.], 2008. p. 94–98. Citado na página 47.

____. An analysis and survey of the development of mutation testing. **IEEE Transactions on Software Engineering**, v. 37, n. 5, p. 649–678, Sep. 2011. ISSN 0098-5589. Citado nas páginas 40 e 46.

JOHNSON, R. A.; BHATTACHARYYA, G. K. **Statistics: Principles and Methods**. Wiley, 2019. ISBN 9781119497110. Disponível em: <https://books.google.com.br/books?id=0k6MDwAAQBAJ>. Citado na página 86.

JUMBLE. 2015. <http://jumble.sourceforge.net/>. (Accessed on 02/28/2019). Citado na página 47.

JUST, R. The major mutation framework: Efficient and scalable mutation analysis for java. In: ACM. **Proceedings of the 2014 International Symposium on Software Testing and Analysis**. [S.l.], 2014. p. 433–436. Citado na página 47.

JUST, R.; KURTZ, B.; AMMANN, P. Inferring mutant utility from program context. In: ACM. **Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis**. [S.l.], 2017. p. 284–294. Citado nas páginas 53 e 54.

KAPFHAMMER, G. M.; MCMINN, P.; WRIGHT, C. J. Search-based testing of relational schema integrity constraints across multiple database management systems. In: IEEE. **2013 ieee sixth international conference on software testing, verification and validation**. [S.l.], 2013. p. 31–40. Citado na página 49.

KIM, S.-W.; HARROLD, M. J.; KWON, Y.-R. Mugamma: Mutation analysis of deployed software to increase confidence and assist evolution. In: IEEE. **Second Workshop on Mutation Analysis (Mutation 2006-ISSRE Workshops 2006)**. [S.l.], 2006. p. 10–10. Citado na página 48.

KING, K.; OFFUTT, A. A fortran language system for mutation-based software testing. **Software: Practice and Experience**, v. 21, n. 7, p. 685–718, 1991. Cited By 214. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-0026185573&doi=10.1002%2fspe.4380210704&partnerID=40&md5=63bb76053636c342797821b6afb78609>. Citado na página 113.

KINTIS, M.; PAPADAKIS, M.; MALEVRIS, N. Evaluating mutation testing alternatives: A collateral experiment. In: IEEE. **2010 Asia Pacific Software Engineering Conference**. [S.l.], 2010. p. 300–309. Citado na página 49.

KRENN, W.; SCHLICK, R.; TIRAN, S.; AICHERNIG, B.; JOBSTL, E.; BRANDL, H. Momut:: Uml model-based mutation testing for uml. In: IEEE. **2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)**. [S.l.], 2015. p. 1–8. Citado na página 47.

KURTZ, B.; AMMANN, P.; DELAMARO, M. E.; OFFUTT, J.; DENG, L. Mutant subsumption graphs. In: **2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops**. [S.l.: s.n.], 2014. p. 176–185. Citado nas páginas 43 e 53.

KURTZ, B.; AMMANN, P.; OFFUTT, J.; DELAMARO, M. E.; KURTZ, M.; GöKcE, N. Analyzing the validity of selective mutation with dominator mutants. In: **Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering**. New York, NY, USA: ACM, 2016. (FSE 2016), p. 571–582. ISBN 978-1-4503-4218-6. Disponível em: <http://doi.acm.org/10.1145/2950290.2950322>. Citado nas páginas 53 e 54.

KUSANO, M.; WANG, C. Ccmutator: A mutation generator for concurrency constructs in multithreaded c/c++ applications. In: IEEE PRESS. **Proceedings of the 28th IEEE/ACM**

International Conference on Automated Software Engineering. [S.l.], 2013. p. 722–725. Citado na página 47.

LE, D.; ALIPOUR, M. A.; GOPINATH, R.; GROCE, A. Mucheck: An extensible tool for mutation testing of haskell programs. In: ACM. **Proceedings of the 2014 international symposium on software testing and analysis**. [S.l.], 2014. p. 429–432. Citado na página 47.

LI, N.; WEST, M.; ESCALONA, A.; DURELLI, V. H. Mutation testing in practice using ruby. In: IEEE. **2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)**. [S.l.], 2015. p. 1–6. Citado na página 48.

