MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÕES

PÁTRIA AMADA BRASIL GOVERNO FEDERAL

sid.inpe.br/mtc-m21d/2021/09.24.11.10-TDI

UMA INVESTIGAÇÃO SOBRE META E HIPER-HEURÍSTICAS PARA TESTE DE INTEGRAÇÃO DE SOFTWARE

Camila Pereira Sales

Dissertação de Mestrado do Curso de Pós-Graduação em Computação Aplicada, orientada pelo Dr. Valdivino Alexandre de Santiago Júnior, aprovada em 20 de setembro de 2021.

URL do documento original:
<http://urlib.net/ 8JMKD3MGP3W34T/45FQRFS>

INPE
São José dos Campos
2021

PUBLICADO POR:

Instituto Nacional de Pesquisas Espaciais - INPE
Coordenação de Ensino, Pesquisa e Extensão (COEPE)
Divisão de Biblioteca (DIBIB)
CEP 12.227-010
São José dos Campos - SP - Brasil
Tel.:(012) 3208-6923/7348
E-mail: pubtc@inpe.br

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÕES

PÁTRIA AMADA BRASIL GOVERNO FEDERAL

sid.inpe.br/mtc-m21d/2021/09.24.11.10-TDI

UMA INVESTIGAÇÃO SOBRE META E HIPER-HEURÍSTICAS PARA TESTE DE INTEGRAÇÃO DE SOFTWARE

Camila Pereira Sales

Dissertação de Mestrado do Curso de Pós-Graduação em Computação Aplicada, orientada pelo Dr. Valdivino Alexandre de Santiago Júnior, aprovada em 20 de setembro de 2021.

URL do documento original:
<http://urlib.net/ 8JMKD3MGP3W34T/45FQRFS>

INPE
São José dos Campos
2021

Dados Internacionais de Catalogação na Publicação (CIP)

Sales, Camila Pereira.

Sa32i Uma investigação sobre meta e hiper-heurísticas para teste de integração de software / Camila Pereira Sales. – São José dos Campos : INPE, 2021.

xx + 135 p. ; (sid.inpe.br/mtc-m21d/2021/09.24.11.10-TDI)

Dissertação (Mestrado em Computação Aplicada) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2021.

Orientador : Dr. Valdivino Alexandre de Santiago Júnior.

1. Teste de integração de software. 2. Meta-heurísticas. 3. Hiperheurísticas. 4. Aplicações em C++. 5. Automatização. I.Título.

CDU 004.415.53

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÕES

INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

DEFESA FINAL DE DISSERTAÇÃO DE CAMILA PEREIRA SALES BANCA Nº 263/2021, REG 438870/2019

No dia 20 de setembro de 2021, as 10h, por teleconferência, o(a) aluno(a) mencionado(a) acima defendeu seu trabalho final (apresentação oral seguida de arguição) perante uma Banca Examinadora, cujos membros estão listados abaixo. O(A) aluno(a) foi APROVADO(A) pela Banca Examinadora, por unanimidade, em cumprimento ao requisito exigido para obtenção do Título de Mestre em Computação Aplicada. O trabalho precisa da incorporação das correções sugeridas pela Banca Examinadora e revisão final pelo(s) orientador(es).

Novo título: “Uma investigação sobre meta e hiper-heurísticas para teste de integração de software”

Membros da banca:

Dr. Stephan Stephany - Presidente - INPE
Dr. Valdivino Alexandre de Santiago Júnior - Orientador - INPE
Dra. Karine Reis Ferreira Gomes - Membro Interno - INPE
Dra. Silvia Regina Vergilio - Membro Externo - UFPR

Documento assinado eletronicamente por **Stephan Stephany**, **Pesquisador Titular**, em 22/09/2021, às 10:40 (horário oficial de Brasília), com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Silvia regina vergilio (E)**, **Usuário Externo**, em 22/09/2021, às 16:21 (horário oficial de Brasília), com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Valdivino Alexandre de Santiago Júnior**, **Tecnologista em Ciência e Tecnologia**, em 22/09/2021, às 16:40 (horário oficial de Brasília), com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Karine Reis Ferreira Gomes**, **Tecnologista**, em 22/09/2021, às 17:37 (horário oficial de Brasília), com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

AGRADECIMENTOS

Agradeço primeiramente a Deus pela oportunidade e pelo privilégio que me foi dado em compartilhar tamanha experiência.

À minha família, sem eles nada disso seria possível. Obrigada pelo apoio, carinho e compreensão.

Aos professores da CAP, em especial ao meu orientador, Dr. Valdivino Alexandre de Santiago Júnior, por todo o conhecimento obtido.

Aos amigos e colegas que participaram desta etapa, agradeço por cada ensinamento.

À CAPES, pelo suporte financeiro sem o qual não seria possível a realização.

E a todos que direta e indiretamente participaram deste capítulo tão importante, o meu muito obrigada.

RESUMO

Teste de software é uma atividade que permite melhorar a qualidade de aplicações e, nesse contexto, existe um subcampo que é o Teste Baseado em Busca (TBB). A despeito de existir um grande interesse da comunidade acadêmica em TBB, existem poucos trabalhos que lidam com teste de integração de software via métodos de otimização e, mesmo assim, tais esforços não se relacionam à geração de casos de teste de integração. Além disso, é muito interessante que os métodos/metodologias possam gerar casos de teste com base apenas no código-fonte desde que, em muitos casos, não há uma documentação extensa sobre o produto. Essa dissertação de mestrado objetiva contribuir para a atividade de teste de integração de software via algoritmos de otimização, nesse caso meta-heurísticas e hiper-heurísticas. Duas hipóteses foram formuladas, sendo uma relacionada a viabilidade dos algoritmos de otimização para gerar os casos de teste de integração, e a outra em termos de quais algoritmos teriam um melhor desempenho sob a ótica da comunidade de otimização. Um método, denominado Software **In**tegration Testing via **Me**taheuristics and **Hy**per-heuristics (InMeHy), foi concebido para alcançar o objetivo de viabilidade, assim como ferramentas foram implementadas para apoiar o método. O InMeHy propõe gerar casos de teste baseando-se apenas no código-fonte desenvolvido em C++. Dois experimentos controlados, considerando como estudos de caso produtos relacionados a Sistemas de Informações Geográficas (SIGs), foram realizados para abordar a hipótese sobre desempenho, que no fundo trata de analisar a capacidade de generalização dos algoritmos. A hipótese sobre viabilidade foi aceita. No entanto, a segunda hipótese sobre desempenho foi rejeitada no contexto dos experimentos realizados, considerando os algoritmos e estudos de caso selecionados, onde pode-se afirmar que as meta-heurísticas clássicas (mais antigas) tiveram um melhor desempenho, não somente comparadas às hiper-heurísticas de seleção selecionadas mas, também, em relação às meta-heurísticas para inúmeros objetivos (*many-objective*) mais recentes. No entanto, é importante enfatizar que ainda não se pode generalizar as conclusões sobre a resposta ao problema da generalização dos algoritmos de otimização, pelo fato da hipótese sobre desempenho ter sido rejeitada. Em outras palavras, isso não pode ser entendido como um fato de que as meta-heurísticas multiobjetivo (clássicas) são realmente melhores do que as hiper-heurísticas de seleção e do que as meta-heurísticas para inúmeros objetivos, para qualquer tipo de problema. Além disso, essa conclusão, baseada em experimentações, parece estar relacionada aos teoremas sem almoço grátis (*no free lunch theorems*). Portanto, deseja-se apontar a necessidade de um número maior de experimentações rigorosas, para abordar a questão da generalização dos algoritmos de otimização na prática.

Palavras-chave: Teste de Integração de Software. Meta-heurísticas. Hiper-heurísticas. Aplicações em C++. Automatização. Experimentos Controlados.

INVESTIGATING META AND HYPER-HEURISTICS FOR SOFTWARE INTEGRATION TESTING

ABSTRACT

Software testing is an activity that allows improving the quality of applications and, in this context, there is a subarea which is Search-Based Software Testing (SBST). Despite the great interest of the academic community in SBTS, there are few studies dealing with software integration testing via optimization methods, and even so, such efforts are not related to the generation of integration test cases. Also, it is very interesting that the methods/methodologies can generate test cases based only on the source code since, in many cases, there is not extensive documentation about the product. This master's thesis aims to contribute to the software integration testing activity via optimization algorithms, in this case meta-heuristics and hyper-heuristics. Two hypotheses were formulated, one related to the feasibility of optimization algorithms to generate the integration test cases, and the other in terms of which algorithms would perform better from the perspective of the optimization community. A method, called Software **In**tegration Testing via **Me**ta-heuristics and **Hy**per-heuristics (InMeHy), was conceived to achieve the feasibility goal, as well as tools were implemented to support the method. InMeHy proposes to generate test cases based only on the source code developed in C++. Two controlled experiments, considering as case studies products related to Geographic Information Systems (GIS), were carried out to address the performance hypothesis, which ultimately deals with analyzing the generalizability of algorithms. The feasibility hypothesis was accepted. However, the second hypothesis about performance was rejected in the context of the experiments performed, considering the selected algorithms and case studies, where it can be said that the classical (older) meta-heuristics performed better not only compared to the selected selection hyper-heuristics but also in relation to meta-heuristics for a number of more recent (many-objective) goals. However, it is important to emphasize that the conclusions about the answer to the problem of generalization of optimization algorithms cannot be generalized yet, because the hypothesis about performance has been rejected. In other words, this cannot be understood as a fact that multi-objective (classical) meta-heuristics are actually better than selection hyper-heuristics and multipurpose metaheuristics for any type of problem. Furthermore, this conclusion, based on experiments, seems to be related to the no free lunch theorems. Therefore, we want to point out the need for a greater number of rigorous experiments to address the issue of generalization of optimization algorithms in practice.

Keywords: Software Integration Testing. Meta-heuristics. Hyper-heuristics.. C++ Applications. Automation. Controlled Experiments.

LISTA DE FIGURAS

Pág.

LISTA DE TABELAS

Pág.

LISTA DE ABREVIATURAS E SIGLAS

AE	–	Algoritmos Evolucionários
CGOBT	–	Coordenação-Geral de Observação da Terra
IBEA	–	*Indicator-Based Evolutionary Algorithm*
IGD	–	*Inverted Generational Distance*
IGD+	–	*Inverted Generational Distance Plus*
INPE	–	Instituto Nacional de Pesquisas Espaciais
HH	–	Hiper-heurística
HBN	–	Heurística de Baixo Nível
JMetal	–	*Metaheuristic Algorithms in Java*
LGR	–	Linguagem de Representação de Grafos
MOEA	–	*Multi-Objective Evolutionary Algorithms*
MOMBI-II	–	*Many Objective Meta-heuristic Based on the R2 Indicator*
NMOEA	–	*Many-Objective Evolutionary Algorithms*
NSGA-III	–	*Non-dominated Sorting Genetic Algorithm III*
SBST	–	*Search-Based Software Testing*
SIG	–	Sistemas de Informações Geográfica
SST	–	Software Sob Teste
STF	–	Solução como Suíte de Teste com Tamanho Fixo de Casos de Teste
STV	–	Solução como Suíte de Teste com Tamanho Variável de Casos de Teste
TBB	–	Teste Baseado em Busca
UML	–	*Unified Modeling Language*

SUMÁRIO

Pág.

1 INTRODUÇÃO

Metodologias, métodos e técnicas de Teste de software devem ser parte integrante do processo de desenvolvimento de software, uma vez que têm impacto direto na qualidade do produto entregue. Quando os defeitos são encontrados após o início da utilização de um sistema de software, isso pode acarretar em diversos tipos de prejuízos, principalmente quando levados em consideração sistemas críticos (LEVESON; TURNER, 1993; CIGNITI, 2021) como, por exemplo, software desenvolvidos para aplicações aeroespaciais (DALMAU; GIGOU, 1997; SANTIAGO JÚNIOR; VIJAYKUMAR, 2012; BRITISH BROADCASTING CORPORATION, 2019).

Outras aplicações de software, embora não sejam críticas, são de extrema importância, pois as mesmas podem ser utilizadas para apoiar o desenvolvimento de Sistemas de Informações Geográficas (SIGs) personalizados, como é o caso da biblioteca TerraLib (CÂMARA et al., 2008) desenvolvida pelo Instituto Nacional de Pesquisas Espaciais (INPE). Outras ferramentas servem para apoiar a análise de imagens de sensoriamento remoto, como é o caso da ferramenta GeoDMA (KÖRTING et al., 2013), também em desenvolvimento no INPE. Devido a isso, é de extrema relevância a existência de métodos/técnicas de geração de casos de Teste de software disponíveis que apresentem bons resultados para revelação de defeitos. Como é de conhecimento, a realização de Teste de maneira exaustiva não é uma alternativa viável em sistemas não triviais (SANTIAGO JÚNIOR et al., 2008) embora alguns trabalhos já tenham proposto abordagens quase exaustivas (PONZIO et al., 2016). Portanto, os métodos/técnicas de geração de casos de teste[1] servem para selecionar, de um conjunto infinito de possibilidades, os dados de entrada de teste que podem revelar a maior quantidade possível de defeitos no software.

Dentre as diversas abordagens propostas para gerar casos de teste, uma das que tem obtido destaque no momento é o Teste Baseado em Busca (TBB) (HARMAN et al., 2015; SAEED et al., 2016; BALERA; SANTIAGO JÚNIOR, 2019; KHARI; KUMAR, 2019). No fundo, busca e otimização é um subcampo da Inteligência Artificial (IA) e, desse modo, TBB também se relaciona à IA. TBB se baseia no fato de que os objetivos de Teste se assemelham à funções objetivo e, portanto, algoritmos de otimização podem ser usados para ajudar nesse sentido. Uma quantidade considerável de estudos têm sido publicados em TBB abordando vários tipos de Testes, tais como Teste funcio-

[1] Um **caso de teste** é usualmente formado por dados de entrada de teste e resultados esperados. No entanto, em alguns contextos como no caso de automatização da geração de casos de teste baseada em código-fonte, pode-se considerar os casos de teste somente como sendo os dados de entrada de teste. Essa é a perspectiva desse trabalho.

nal (WEGENER; BÜHLER, 2004), Teste de segurança (EVERSON; FIELDSEND, 2006), Qualidade de Serviço (QoS) (PENTA et al., 2007), Teste de Interação Combinatória (TIC) (GARVIN et al., 2011; PETKE et al., 2015), Teste de banco de dados (MCMINN et al., 2016), Teste Baseados em Modelos (ASOUDEH; LABICHE, 2014), entre outros. Em TBB, meta-heurísticas (DOKEROGLU et al., 2019) tais como Algoritmos Evolucionários (AEs) (Algoritmo Genético (MCCAFFREY, 2010; PETKE et al., 2015)), Otimização de Enxame de Partículas (MAHMOUD; AHMED, 2015; WU et al., 2015), *Harmony Search* (ALSEWARI; ZAMLI, 2012), *Simulated Annealing* (SA) (GARVIN et al., 2011) têm sido empregadas, dominando o subcampo de fato (BALERA; SANTIAGO JÚNIOR, 2019). A motivação para o uso desses algoritmos é o fato dos mesmos resolverem problemas de otimização não triviais, devido às suas capacidades em obter as melhores/mais adequadas soluções em relativamente pouco tempo, mesmo quando enfrentam problemas de tamanho muito grande.

Atualmente, existem ferramentas que apóiam a geração automática de casos de teste de unidade tais como Evosuite (FRASER; ARCURI, 2013) e Austin (LAKHOTIA et al., 2010). O Evosuite é uma ferramenta que utiliza um AE para gerar testes de unidada (em JUnit), baseando em código-fonte Java. Por sua vez, Austin realiza a geração de casos de teste de unidade, também fazendo uso de um AE, mas o código-fonte analisado deve ser escrito em linguagem C.

Apesar do sucesso das meta-heurísticas, pesquisadores afirmam que tais algoritmos ainda não são facilmente capazes de serem aplicados a novos problemas de otimização, com nenhuma ou mínima mudança, ou mesmo a novas instâncias do mesmo problema (BURKE et al., 2019). A capacidade de um algoritmo de otimização de resolver bem não apenas um problema específico, mas uma série de problemas distintos, é uma medida de sua **generalização**. Assim sendo, quanto maior for a capacidade de generalização de um algoritmo, melhor ele é. De acordo com essa perspectiva, as meta-heurísticas teriam baixa capacidade de generalização.

Portanto, houve a necessidade para o desenvolvimento de hiper-heurísticas, que são técnicas de busca de mais alto nível com o objetivo de obter maior capacidade de generalização (BURKE et al., 2013; BURKE et al., 2019; DRAKE et al., 2020). Nas hiper-heurísticas, a busca é realizada no espaço de heurísticas (ou componentes de heurística) ao invés de ser realizada diretamente no espaço das variáveis de decisão (espaço de soluções) (SANTIAGO JÚNIOR et al., 2020). Assim, a princípio, as hiper-heurísticas seriam mais gerais do que as meta-heurísticas. As hiper-heurísticas têm tido uma grande aceitação da comunidade de TBB, mas ainda precisam ser mais

exploradas (BALERA; SANTIAGO JÚNIOR, 2019).

1.1 Motivação

Para problemas de otimização diversos, vários estudos têm demonstrado um melhor desempenho das hiper-heurísticas em relação às meta-heurísticas (ALMEIDA et al., 2020; MAASHI et al., 2014; LI et al., 2019; SANTIAGO JÚNIOR et al., 2020). Em termos de TBB, a comunidade têm publicado trabalhos onde hiper-heurísticas têm sido usadas para geração de dados para TIC (JIA et al., 2015; ZAMLI et al., 2017), problema de ordenamento para Teste de integração (GUIZZO et al., 2015a; GUIZZO et al., 2015b; MARIANI et al., 2016; GUIZZO et al., 2017), derivação de produtos para teste de linhas de produto de software (FERREIRA et al., 2016; JAKUBOVSKI FILHO et al., 2017; FERREIRA et al., 2017), estratégias de geração de mutantes de segunda ordem (LIMA; VERGILIO, 2017), entre outros.

Mesmo que as hiper-heurísticas venham demonstrando, de uma maneira geral, um melhor desempenho do que as meta-heurísticas tanto para problemas diversos como no contexto de TBB, é interessante que investigações mais aprofundadas possam ser realizadas, para que se possa responder melhor a questão da generalização. Isso é particularmente válido se está se lidando com problemas não triviais de otimização discretos e reais, e não somente com *benchmarks* tais como DTLZ (DEB et al., 2005) e CEC 2009 (ZHANG et al., 2009).

No contexto de Teste de software, existem diferentes níveis de Teste, os quais estão relacionados à alguma fase de desenvolvimento do produto de software. Particularmente, o nível de Teste de integração ocorre quando as unidades (e.g. classes ou arquivos de acordo com o Paradigma de Orientação a Objetos) começam a ser integradas (SANTIAGO JÚNIOR, 2011). É um nível de Teste importante onde, usualmente, não se tem disponível todas as classes do sistema completo e a ênfase é dada em construir a estrutura da aplicação.