LINARES-VÁSQUEZ, M.; BAVOTA, G.; TUFANO, M.; MORAN, K.; PENTA, M. D.; VENDOME, C.; BERNAL-CÁRDENAS, C.; POSHYVANYK, D. Enabling mutation testing for android apps. In: ACM. **Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering**. [S.l.], 2017. p. 233–244. Citado na página 47.

LIPTON, R. J.; SAYWARD, F. G. The status of research on program mutation. In: FORT LAUDERDALE FLORIDA, USA. **Digest for the Workshop on Software Testing and Test Documentation**. [S.l.], 1978. p. 355–373. Citado na página 48.

LOURIDAS, P.; EBERT, C. Machine Learning. **{IEEE} Software**, v. 33, n. 5, p. 110–115, 2016. Disponível em: <https://doi.org/10.1109/MS.2016.114>. Citado nas páginas 55 e 56.

MA, Y.-S.; OFFUTT, J.; KWON, Y. R. Mujava: an automated class mutation system. **Software Testing, Verification and Reliability**, Wiley Online Library, v. 15, n. 2, p. 97–133, 2005. Citado na página 48.

MA, Y.-S.; OFFUTT, J.; KWON, Y.-R. Mujava: A mutation system for java. In: **Proceedings of the 28th International Conference on Software Engineering**. New York, NY, USA: ACM, 2006. (ICSE '06), p. 827–830. ISBN 1-59593-375-1. Disponível em: <http://doi.acm.org/10.1145/1134285.1134425>. Citado nas páginas 48 e 115.

MACIEL, A. C. **Avaliação da qualidade de oráculos de teste utilizando mutação**. Tese (Doutorado) — Universidade de São Paulo, 2017. Citado na página 49.

MADEYSKI, L.; ORZESZYNA, W.; TORKAR, R.; JOZALA, M. Overcoming the equivalent mutant problem: A systematic literature review and a comparative experiment of second order mutation. **IEEE Transactions on Software Engineering**, IEEE, v. 40, n. 1, p. 23–42, 2014. Citado nas páginas 31, 36 e 53.

MADEYSKI, L.; RADYK, N. Judy–a mutation testing tool for java. **IET software**, IET, v. 4, n. 1, p. 32–42, 2010. Citado na página 47.

MADIRAJU, P.; NAMIN, A. S. Paraμ–a partial and higher-order mutation tool with concurrency operators. In: IEEE. **2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops**. [S.l.], 2011. p. 351–356. Citado na página 48.

MATEO, P. R.; USAOLA, M. P.; OFFUTT, J. Mutation at system and functional levels. In: IEEE. **2010 Third International Conference on Software Testing, Verification, and Validation Workshops**. [S.l.], 2010. p. 110–119. Citado na página 46.

MATHUR, A. P. Performance, effectiveness, and reliability issues in software testing. In: IEEE. **1991 The Fifteenth Annual International Computer Software & Applications Conference**. [S.l.], 1991. p. 604–605. Citado na página 40.

MATHUR, A. P.; WONG, W. E. Evaluation of the cost of alternate mutation strategies. **VII Simpósio Brasileiro de Engenharia de Software**, p. 320–334, 1993. Citado nas páginas 40, 41 e 44.

MIRSHOKRAIE, S.; MESBAH, A.; PATTABIRAMAN, K. Efficient javascript mutation testing. In: IEEE. **2013 IEEE Sixth International Conference on Software Testing, Verification and Validation**. [S.l.], 2013. p. 74–83. Citado na página 48.

____. Guided mutation testing for javascript web applications. **IEEE Transactions on Software Engineering**, IEEE, v. 41, n. 5, p. 429–444, 2015. Citado na página 48.

MUBPEL - WS-BPEL Testing Tools - Redmine. 2017. <https://neptuno.uca.es/redmine/projects/sources-fm/wiki/MuBPEL/>. (Accessed on 02/28/2019). Citado na página 47.

MUTATEPY - | Pierre-Cyrille Héam. 2017. <http://members.femto-st.fr/pierre-cyrille-heam/mutatepy>. (Accessed on 02/28/2019). Citado na página 48.

MYERS, G.; SANDLER, C.; BADGETT, T. **The Art of Software Testing**. Wiley, 2011. (ITPro collection). ISBN 9781118133156. Disponível em: <https://books.google.com.br/books?id=GjyEFPkMCwcC>. Citado na página 29.