Muitos esforços relacionados à geração de casos de Teste de integração já foram publicados (VINCENZI et al., 2001; PINTE et al., 2008; OGATA; MATSUURA, 2010; SHIN et al., 2013). Assim como mencionado anteriormente, existem trabalhos que usam hiper-heurísticas para definir a ordem em que as unidades devem ser integradas, pois tal ordem pode impactar em diversos aspectos tais como o mascaramento de defeitos e aumentar o custo total de execução (MARIANI et al., 2016; GUIZZO et al., 2017). No entanto, até onde se tem conhecimento, existe uma escassez na literatura de trabalhos que usam métodos de otimização, tais como meta-heurísticas e hiper-

heurísticas, para gerar casos de Teste de integração.

A linguagem C++ ainda detém uma posição de destaque no âmbito das linguagens de programação, onde se mantém como uma das linguagens mais usadas de acordo com o índice Tiobe (TIOBE, 2021). O INPE desenvolveu e desenvolve diversos produtos de software utilizando a linguagem de programação C++. Alguns exemplos são as ferramentas TerraLib (CÂMARA et al., 2008), GeoDMA (KÖRTING et al., 2013) e TerraAmazon (TERRAAMAZON, 2020).

A tarefa de geração de casos de Teste de software de forma manual é complexa, e essa complexidade pode ocasionar na geração de testes que tenham uma baixa efetividade, uma vez que podem ser gerados casos de testes que cobrem pequenas partes do código-fonte, e que não sejam capazes de identificar os defeitos contidos no software. Nesse caso, os casos de teste precisam ser melhorados para verificar a integridade do software desenvolvido (YOSHIDA et al., 2017).

Tendo em vista essas observações apresentadas, é interessante investigar a viabilidade e eficácia de métodos de otimização, tais como meta-heurísticas e hiper-heurísticas, que possibilitem apoiar o Teste de integração de software para aplicações desenvolvidas na linguagem de programação C++. Essa investigação pode ser de grande valia não somente para o INPE mas para organizações que desenvolvem aplicações em C++. Além disso, se basear somente no código-fonte da aplicação C++ para gerar os casos de teste é algo valioso pois, usualmente, a documentação de um produto de software pode facilmente ficar defasada em relação ao código-fonte desenvolvido em metodologias de desenvolvimento de software mais tradicionais (e.g. modelo em V). Já em outras metodologias, tais como as metodologia ágeis (PERKUSICH et al., 2020), não existe muita ênfase em documentação dos seus produtos, se comparadas às metodologias tradicionais.

1.2 Objetivos e hipóteses

O objetivo geral do presente trabalho é contribuir para a atividade de Teste de integração de software via meta-heurísticas e hiper-heurísticas. Os objetivos específicos desse trabalho são:

- Viabilidade. Esse objetivo almeja verificar a viabilidade das meta-heurísticas e hiper-heurísticas para gerar os casos de Teste de integração;
- Desempenho. Esse objetivo almeja verificar quais dos dois tipos de algoritmos, meta-heurísticas e hiper-heurísticas, possuem o melhor desempenho

em termos de métricas propostas pela comunidade de otimização.

Para alcançar o objetivo de viabilidade, foi desenvolvido um método denominado Software **In**tegration Testing via **Me**taheuristics and **Hy**per-heuristics (InMeHy) (SALES; SANTIAGO JÚNIOR, 2020) que, baseando-se apenas no código-fonte de aplicações desenvolvidas em C++, realiza a geração de um grafo direcionado[2] que representa a integração de diversas classes/arquivos da aplicação. A partir desse grafo e baseando-se nos algoritmos de otimização, os casos de Teste de integração são gerados. Conforme apresentado no Capítulo 3, duas formas de geração de casos de teste foram desenvolvidas. Na **Solução como Suíte de Teste com Tamanho Variável de Casos de Teste** (STV), cada solução é uma suíte de teste[3] e a mesma pode ter um tamanho variável de casos de teste. Na **Solução como Suíte de Teste com Tamanho Fixo de Casos de Teste** (STF), cada solução é uma suíte de teste e a mesma tem um tamanho fixo e predefinido de casos de teste.

Em termos do objetivo específico relacionado a desempenho, dois experimentos controlados foram realizados. No primeiro experimento, foram consideradas a estratégia STV e:

- duas meta-heurísticas (AEs) multiobjetivo clássicas (*Multi-Objective Evolutionary Algorithms* (MOEAs)): *Indicator-Based Evolutionary Algorithm* (IBEA) (ZITZLER; KÜNZLI, 2004) e *Strength Pareto Evolutionary Algorithm-2* (SPEA2) (ZITZLER et al., 2001);
- duas meta-heurísticas (AEs) para inúmeros objetivos (*Many-Objective Evolutionary Algorithms* (MAEAs)) mais recentes: *Nondominated Sorting Genetic Algorithm-III* (NSGA-III) (DEB; JAIN, 2014) e *Metaheuristic Based on the R2 indicator-II* (MOMBI-II) (GÓMEZ; COELLO, 2015).

No segundo experimento, o foco é a estratégia STF e, além das meta-heurísticas do caso anterior, foram consideradas três hiper-heurísticas de seleção. Duas hiper-heurísticas são da família *Hyper-Heuristic based on Reinforcement LearnIng, Balanced Heuristic Selection and Group Decision AccEptance* (SANTIAGO JÚNIOR et al., 2020), onde uma é baseada na regra de responsabilidade (HRISE_R), e a outra

[2]Nesse trabalho, os termos "grafo direcionado" e "grafo" serão usados de forma intercambiável.

[3]Uma **sequência** difere de um **conjunto** porque a repetição de elementos é permitida e a ordem é importante. Uma **suíte de teste** é uma sequência de casos de teste.

considera a regra da maioria (HRISE_M). A outra hiper-heurística escolhida foi a *Choice Function* (HH-CF) (MAASHI et al., 2014).

Nos dois experimentos, a análise de desempenho foi realizada considerando os indicadores de qualidade, hipervolume (ZITZLER; THIELE, 1999), indicador ϵ (ZITZLER et al., 2003), e *Modified Inverted Generational Distance* (IGD+) (ISHIBUCHI et al., 2015), os quais são frequentemente adotados pela comunidade de otimização. Em ambos os experimentos, os estudos de caso foram dois produtos de software do INPE: TerraLib (CÂMARA et al., 2008) e GeoDMA (KÖRTING et al., 2013).

As hipóteses que se desejam validar nessa pesquisa podem ser formuladas da seguinte forma, onde cada hipótese se relaciona a um objetivo específico.

> **Hipótese 1:** É viável realizar a geração de casos de Teste de integração para aplicações desenvolvidas em C++ considerando algoritmos de otimização;
>
> **Hipótese 2:** As hiper-heurísticas possuem um melhor desempenho do que as meta-heurísticas.

1.3 Processo metodológico

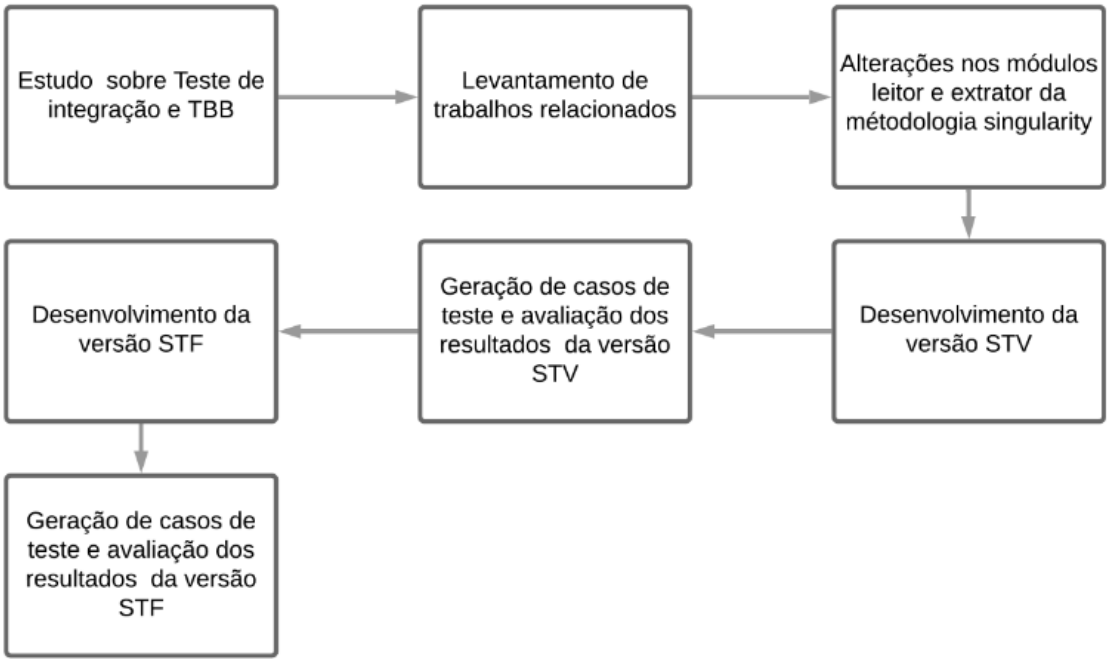
Visando alcançar o objetivo desta dissertação, foram definidas algumas atividades para guiar o processo de desenvolvimento do método, sendo estas categorizadas entre atividades teóricas e práticas. Esse processo pode ser visualizado na Figura 1.1, onde as atividades teóricas estão representadas pelas linhas contínuas e as práticas pelas linhas tracejadas. Primeiramente, foi realizada uma revisão da literatura para obter os trabalhos relacionados à Teste de integração e TBB, finalizando assim a parte teórica.

As atividades práticas se iniciaram alterando alguns módulos originários do método Singularity (ERAS et al., 2019) para atenderem ao escopo deste trabalho. Singularity é um método para geração de casos de Teste de unidade para aplicações desenvolvidas em C++.