NAMIN, A. S.; ANDREWS, J. H.; MURDOCH, D. J. Sufficient mutation operators for measuring test effectiveness. In: ACM. **Proceedings of the 30th international conference on Software engineering**. [S.l.], 2008. p. 351–360. Citado na página 41.

NESTER - free software that helps to do effective unit testing in C#. 2013. <http://nester.sourceforge.net/>. (Accessed on 02/28/2019). Citado na página 48.

OFFUTT, A. J.; LEE, A.; ROTHERMEL, G.; UNTCH, R. H.; ZAPF, C. An experimental determination of sufficient mutant operators. **ACM Trans. Softw. Eng. Methodol.**, ACM, New York, NY, USA, v. 5, n. 2, p. 99–118, abr. 1996. ISSN 1049-331X. Disponível em: <http://doi.acm.org/10.1145/227607.227610>. Citado nas páginas 41 e 44.

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. (ICSE '93), p. 100–107. ISBN 0-89791-588-7. Disponível em: <http://dl.acm.org/citation.cfm?id=257572.257597>. Citado nas páginas 40, 41 e 44.

OFFUTT, A. J.; UNTCH, R. H. Mutation 2000: Uniting the orthogonal. In: ____. **Mutation Testing for the New Century**. Boston, MA: Springer US, 2001. p. 34–44. ISBN 978-1-4757-5939-6. Disponível em: <https://doi.org/10.1007/978-1-4757-5939-6_7>. Citado nas páginas 38 e 39.

OFFUTT, J.; MA, Y.-S.; KWON, Y.-R. An experimental mutation system for java. **ACM SIGSOFT Software Engineering Notes**, ACM, v. 29, n. 5, p. 1–4, 2004. Citado na página 48.

OMAR, E.; GHOSH, S.; WHITLEY, D. Homaj: A tool for higher order mutation testing in aspectj and java. In: IEEE. **2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops**. [S.l.], 2014. p. 165–170. Citado na página 47.

PAPADAKIS, M.; JIA, Y.; HARMAN, M.; Le Traon, Y. Trivial Compiler Equivalence: A Large Scale Empirical Study of a Simple, Fast and Effective Equivalent Mutant Detection Technique. In: IEEE PRESS. **2015 IEEE/ACM 37th IEEE International Conference on Software Engineering**. IEEE, 2015. p. 936–946. ISBN 978-1-4799-1934-5. Disponível em: <http://ieeexplore.ieee.org/document/7194639/>. Citado na página 51.

PAPADAKIS, M.; KINTIS, M.; ZHANG, J.; JIA, Y.; TRAON, Y. L.; HARMAN, M. Chapter six - mutation testing advances: An analysis and survey. In: MEMON, A. M. (Ed.). **Advances in Computers**. Elsevier, 2019, (Advances in Computers, v. 112). p. 275 – 378. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0065245818300305>. Citado nas páginas 31, 35, 38 e 46.

PARASOFT Insure++ Memory Debugging | Parasoft. 2019. <https://www.parasoft.com/products/insure>. (Accessed on 02/28/2019). Citado na página 47.

PARSAI, A.; MURGIA, A.; DEMEYER, S. Littledarwin: a feature-rich and extensible mutation testing framework for large and complex java systems. In: SPRINGER. **International Conference on Fundamentals of Software Engineering**. [S.l.], 2017. p. 148–163. Citado na página 47.

PIZZOLETO, A. V.; FERRARI, F. C.; OFFUTT, J.; FERNANDES, L.; RIBEIRO, M. A systematic literature review of techniques and metrics to reduce the cost of mutation testing. **Journal of Systems and Software**, v. 157, 2019. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85070519994&doi=10.1016%2Fj.jss.2019.07.100&partnerID=40&md5=16c5597d0bcf90620a695227f5273989>. Citado na página 54.

PLEXTEST | ITRegister. 2009. <http://www.itregister.com.au/products/plextest>. (Accessed on 02/28/2019). Citado na página 48.

POLO, M.; TENDERO, S.; PIATTINI, M. Integrating techniques and tools for testing automation. **Software Testing, Verification and Reliability**, Wiley Online Library, v. 17, n. 1, p. 3–39, 2007. Citado na página 49.

PRAPHAMONTRIPONG, U.; OFFUTT, J. Applying mutation testing to web applications. In: IEEE. **2010 Third International Conference on Software Testing, Verification, and Validation Workshops**. [S.l.], 2010. p. 132–141. Citado na página 49.