Em seguida foi iniciado o processo de desenvolvimento e implementação em software da primeira estratégia do método InMeHy para geração de casos de teste: a versão STV (SALES; SANTIAGO JÚNIOR, 2020). Realizou-se, então, o primeiro experimento controlado com a geração de casos de Teste de integração e avaliação dos resultados. Verificou-se que a estratégia STV poderia gerar casos de teste inconsistentes (vide Capítulo 3) e, assim, teve início a um conjunto de modificações para gerar e

implementar em software a estratégia STF do método InMeHy. Assim como no caso anterior, um segundo experimento controlado foi realizado.

Figura 1.1 - Processo metodológico envolvendo as principais atividades desenvolvidas neste trabalho.



Fonte: Produção do autor.

1.4 Organização do texto

Os capítulos restantes desse manuscrito estão organizados da seguinte maneira:

- Capítulo 2: Esse capítulo apresenta conceitos relacionados aos temas dessa dissertação de mestrado, assim como trabalhos relacionados;
- Capítulo 3: Nesse capítulo é descrito, de forma detalhada, o método proposto, o InMeHy, sua estrutura e funcionamento, e os processos envolvidos;
- Capítulo 4: Nesse capítulo são descritas as avaliações experimentais realizadas;

- Capítulo 5: Nesse capítulo são apresentados os resultados das avaliações experimentais, assim como são realizadas discussões baseadas nesses resultados;
- Capítulo 6: Esse capítulo mostra como pode ser realizada a conversão dos casos de testes abstratos, gerados pela versão STF, em casos de teste executáveis, e a execução dos mesmos;
- Capítulo 7: Nesse capítulo, as considerações finais são apresentadas, assim como direções futuras.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Teste de software

Quando se almeja realizar o desenvolvimento de um produto de qualidade, é importante identificar inconsistências presentes no software que o impeçam de desempenhar o processo segundo descrito em suas especificações. Essas inconsistências podem surgir em todas as fases do ciclo de vida de um produto, mas quanto antes forem detectadas, menor será impacto que elas causarão no produto como um todo. Teste de software é uma das atividades de Verificação e Validação (V&V) (SANTIAGO JÚNIOR, 2011) que permitem criar um sistema com qualidade.

A atividade de Teste de software contém um extenso conjunto de definições de termos. Nesse trabalho, são adotadas as seguintes definições, segundo (DELAMARO et al., 2017):

- **defeito**: é um passo, processo ou definição de dados incorreto, por exemplo, uma instrução ou comando incorreto no programa;
- **engano**: é uma ação humana que produz um resultado incorreto, por exemplo, uma ação incorreta feita pelo programador;
- **erro**: ocorre quando o valor esperado e o valor obtido não são os mesmos, ou seja, qualquer resultado inesperado na execução do programa constitui um erro;
- **falha**: é a produção de uma saída incorreta em relação à especificação.

Além disso, a atividade de Teste de software é composta por um conjunto de tarefas, que combinadas buscam garantir a integridade do software, de forma que tanto a metodologia empregada para construção do software, como o produto em si, estejam condizentes com o que o descrito nas especificações do software (DELAMARO et al., 2017). Estas atividades não podem ser consideradas simples, uma vez que a sua realização pode abordar um montante, muitas vezes significativo, de variáveis que podem estar fora do controle do testador, como por exemplo, a natureza distinta dos defeitos introduzidos no código-fonte. Por esse motivo, se torna necessário que a atividade de Teste de software seja dividida em fases onde cada uma tenha um objetivo especifico e distinto aos outros (DELAMARO et al., 2017). A seguir, as fases gerais da atividade de Teste de software são mencionadas:

- **Teste de unidade**: tem por objetivo realiza a verificação de cada unidade que constitui o software, de maneira individual, para determinar se cada uma delas cumpre com o que foi especificado. Tem foco nas menores unidades de um programa como funções, procedimentos, métodos ou classes.

- **Teste de integração**: objetiva encontrar falhas de integração entre as unidades, normalmente executado após a finalização dos Testes de unidade. A ênfase é dada na construção da estrutura do sistema. Uma vez que as entidades estão trabalhando em conjunto, é preciso verificar as interações existentes, investigando se as partes funcionam de maneira adequada, não acarretando em novos defeitos;

- **Teste de sistema**: é um teste que é executado quando o sistema está todo desenvolvido. O objetivo é verificar se as funcionalidades especificadas nas especificações de requisitos estão todas corretamente implementadas. Requisitos não funcionais são explorados nessa fase também;

- **Teste de regressão**: o objetivo é verificar se a manutenção não inseriu novos defeitos.

2.1.1 Teste Baseado em Busca

Quando o problema de testar um software é formulado como um problema de otimização, tem-se o contexto de Teste Baseado em Busca (TBB) ou *Search-Based Software Testing* (SBST) (HARMAN et al., 2015). TBB é um subcampo da Engenharia de Software Baseada em Busca, onde os trabalhos publicados na área utilizam majoritariamente meta-heurísticas, sendo que ainda são necessários mais estudos utilizando hiper-heurísticas para apoiar o Teste de software.

De acordo com (MCMINN, 2011), o primeiro trabalho publicado nesta área foi o de (MILLER; SPOONER, 1976). Desde então, vários esforços nesta área foram empregados, de maneira que atualmente, podem ser encontrados trabalhos referentes a utilização de TBB aplicados em muitas técnicas de design de casos de teste, abrangendo desde Testes de caixa branca até Testes de caixa preta (AFZAL et al., 2009).

Entretanto existem problemas que ainda não foram bem explorados, como, o Teste de propriedades não funcionais e técnicas para encontrar estratégias de geração de dados de teste multiobjetivo, uma vez que os trabalhos com o objetivo de gerar casos de teste, normalmente utilizavam otimização mono-objetiva (HARMAN et al., 2015).

Os resultados obtidos pelos estudos efetuados nesta área revelam seu potencial, demonstrando como tornou possível a resolução de problemas de Engenharia de Software, e de Teste de software, que estavam sem solução satisfatória até então.

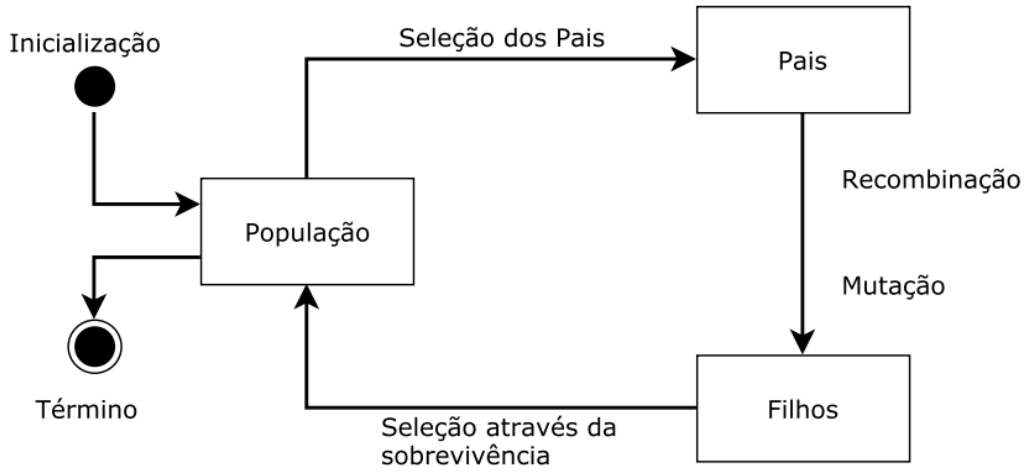
Antes de começar a discorrer sobre as meta-heurísticas e hiper-heurísticas usadas nessa dissertação, é importante definir os tipos de problema de otimização, de acordo com o que a comunidade mais aceita no momento (GÓMEZ; COELLO, 2015). Portanto, problemas com apenas uma função objetivo são denominados de **mono-objetivo**, problemas com dois ou três objetivos são denominados **multiobjetivo**, e problemas com quatro ou mais funções objetivo são considerados problemas com **inúmeros objetivos** (*many-objective*).

2.2 Meta-heurística

Meta-heurísticas podem ser definidas como um método heurístico de alto nível, as quais são propostas para a solução de uma ampla gama de problemas de otimização, ou seja, não específicas para um determinado problema (DOKEROGLU et al., 2019). As meta-heurísticas baseadas em AEs lidam com uma população, um conjunto de soluções, que evolui por meio de interações entre seus indivíduos. Durante estas interações procura-se manter as características desejáveis, pertinentes a cada população, com o intuito de, melhorar, ao decorrer das gerações, a qualidade média das soluções, sem comprometer a diversidade existente dentro da população (ASSUNÇÃO, 2012).

No contexto de Algoritmos Genéticos, cada indivíduo da população, ou seja, cada solução, é denominado como cromossomo. Para realizar a medição da aptidão desses indivíduos, é necessário fazer a definição das funções objetivo. Para realizar a alteração da população existem dois operadores principais responsáveis por esta ação: a mutação, que faz a modificação de uma das variáveis de decisão de um indivíduo e que, usualmente, tem uma baixa probabilidade relacionada, e recombinação (*crossover*), que faz a geração de novos indivíduos utilizando apenas a informação genética de dois outros indivíduos selecionados aleatoriamente. Indivíduos que apresentam baixa aptidão, de acordo com as funções objetivo, têm menor probabilidade de serem selecionados para recombinação. Este processo pode ser visualizado na Figura 2.1.

Figura 2.1 - Fluxo do processo de um algoritmo evolutivo.



Fonte: Lima (2017).

A seguir serão descritas brevemente as meta-heurísticas que serão utilizadas neste trabalho.

2.2.1 IBEA

Proposto em (ZITZLER; KÜNZLI, 2004), o IBEA é, como o próprio nome sugere, uma meta-heurística baseada em indicadores. O esquema de atribuição de aptidão desse algoritmo evolutivo é baseado em uma comparação pareada de soluções contidas em uma população usando um indicador de qualidade binário. O esquema de seleção para reprodução é um torneio binário entre indivíduos escolhidos aleatoriamente. Esse é um algoritmo inicialmente projetado para problemas multiobjetivo.