PRESSMAN, R. S. **Engenharia de Software: Uma Abordagem Profissional**. [S.l.]: AMGH Editora Ltda., 2016. ISBN 9788580555332. Citado na página 29.

RASCHKA, S. **Python machine learning**. [S.l.]: Packt Publishing Ltd, 2015. Citado nas páginas 55 e 71.

RICHARD, H. A.; DEMILLO, R. A.; HATHAWAY, B.; HSU, W.; HSU, W.; KRAUSER, E. W.; MARTIN, R. J.; MATHUR, A. P.; SPAFFORD, E. H. **Design Of Mutant Operators For The C Programming Language**. [S.l.], 1989. Citado nas páginas 64 e 114.

SCHULER, D.; DALLMEIER, V.; ZELLER, A. Efficient mutation testing by checking invariant violations. In: **Proceedings of the 18th International Symposium on Software Testing and Analysis, ISSTA 2009**. New York, NY, USA: Association for Computing Machinery, 2009. (ISSTA '09), p. 69–79. ISBN 9781605583389. Disponível em: <https://doi.org/10.1145/1572272.1572282>. Citado na página 52.

SCHULER, D.; ZELLER, A. Javalanche: efficient mutation testing for java. In: ACM. **Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering**. [S.l.], 2009. p. 297–298. Citado na página 47.

____. (Un-) covering equivalent mutants. In: IEEE. **2010 Third International Conference on Software Testing, Verification and Validation**. [S.l.], 2010. p. 45–54. Citado na página 52.

____. Covering and Uncovering Equivalent Mutants. In: **Software Testing Verification and Reliability**. [S.l.: s.n.], 2013. ISSN 09600833. Citado nas páginas 51, 52 e 95.

SHAHRIAR, H.; ZULKERNINE, M. Music: Mutation-based sql injection vulnerability checking. In: IEEE. **2008 The Eighth International Conference on Quality Software**. [S.l.], 2008. p. 77–86. Citado na página 48.

____. Mutation-based testing of format string bugs. In: IEEE. **2008 11th IEEE High Assurance Systems Engineering Symposium**. [S.l.], 2008. p. 229–238. Citado na página 48.

SHALEV-SHWARTZ, S.; BEN-DAVID, S. **Understanding Machine Learning: From Theory to Algorithms**. USA: Cambridge University Press, 2014. ISBN 1107057132. Citado nas páginas 58 e 59.

SMITH, B. H.; WILLIAMS, L. An empirical evaluation of the mujava mutation operators. In: IEEE. **Testing: Academic and Industrial Conference Practice and Research Techniques-MUTATION (TAICPART-MUTATION 2007)**. [S.l.], 2007. p. 193–202. Citado na página 47.

TANAKA, A. **Equivalence Testing for FORTRAN Mutation System Using Data Flow Analysis.** [S.l.], 1981. Citado na página 47.

TOKUMOTO, S.; YOSHIDA, H.; SAKAMOTO, K.; HONIDEN, S. Muvm: Higher order mutation analysis virtual machine for c. In: IEEE. **2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)**. [S.l.], 2016. p. 320–329. Citado na página 48.

TUYA, J.; SUAREZ-CABAL, M. J.; RIVA, C. D. L. Sqlmutation: A tool to generate mutants of sql database queries. In: IEEE. **Second Workshop on Mutation Analysis (Mutation 2006-ISSRE Workshops 2006)**. [S.l.], 2006. p. 1–1. Citado na página 49.

UNTCH, R. H. Mutation-based software testing using program schemata. In: **Proceedings of the 30th Annual Southeast Regional Conference**. New York, NY, USA: ACM, 1992. (ACM-SE 30), p. 285–291. ISBN 0-89791-506-2. Disponível em: <http://doi.acm.org/10.1145/503720.503749>. Citado nas páginas 40 e 49.

____. Schema-based mutation analysis: A new test data adequacy assessment method. 1997. Citado na página 49.

UNTCH, R. H.; OFFUTT, A. J.; HARROLD, M. J. Mutation analysis using mutant schemata. **SIGSOFT Softw. Eng. Notes**, ACM, New York, NY, USA, v. 18, n. 3, p. 139–148, jul. 1993. ISSN 0163-5948. Disponível em: <http://doi.acm.org/10.1145/174146.154265>. Citado nas páginas 39 e 40.

____. Mutation analysis using mutant schemata. In: ACM. **ACM SIGSOFT Software Engineering Notes**. [S.l.], 1993. v. 18, n. 3, p. 139–148. Citado na página 49.

USAOLA, M. P.; ROJAS, G.; RODRÍGUEZ, I.; HERNÁNDEZ, S. An architecture for the development of mutation operators. In: IEEE. **2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)**. [S.l.], 2017. p. 143–148. Citado na página 46.

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. 06, p. 675–689, 2002. Disponível em: <https://doi.org/10.1142/S021819400200113X>. Citado na página 52.

WALSH, T. A.; MCMINN, P.; KAPFHAMMER, G. M. Automatic detection of potential layout faults following changes to responsive web pages (n). In: IEEE. **2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)**. [S.l.], 2015. p. 709–714. Citado na página 48.

WOHLIN, C.; RUNESON, P.; HÖST, M.; OHLSSON, M. C.; REGNELL, B.; WESSLÉN, A. **Experimentation in software engineering**. [S.l.]: Springer Science & Business Media, 2012. Citado nas páginas 64 e 65.

WONG, W.; MATHUR, A. P. Reducing the cost of mutation testing: An empirical study. **Journal of Systems and Software**, v. 31, n. 3, p. 185 – 196, 1995. ISSN 0164-1212. Disponível em: <http://www.sciencedirect.com/science/article/pii/0164121294000980>. Citado nas páginas 31, 40, 41 e 44.

WONG, W. E.; MALDONADO, J. C.; DELAMARO, M. E.; SOUZA, S. D. R. S. D. A comparison of selective mutation testing in c and fortran. **Workshop do Projeto Validação e Teste de Sistemas de Operação**, p. 71–84, jan 1997. Citado nas páginas 17, 41, 42 e 44.

WONG, W. E.; MATHUR, A. P. Fault detection effectiveness of mutation and data flow testing. **Software Quality Journal**, v. 4, n. 1, p. 69–83, Mar 1995. ISSN 1573-1367. Disponível em: <https://doi.org/10.1007/BF00404650>. Citado na página 41.

ZHANG, L.; GLIGORIC, M.; MARINOV, D.; KHURSHID, S. Operator-based and random mutant selection: Better together. In: IEEE PRESS. **Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering**. [S.l.], 2013. p. 92–102. Citado nas páginas 17, 43 e 44.

ZHANG, L.; HOU, S.-S.; HU, J.-J.; XIE, T.; MEI, H. Is operator-based mutant selection superior to random mutant selection? In: ACM. **Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1**. [S.l.], 2010. p. 435–444. Citado na página 41.

ZHANG, L.; XIE, T.; ZHANG, L.; TILLMANN, N.; HALLEUX, J. D.; MEI, H. Test generation via dynamic symbolic execution for mutation testing. In: IEEE. **2010 IEEE International Conference on Software Maintenance**. [S.l.], 2010. p. 1–10. Citado na página 47.

ZHOU, C.; FRANKL, P. Mutation testing for java database applications. In: IEEE. **2009 International Conference on Software Testing Verification and Validation**. [S.l.], 2009. p. 396–405. Citado na página 47.

____. Jdama: Java database application mutation analyser. **Software Testing, Verification and Reliability**, Wiley Online Library, v. 21, n. 3, p. 241–263, 2011. Citado na página 47.

ZIP of files used to build the Csaw mutation tool. 2007. <http://www.skicambridge.com/papers/Csaw_v1_files.html>. (Accessed on 02/28/2019). Citado na página 47.

APÊNDICE

A

OPERADORES DE MUTAÇÃO

Quadro 4 – Operadores de mutação da linguagem Fortran-77

Classe	Sigla	Descrição
cca	aar	referência de *array* para substituição de referência de *array*
	acr	referência de *array* para substituição constante
	asr	referência de *array* para substituição de variável escalar
	car	constante para substituição de referência de *array*
	cnr	substituição de nome de *array* comparável
	csr	constante para substituição de variáveis escalares
	sar	variável escalar para substituição de referência de *array*
	scr	escalar para substituição constante
	src	substituição constante de origem
	svr	substituição de variável escalar
pda	abs	inserção de valor absoluto
	aor	substituição de operador aritmético
	crp	substituição constante
	dsa	alterações na instrução de dados
	icr	substituição do conector lógico
	ror	substituição de operador relacional
	uoi	inserção do operador unário
sal	der	substituição final do DO
	glr	substituição de marcação GOTO
	rsr	substituição de instrução RETURN
	san	análise de instrução (substituição por TRAP)
	sdl	instrução de deleção

Fonte: King e Offutt (1991).