2.2.2 SPEA2

Outro algoritmo inicialmente concebido para problemas multiobjetivo é o SPEA2, proposto por (ZITZLER et al., 2001). Tal técnica é um aprimoramento de seu antecessor, SPEA, as principais melhorias que esta nova versão traz, em relação ao anterior, são: um esquema de atribuição de aptidão melhorado, que leva cada indivíduo em consideração, quantos indivíduos ele domina e por quais é dominado; a incorporação de uma técnica para realizar a estimativa de densidade do vizinho mais próximo,

que auxilia no processo de pesquisa; e a substituição do algoritmo de agrupamento por um de truncamento permite conservar as soluções limite.

2.2.3 NSGA-III

O NSGA-III proposto por (DEB; JAIN, 2014) é uma melhoria do NSGA-II, elaborado com o objetivo de proporcionar um melhor desempenho do framework ao lidar com problemas com um número muito alto de objetivos (*many-objective*). O NSGA-III é caracterizado pelo processo de atribuição de nicho chamado de classificação não-dominada baseada em pontos de referência. Esse é considerado um algoritmo adequado para problemas com inúmeros objetivos.

2.2.4 MOMBI-II

O algoritmo MOMBI-II é um aprimoramento do algoritmo MOMBI, onde é substituída a métrica ponderada de Tchebycheff, pela *Achievement Scalarizing Function*. Este algoritmo, assim como o IBEA, também faz uso de indicadores de qualidade para guiar a busca, mas nesse caso, o indicador escolhido é o R2 (GÓMEZ; COELLO, 2015). O indicador R2 é utilizado para classificar a população ao longo da execução do algoritmo. Essa população resulta da união da população ancestral com a descendente. Também é considerado um algoritmo para problemas com inúmeros objetivos.

2.3 Hiper-heurística

De acordo com Drake et al. (2020), hiper-heurística pode ser definida como "uma metodologia de pesquisa automatizada de alto nível que explora um espaço de pesquisa de heurísticas de baixo nível (...) ou componentes de heurísticas, para resolver problemas computacionalmente difíceis".

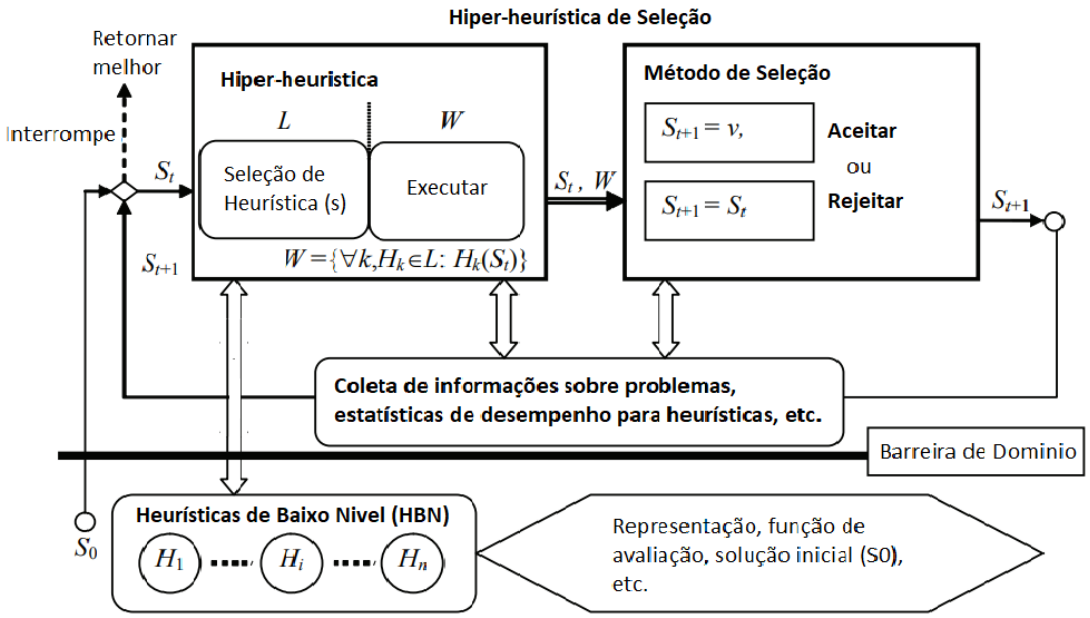
As hiper-heurísticas podem ser classificadas, de acordo com a natureza do espaço de busca, da seguinte forma (BURKE et al., 2013):

- hiper-heurísticas de Seleção: metodologias para escolher ou selecionar heurísticas existentes;
- hiper-heurísticas de Geração: metodologias para gerar novas heurísticas a partir de componentes de heurísticas existentes.

As hiper-heurísticas de seleção têm predominado sobre as de geração no contexto de

TBB. Uma explicação para esse fato é que as hiper-heurísticas de seleção têm menor complexidade para serem implementadas se comparadas às hiper-heurísticas de geração (BALERA; SANTIAGO JÚNIOR, 2019). O fluxo do processo de uma hiper-heurística de seleção pode ser observado na Figura 2.2, onde uma determinada solução candidata S_t, em um momento t, é modificada para uma nova solução, utilizando uma ou mais Heurísticas de Baixo Nível (HBNs) selecionadas. Perceber que uma solução S_t pode ser, de fato, uma população de soluções. Após isto é aplicado um método de aceitação com o intuito de aceitar ou rejeitar a solução resultante, este processo ocorre até que seja atingido o critério de parada (ÖZCAN et al., 2010).

Figura 2.2 - Estrutura de uma hiper-heurística de seleção.



Fonte: Adaptado de Özcan et al. (2010).

A característica que distingue as hiper-heurísticas, em relação às meta-heurísticas, é que elas, como métodos de alto nível, realizam a pesquisa no espaço formado por um conjunto de heurísticas de baixo nível que exploram o espaço das soluções. Uma das vantagens do uso de abordagens hiper-heurísticas, de acordo com a comunidade que as apoiam, está no fato de serem mais gerais (generalização) que as soluções meta-heurísticas e, assim, podem ser aplicadas a uma classe de problemas, e não a

um único problema específico.

Nas subseções seguintes, serão apresentadas as hiper-heurísticas que foram aplicadas neste trabalho.

2.3.1 Família HRISE

Em SANTIAGO JÚNIOR et al. (2020), os autores apresentaram duas hiper-heurísticas, denominadas HRISE_R e HRISE_M, as quais possuem um método de seleção de HBNs baseado em *roulette wheel* apoiado por Aprendizado por Reforço e seguido por um procedimento balanceado de *exploitation/exploration*. Além disso, existe uma estratégia de aceitação de populações em dois níveis: somente se as soluções melhorarem (*only improving*) seguida por uma tomada de decisões em grupo com diversos métodos de aceitação. A hiper-heurística HRISE_R usa a regra de responsabilidade, ao passo que a HRISE_M se baseia na regra da maioria.

2.3.2 Choice Function (HH-CF)

A hiper-heurística *Choice Function*, HH-CF, proposta por (MAASHI et al., 2014), objetiva resolver problemas de otimização multiobjetivo. Essa abordagem de alto nível controla e combina os pontos fortes de três MOEAs, que são utilizados como HBNs: NSGA-II (DEB; GOEL, 2001), SPEA2 (ZITZLER et al., 2001), e *Multiobjective Genetic Algorithm* (MOGA) (FONSECA; FLEMING, 2005). Nesse trabalho de mestrado, a HBN MOGA foi substituída pelo algoritmo IBEA. A estratégia de aceitação utilizada é todas aceitas (*all moves*), o que significa que o resultado de cada HBN é aceito, não importando se melhora ou não a solução.

2.4 Indicadores de qualidade

Os indicadores de qualidade servem para atestar quão boas são as populações geradas pelos algoritmos de otimização. Nesse trabalho, os seguintes indicadores foram usados:

- Hipervolume: esta é uma métrica que deve ser maximizada, ou seja, quanto maior o resultado maior a qualidade da solução. Este indicador computa a área, ou volume, de acordo com a quantidade de objetivos, do espaço de objetivos que uma fronteira de Pareto domina (ZITZLER; THIELE, 1999). Este indicador tem como maior desvantagem o seu elevado custo computacional, tendo em vista que cresce exponencialmente de acordo com a quantidade de objetivos (WHILE et al., 2005);

- ϵ: proposto em (ZITZLER et al., 2003) esta métrica deve ser minimizada, pois quanto menor for o valor de ϵ, significa que menos esforço precisou ser demandado para que uma solução ótima S dominasse uma solução P, portanto, quanto menor o resultado melhor a solução. Este indicador se mostra como uma ótima opção para problemas multi-objetivo, tendo em vista que apresenta baixo custo computacional em relação aos demais;

- IGD+: este indicador, assim como o indicador ϵ, deve ser minimizado, uma vez que este indicador mede a distancia entre a fronteira de Pareto. Proposto por (ISHIBUCHI et al., 2015) este indicador é a modificação do indicador IGD(*Inverted Generational Distance*), para que este tenha uma fraca conformidade com Pareto, o que é a sua principal vantagem em relação ao IGD, uma vez que este não é compatível com Pareto (ISHIBUCHI et al., 2019).

2.5 Trabalhos relacionados

Nesta subseção serão listados e descritos alguns trabalhos que, de alguma forma, se relacionam com o trabalho proposto, seja por terem como artefato de estudo o código-fonte, utilizarem algoritmos evolutivos, gerarem casos de testes, ou abordarem o nível de Teste de integração. Tais trabalhos podem ser separados de acordo com o tipo de Teste abordado. É importante destacar que alguns desses trabalhos se assemelham com o trabalho proposto em mais de um aspecto, mas todos demonstram algum processo parecido com o proposto.

O trabalho proposto por Assunção et al. (2011), realizou um estudo comparativo da aplicação de algoritmos evolutivos multiobjetivos, para estabelecer sequências de Teste de integração de classes. Foi utilizado um *parser* em conjunto com engenharia reversa aplicada no código-fonte escrito em Java. Nesse mesmo contexto do Teste de integração, o trabalho de (YANMEI et al., 2018) apresenta a aplicação do algoritmo de otimização de enxame de partículas ao problema de integração de classes e ordem de teste. Para os dados de entrada é realizada uma análise estática no código-fonte Java, obtém-se o diagrama de classes correspondente e, em seguida, a classe é mapeada para um espaço unidimensional.

Ambos os trabalhos citados acima não focam na atividade de geração de casos de teste, mas sim em determinar a sequência em que os casos de teste devem ser executados. Nesses casos específicos foram utilizados como dados de entrada o código-fonte, além de fazerem o uso de algoritmos de otimização.

Assim como os trabalhos acima, outros também fazem uso da linguagem Java, mas abordando a geração de casos de teste a nível de cobertura. No *Whole Test Suite* (WTS), a geração é feita por meio de uma estratégia em que, em vez de procurar um único caso de teste para cada objetivo de cobertura individual na sequência, altera o problema de pesquisa para uma busca por uma suíte de teste que cubra todos os objetivos de cobertura ao mesmo tempo (ROJAS et al., 2017). No entanto, os autores fazem uso de um único valor de função objetivo que agrega os valores de todas as funções objetivas, medidas para os casos de teste contidos em uma suíte de testes. Além disso, não se pode considerar diferentes tipos de metas de cobertura (e suas respectivas funções objetivo) ao mesmo tempo (por exemplo, cobertura de linhas e ramos).

Em (PANICHELLA et al., 2018), o *Dynamic Many-Objective Sorting Algorithm* (DynaMOSA) foi apresentado, especificamente para tratar do problema de geração de casos de teste no contexto de Teste de cobertura. É uma abordagem de inúmeros objetivos. Assim como o WTS, a abordagem não pode considerar diferentes tipos de metas de cobertura ao mesmo tempo. Tanto o WTS como o DynaMOSA, se baseiam em código-fonte em Java, ao passo que o método apresentado nessa pesquisa se baseia em código em C++. Assim como o DynaMOSA, o InMeHy é perfeitamente aplicável a problemas com inúmeros objetivos.

O estudo feito por (TONELLA, 2004) propõe a geração de Testes de unidade por meio de algoritmos de otimização com o suporte dado pela ferramenta *evolutionary Testing of classes* (eToc). A metodologia contempla também a conversão de dados de teste para o formato da ferramenta JUnit. Como dados de entrada a metodologia recebe o código-fonte na linguagem Java. Ainda no contexto de Teste de unidade, o estudo (DEVASENA; VALARMATHI, 2012) propõe uma ferramenta para geração de casos de testes de unidade por meio de um gerador de grafo de fluxo de controle, que é utilizado para gerar automaticamente casos de teste usando a técnica de busca de dispersão.

No contexto de Teste combinatorial, o trabalho de (LIN et al., 2015), é proposta uma estrutura meta-heurística de dois modos para a geração de arranjos com restrições. A partir dessa estrutura, é desenvolvido um algoritmo meta-heurístico chamado TCA. Nesse mesmo contexto, o estudo de Zamli et al. (2016) propôs uma estratégia híbrida de geração de Testes combinatoriais *t-way*, chamada *High Level Hyper-Heuristic* (HHH). O HHH adota o *Tabu Search* como sua heurística de alto nível e faz uso de quatro meta-heurísticas de baixo nível, incluindo otimização baseada no ensino-

aprendizagem, algoritmo global de vizinhança, otimização de enxame de partículas e algoritmo de pesquisa de cuco. Nestes trabalhos, foi feito o uso de meta e hiper-heurísticas para geração de casos de teste combinatoriais.

A técnica desenvolvida por (PINTE et al., 2008) apresenta um procedimento para a geração automática de dados de Teste de integração com base em algoritmos genéticos, utilizando como entrada diagramas de máquina de estados *Unified Modeling Language* (UML), permitindo a geração e otimização de suítes de testes de integração, que atendem a vários critérios de cobertura baseados no estado. (BRIAND et al., 2012) também faz uso do diagrama UML de máquina de estado, mas neste caso é utilizado juntamente com outro diagrama comportamental, o de sequência, estes dois diagramas são combinados para a geração de um grafo de fluxo de controle, que é utilizado para a geração dos casos de Teste de integração de acordo com dados conhecidos baseados em acoplamento. Uma limitação destas metodologias é a necessidade de que existam diagramas que descrevam o software, o que nem sempre ocorre.

Pode-se notar que a utilização de AEs para geração/seleção de casos de teste, independentemente do nível de Teste, é uma área de bastante interesse em TBB. Por exemplo, o problema de determinar a ordem em que as unidades devem ser integradas apresenta diversas propostas de utilização de AEs para solucioná-lo (COLANZI et al., 2011; SHARMA; SIBAL, 2014; BANSAL et al., 2010; GUIZZO et al., 2017; MARIANI et al., 2016).

A Tabela 2.1 sumariza os trabalhos relacionados apresentados e a proposta dessa dissertação sob quatro características:

- Uso de algoritmos de otimização (Otim);
- Geração de casos de teste (Gct);
- Aborda o nível de integração de teste (Int);
- Geração de casos de teste baseada em código-fonte (Fonte).

Nessa tabela, também foram incluídas as ferramentas Evosuite e Austin mencionadas no Capítulo 1. A partir dessa tabela, observa-se que a sua maioria não se baseia em código-fonte para gerar casos de teste. Além disso, nenhum dos trabalhos usou meta-heurísticas e/ou hiper-heurísticas para gerar os casos de teste para o nível de Teste de integração. O método proposto seria o único a cobrir as quatro características

apresentadas na Tabela 2.1. Desse modo, essa dissertação de mestrado aborda todos esses pontos para contribuir para a melhoria da qualidade de sistemas de software.

Tabela 2.1 - Características abordadas pelos estudos relacionados e do método proposto.

Artigo	Otim	Gct	Int	Fonte
(ASSUNÇÃO et al., 2011)	x		x	x
(YANMEI et al., 2018)	x		x	x
(TONELLA, 2004)	x	x		x
(DEVASENA; VALARMATHI, 2012)	x	x		x
(LIN et al., 2015)	x	x		
(ZAMLI et al., 2016)	x	x		
(PINTE et al., 2008)	x	x	x	
(BRIAND et al., 2012)		x	x	
(GUIZZO et al., 2017)	x		x	
(MARIANI et al., 2016)	x		x	
(ROJAS et al., 2017)	x	x		x
(PANICHELLA et al., 2018)	x	x		x
(FRASER; ARCURI, 2013)	x	x		x
(LAKHOTIA et al., 2010)	x	x		x
InMeHy	x	x	x	x

2.6 Considerações finais sobre este capítulo

Neste capítulo foi apresentada uma breve fundamentação teórica, onde foi abordado os principais conceitos contidos nesta dissertação. A partir da busca de trabalhos relacionados pode-se perceber que não existem trabalhos, até onde se sabe, que tenham abordado as quatro principais características deste trabalho. Deste modo no capítulo a seguir será detalhada a metodologia proposta, bem como a ferramenta que facilita a sua aplicação.

3 O MÉTODO InMeHy

Esse capítulo apresenta a descrição do método InMeHy e seu processo para realizar a geração de casos de teste de integração, a partir de código-fonte em C++ e por meio de meta e hiper-heurísticas. O método é composto por uma etapa responsável por fazer a transformação do código-fonte em C++ para um grafo e, também, de outra etapa responsável por realizar o procedimento para geração dos casos de testes abstratos. Também faz parte do método o procedimento para analisar o desempenho das meta e hiper-heurísticas. Porém, antes de entrar em detalhes sobre o método proposto, é necessário dar algumas definições que foram adotadas neste trabalho.

3.1 Definições

Primeiramente, existem duas estratégias para geração de casos de teste. Na STV, cada solução é uma suíte de teste e a mesma pode ter um tamanho variável de casos de teste. Na STF, cada solução é uma suíte de teste e a mesma tem um tamanho fixo e predefinido de casos de teste.

3.1.1 Definições para STV

As definições para a estratégia STV (SALES; SANTIAGO JÚNIOR, 2020) são dadas nessa subseção.

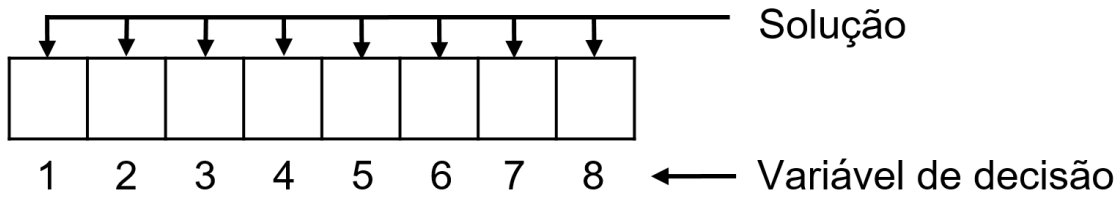
Definição 1. ***Caso de teste abstrato****: Um caso de teste abstrato é aquele cuja representação não permite que este seja efetivamente executado pelo Software Sob Teste (SST). Esse caso de teste abstrato serve como um guia para gerar o caso de teste verdadeiramente executável. Na estratégia STV, um caso de teste abstrato é uma sequência de vértices do grafo direcionado integrado, obtido pela leitura dos arquivos do SST.*

Como foi feita a utilização de meta-heurísticas e hiper-heurísticas, que são todas baseadas em AEs, tem-se a necessidade de realizar algumas associações, conforme mostrado a seguir.