Quadro 5 – Operadores de mutação da linguagem C

Classe	**Sigla**	**Descrição**
Mutação de Instrução	STRP	Laço em execução de instrução
	STRI	Laço em condição *if*
	SSDL	Instrução de deleção
	SRSR	Substituição de instrução de Retorno
	SGLR	Substituição de marcação goto
	SCRB	Substituição do *continue* por *break*
	SBRC	Substituição do *break* por *continue*
	SBRn	Saída do enésimo nível de enclausuramento
	SCRN	Continuação até o enésimo nível de enclausuramento
	SWDD	Substituição do *while* por *do-while*
	SDWD	Substituição do *do-while* por *while*
	SMTT	Multiplos saltos de laço
	SMTC	Múltiplos saltos de *continue*
	SSOM	Mutação de operador de sequência
	SMVB	Movimento de chave para cima ou para baixo
	SSWM	Mutação de instrução *Switch*
Mutação de Operadores	Obom	Mutação de operadores binários
	Ouor	Mutação de operadores unários
	Oido	Incremento/Decremento
	OLNG	Negação lógica
	OCNG	Negação de contexto lógico
	OBNG	Negação de operador *bitwise*
	OIPM	Mutação de precedência de operador de indicação
	OCOR	Substituição de operador de conversão
Mutação de variável	Vsrr	Substituição de referência de variável escalar
	Varr	Substituição de referência de *array*
	Vtrr	Substituição de referência de estrutura
	Vprr	Substituição de referência de ponteiro
	Vscr	Substituição de componente da estrutura
	Vasm	Mutação de índice de referência de *array*
	Vdtr	Laço de domínio
	Vtwd	Mutação de giro
Mutação de constante	Crcr	Substituição de constante requerida
	Cccr	Substituição de constante por constante
	Ccsr	Substituição de constante por escalar

Fonte: Richard *et al.* (1989).

Quadro 6 – Operadores de mutação da linguagem Java

Classe	Sigla	Descrição
Nível de método	AOD	Exclusão de operador aritmético
	AOI	Inserção de operador aritmético
	AOR	Substituição de operador aritmético
	ASR	Substituição de operador de atribuição
	COD	Exclusão de operador condicional
	COI	Inserção de operador condicional
	COR	Substituição de operador condicional
	LOD	Exclusão de operador lógico
	LOI	Inserção de operador lógico
	LOR	Substituição de operador lógico
	ROR	Substituição de operador relacional
	SOR	Substituição de Operador de Mudança
Nível de classe	EAM	Alteração no método de acesso
	EMM	Alteração no método de modificação
	EOA	Substituição de atribuição de referência e valor
	EOC	Substituição de atribuição de referência e valor
	IHD	Ocultar Exclusão de variável
	IHI	Ocultar inserção de variável
	IOD	Sobrescrita de método de deleção
	IOP	Sobrescrita de posição de chamada de método
	IOR	Sobrescrita de renomeio de método
	IPC	Chamada explícita da Exclusão de construtor pai
	ISD	Exclusão da palavra-chave *super*
	ISI	Inserção da palavra-chave *super*
	JDC	Criador padrão suportado por Java
	JID	Exclusão de inicialização de variável de membro
	JSD	Exclusão de modificador estático
	JSI	Inserção de modificador estático
	JTD	Exclusão da palavra-chave *this*
	JTI	Inserção da palavra-chave *this*
	OAC	Mudança de ordem de argumento
	OMD	Exclusão de sobrecarga de método
	OMR	Alteração no conteúdo do método de sobrecarga
	PCC	Alteração do tipo de conversão
	PCD	Inserção de operador de conversão de tipo
	PCI	Inserção de operador de conversão de tipo
	PMD	Declaração de variável de instância com tipo de classe pai
	PNC	Nova chamada de método com tipo de classe filho
	PPD	Declaração de variável de parâmetro com tipo de classe filho
	PRV	Atribuição de referência com outro tipo compatível

Fonte: Ma, Offutt e Kwon (2006).


ICMC
USP
SÃO CARLOS