Definição 2. ***Solução como suíte de teste com tamanho variável****: Uma solução é formada por uma sequência de variáveis de decisão. Uma solução de uma população criada por uma meta ou hiper-heurística é, de fato, uma suíte de teste, ou seja, uma sequência de casos de teste abstratos. O número de casos de teste abstratos que uma solução pode ter depende da quantidade de vértices terminais do grafo que representa o código-fonte, e que estão presentes na solução.*

Para um melhor entendimento, a Figura 3.1 mostra o relacionamento entre solução e variável de decisão.

Figura 3.1 - Solução e variável de decisão.



Fonte: Produção do autor.

Definição 3. ***Variável de decisão como passo de teste****: Uma variável de decisão é um elemento de uma solução. O valor de uma variável de decisão de uma solução é um número inteiro que identifica um vértice do grafo direcionado. Portanto, é considerada um passo de Teste de um caso de teste abstrato.*

3.1.2 Definições para STF

As definições para a estratégia STF (SANTIAGO JÚNIOR; SALES, 2021) são dadas nessa seção.

Definição 4. ***Caso de teste abstrato****: Um caso de teste abstrato é aquele cuja representação não permite que este seja efetivamente executado pelo SST. Esse caso de teste abstrato serve como um guia para gerar o caso de teste verdadeiramente executável. Na estratégia STF, um caso de teste abstrato é uma sequência de vértices relacionada a um passeio direcionado (directed walk*[4]*) no grafo direcionado.*

Importante perceber que, agora, um caso de teste abstrato é, de fato, uma sequência de vértices de um passeio direcionado enquanto na estratégia anterior não havia essa restrição em termos de passeio.

Definição 5. ***Solução como suíte de teste com tamanho fixo****: Uma solução é formada por uma sequência de variáveis de decisão. Uma solução de uma população*

[4]Uma sequência finita ou infinita de arestas direcionadas na mesma direção que une uma sequência de vértices é um **passeio direcionado**. Uma **trilha direcionada** é um passeio direcionado em que todas as arestas são diferentes. Cada trilha direcionada é um passeio direcionado, mas o oposto não é verdade.

criada por uma meta ou hiper-heurística é, de fato, uma suíte de teste, ou seja, uma sequência de casos de teste abstratos (sequência de sequências de vértices de passeios direcionados). O número de casos de teste abstratos contidos na solução é fixo na estratégia STF.

Definição 6. ***Variável de decisão como caso de teste abstrato****: Uma variável de decisão é um elemento de uma solução. O valor de uma variável de decisão de uma solução é um número inteiro que identifica uma sequência de vértices de um passeio direcionado no grafo, o qual representa a integração dos arquivos do SST. Portanto, uma variável de decisão representa uma sequência de vértices de um passeio direcionado e que é, também, um caso de teste abstrato.*

Esse método trata, portanto, da geração de casos de teste abstratos, conforme definido acima. Note que, também, esse é um problema de otimização discreta. Deste ponto em diante, salvo quando indicado o contrário, um **caso de teste abstrato** será denotado simplesmente como um **caso de teste** e um **passeio direcionado** será denotado apenas por **passeio**, por simplicidade.

Devido ao fato de existirem duas estratégias, STV e STF, o módulo Gerador (vide Seção 3.3.6) precisa ser diferente para cada uma dessas formas de gerar os casos de teste. Portanto, existem duas subseções no caso desse módulo. No entanto, os demais módulos não precisam sofrer qualquer alteração e são os mesmos, independentemente da estratégia para gerar casos de teste.

3.2 Exemplo de execução

Para apresentar o funcionamento de alguns dos módulos que compõem o método, foi criado um problema para ser utilizado como exemplo de execução: a aplicação `Conta`. Essa aplicação é formada por cinco arquivos distintos, sendo eles: “principal.cpp”, “conta.hpp”, “conta.cpp”, “usuário.hpp” e “usuário.cpp”. Os códigos-fonte contidos nestes arquivos são mostrados respectivamente nas Figuras 3.2, 3.3, 3.4, 3.5 e 3.6. A Tabela 3.1 apresenta os arquivos de inclusão presentes em cada arquivo e a Tabela 3.2 apresenta as instâncias de problema geradas a partir deste exemplo de execução.

Tabela 3.1 - Características do exemplo de execução `Conta`: arquivos e inclusões.

Aquivo	Inclusões
principal.cpp	conta.hpp e usuário.hpp
conta.hpp	usuário.hpp
conta.cpp	conta.hpp
usuário.hpp	sem inclusões
usuário.cpp	usuário.hpp

Figura 3.2 - Exemplo de execução: arquivo principal.cpp.

```
#include "conta.hpp"
#include "usuario.hpp"

int main()
{
  Usuario novoUsuario (22);
  Conta novaConta;

  bool validaUsuario = novoUsuario.verificaIdade();

  if (validaUsuario)
  {
      novaConta.inicializa(novoUsuario, 100);
      novaConta.deposita(10);
  }
  return 0;
}
```

Fonte: Produção do autor.

Figura 3.3 - Exemplo de execução: arquivo conta.hpp.

```
#ifndef __CONTA__
#define __CONTA__

#include <iostream>
#include "usuario.hpp"

using namespace std;

class Conta
{
  Usuario usuario_;
  float saldo_;
public:
  void inicializa(Usuario usuario, float s);
  void deposita(float valor);
};

#endif
```

Fonte: Produção do autor.

Figura 3.4 - Exemplo de execução: arquivo conta.cpp.

```
#include "conta.hpp"

void Conta::inicializa(Usuario usuario, float b){
  usuario_ = usuario;
  saldo_ = b;
  if (saldo_ < 0){
    cout << "error on create account!!!" << endl;
  }
}
void Conta::deposita(float value){
  saldo_ = saldo_ + value;
}
```

Fonte: Produção do autor.

Figura 3.5 - Exemplo de execução: arquivo usuário.hpp.

```
#ifndef __USUARIO__
#define __USUARIO__
#include <iostream>

class Usuario{
    public:
        //Construtors
        Usuario(int idade);

        // getMethods
        bool verificaIdade() const;

    private:
        int idade_;
};
#endif
```

Fonte: Produção do autor.

Figura 3.6 - Exemplo de execução: arquivo usuário.cpp.

```
#include "usuario.hpp"

Usuario::Usuario(int idade){
    idade_ = idade;
}

bool Usuario::verificaIdade() const{
    if (idade_ > 18){
        return true;
    }else{
        return false;
    }
}
```

Fonte: Produção do autor.

Tabela 3.2 - Características do exemplo de execução `Conta`: instâncias de problema e grafo.

Problema	Instância de Problema	#Vértices	#Arestas
`Conta`	ACC1_3	16	19
`Conta`	ACC2_3	23	29

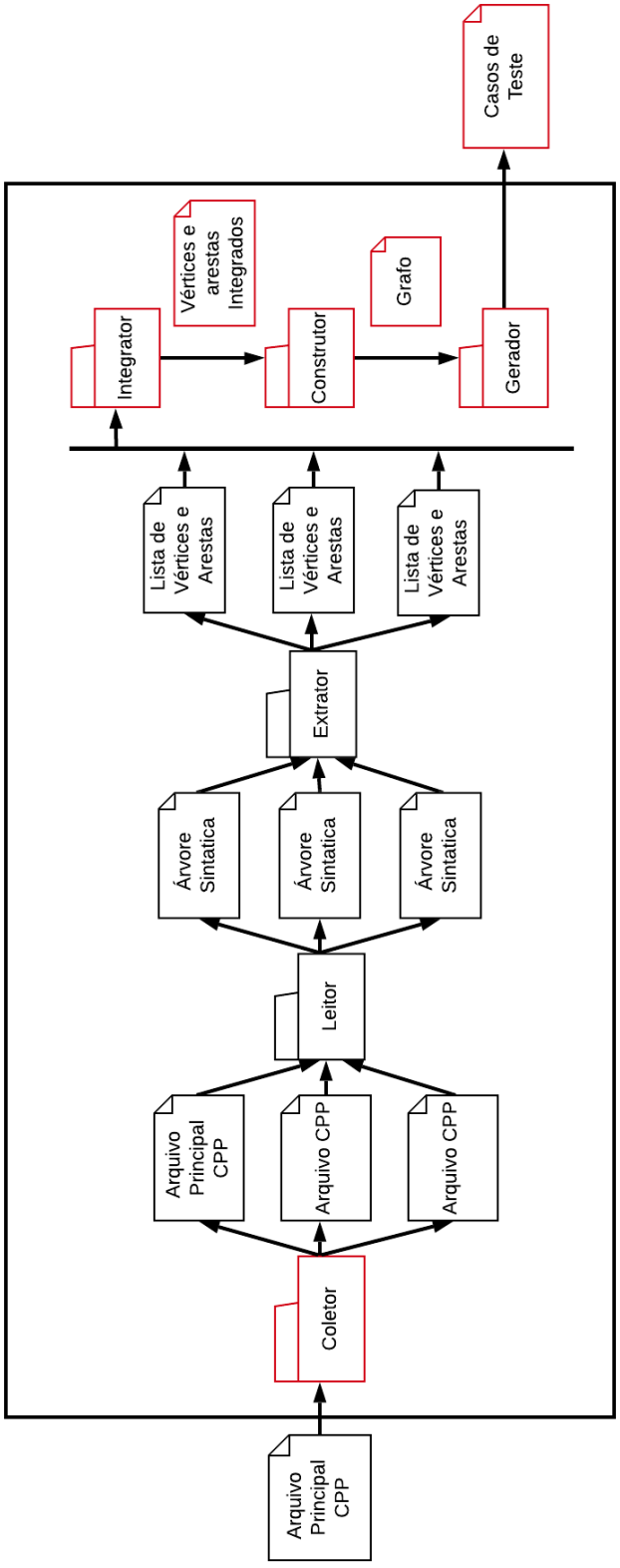
3.3 Estrutura

O método proposto, InMeHy, é composto por seis módulos: Coletor, Leitor, Extrator, Integrador, Construtor e Gerador. Os módulos Leitor e Extrator, bem como seus respectivos algoritmos, são originários do método *Singularity* proposto por (ERAS et al., 2019). A estrutura geral do método InMeHy é apresentada na Figura 3.7. Os módulos que são contribuições deste trabalho estão destacados em vermelho, e os módulos originários do método *Singularity* estão em preto. Percebe-se que o método tem como entrada um arquivo de código-fonte C++, que será utilizado para realizar a integração. O arquivo passado para inicializar o processo é tomado como arquivo base para a integração. A partir dele, os demais arquivos serão coletados recursivamente, e depois, integrados ao arquivo principal conforme apresentem relação.

O Algoritmo 1 descreve o algoritmo principal do método InMeHy. Note que esse algoritmo recebe como entrada os arquivos principais P, o caminho para a pasta raiz de cada programa, CP, a quantidade de arquivos f, e o nível de integração η. Nesse ponto é importante ressaltar uma observação. Ao gerar casos de teste para software de acordo com o paradigma de Programação Orientado a Objetos, uma unidade pode ser entendida como uma classe (PANICHELLA et al., 2018; ROJAS et al., 2017). Nesse sentido, o Teste de integração ocorre para expor defeitos nas interfaces e nas interações entre as classes integradas do sistema (PINTE et al., 2008). No entanto, se o sistema já está desenvolvido e em operação, e dessa forma está "completo", o Teste de integração pode ainda ser uma importante técnica para detectar defeitos de interface que ainda não foram encontrados.

Portanto, nesse contexto de Teste de integração, pode-se definir um arquivo como sendo a unidade. Nesse caso, o processo de integração começa a partir de um arquivo principal definido pelo usuário, e segue considerando todas as outras dependências (métodos, classes, etc.) para criar um modelo único. No contexto do método InMeHy, esse conceito de unidade é utilizado no módulo Coletor (Seção 3.3.1), o qual recebe um código-fonte C ++ (.cpp ou .hpp), que é considerado como o arquivo principal para iniciar o processo de integração.

Figura 3.7 - Arquitetura do método InMeHy.



Fonte: Produção do autor.

Algoritmo 1: PRINCIPAL

Entrada: P, CP, f, η
Saída: ST

```
1  início
2      para cada arquivo_principal ∈ P faça
3          lista_arquivos ← Coletor(arquivo_principal, CP, f)
4          para i = 1 até f/η faça
5              arquivos_integração ← lista_arquivos.próximosArquivos(f)
6              para cada arquivo ∈ arquivos_integração faça
7                  árvore ← Leitor(arquivo)
8                  lista_vértices, lista_arestas ←
                     lista_vértices, lista_arestas ∪ Extrator(árvore)
9              fim
10             vértices_integrados ← retornaPrincipal(lista_vértices)
11             arestas_integradas ← retornaPrincipal(lista_arestas)
12             lista_vértices.remove(vértices_integrados)
13             lista_arestas.remove(arestas_integradas)
14             vértices_integrados ←
                 integradorV(vértices_integrados, lista_vértices)
15             arestas_integradas ←
                 integradorE(vértices_integrados, arestas_integradas, lista_arestas)
16             grafo ←
                 Construtor.criarGrafo(vértices_integrados, arestas_integradas)
17             ST ← ST ∪ Gerador(lista_algoritmos, grafo)
18         fim
19     fim
20     retorna ST
21 fim
```

3.3.1 Módulo Coletor

O módulo Coletor é responsável por receber o arquivo principal (escrito em C++), e a partir dele realizar a busca dos demais arquivos que apresentem alguma ligação com ele, ou seja, quais arquivos estão incluídos com a diretiva *#include*. Esta busca é realizada de maneira recursiva em cada arquivo encontrado. Todos os arquivos relacionados são adicionados a uma lista, que será utilizada para fazer as chamadas

referentes ao módulo Leitor. Este processo é representado pelo Algoritmo 2.

Algoritmo 2: COLETOR

Entrada: $arquivo\_principal, f, CP$

Saída: $lista\_arquivos$

```
1  início
2      índice ← 0
3      lista_aquivos.adicionar(arquivo_principal)
4      repita
5          arquivos ← buscaArquivos(lista_arquivos.get(índex))
6          para cada arquivo_secundário ∈ arquivos faça
7              se arquivo_secundário ∈ CP então
8                  se arquivo_secundário ∉ lista_arquivos então
9                      lista_aquivos ← lista_aquivos ∪ arquivo_secundário
10                     índice ← índice + 1
11                     se arquivo.retornaExtensão() =
                        "hpp" ∨ arquivo.retornaExtensão() = "h" então
12                         adicionaArquivoCpp(arquivo, lista_aquivos)
13                     fim
14                 fim
15             fim
16         fim
17     até f < índice;
18     retorna lista_arquivos
19 fim
```

3.3.1.1 Exemplo de execução

De acordo com o Algoritmo 2, define-se nesse exemplo como parâmetros o arquivo "principal.cpp" (Figura 3.2), o caminho raiz da aplicação, CP, e o número máximo de arquivos $f = 6$. O primeiro passo é a adição do arquivo "principal.cpp" à lista de arquivos (*lista_principal*). Depois disto, é realizada a busca de arquivos a partir da *lista_principal*, onde o que for coletado será iterado, e para cada iteração, serão realizadas algumas verificações. Na linha 7, é verificado se o arquivo coletado existe no diretório do programa. Caso ele exista, na linha 8 é verificado se ele já não foi inserido. Em caso negativo, ele é inserido. Na linha 11, é realizada a verificação da extensão do arquivo: se for igual à .hpp ou .h é chamada a função *adicionaArquivoCpp*, passando como parâmetros o arquivo e a lista de arquivos.

Este método faz as mesmas verificações, mas desta vez, o arquivo a ser verificado é o de mesmo nome do arquivo .h ou .hpp, mas com extensão .cpp. Este processo é repetido para todos os arquivos contidos na *lista_principal* até que chegue ao número máximo de arquivos, ou que não tenha mais arquivos para realizar a busca.

3.3.2 Módulo Leitor

O módulo Leitor tem a função de ler o código-fonte presente no arquivo passado por parâmetro e, a partir dele, gerar uma árvore sintática contendo as informações inerentes ao código-fonte, segundo a gramática determinada. O Algoritmo 3 representa o processo realizado por este módulo, onde primeiramente é realizada a inicialização de um gerador de *parser*, passando como parâmetro a gramática da linguagem C++(VRIGAZOV, 2019). Esse gerador, em última instância, é responsável por gerar árvores sintáticas que serão usadas pelo módulo Extrator.

Algoritmo 3: LEITOR

Entrada: *arquivo*
Saída: *árvore_sintática*

```
1 início
2 |  geradorParser ← inicializaGeradorParser(GramáticaCPP)
3 |  árvore_sintática ← geradorParser(arquivo)
4 |  retorna árvore_sintática
5 fim
```

3.3.2.1 Exemplo de execução

As árvores sintáticas geradas por este módulo, para os arquivos conta.hpp e para conta.cpp, podem ser visualizadas nos Apêndices A.1 e A.2, respectivamente. Nestes apêndices, pode ser visualizada a maneira em que o código é traduzido para a árvore sintática.

3.3.3 Módulo Extrator

Este módulo é responsável por extrair os vértices e as arestas da árvore sintática gerada no módulo anterior. Durante o processo de extração, são coletadas algumas informações relevantes, como por exemplo o tipo de cada vértice, se é uma atribuição ou uma chamada de função, ou até mesmo o nível de escopo de cada vértice, tornando possível a partir disto, determinar os vértices e arestas inerentes à uma função específica em nível de integração. O processo é realizado em duas etapas,

onde primeiro são extraídos os vértices da árvore sintática e em seguida as arestas, como demonstrado no Algoritmo 4.

Algoritmo 4: EXTRATOR

Entrada: *árvore_sintática*
Saída: *vértices*, *arestas*

```
1 início
2 |  vértices ← extratorVértices(árvore_sintática)
3 |  arestas ← extratorArestas(árvore_sintática, vértices)
4 |  retorna vértices,arestas
5 fim
```

3.3.3.1 Exemplo de execução

A saída deste módulo é uma lista de vértices e uma de arestas. Nesta subseção, serão demonstradas as listas dos três arquivos que formam a instância de ACC1_-3 ("principal.cpp", "conta.hpp" e "conta.cpp"). A Figura 3.8 representa a junção das listas de vértices e arestas do arquivo principal. Pode-se observar a relação das instruções do código com o presente na figura. Por exemplo, a abertura da função *main* na Figura 3.2 pode ser visualizada na linha 1, assim como a instância de uma nova conta (linha 7 na Figura 3.2) pode ser vista na linha 3. Assim como o arquivo "principal.cpp", os arquivos "conta.hpp" e "conta.cpp" tem suas listas representadas mostradas nas Figuras 3.9 e 3.10, respectivamente.

Figura 3.8 - Representação da lista individual - arquivo main.cpp.

```
<vertice label="main" element="FUNCTION" class="main" arquivo="0" escopo="0" id="2">
    <vertice label="Usuario_novoU_22" element="STATEMENT" class="main" arquivo="0"
        escopo="1" id="3">
    <vertice label="Conta_novaConta" element="STATEMENT" class="main" arquivo="0" escopo
        ="1" id="4">
    <vertice label="bool_uValido_novoU_verificaIdade" element="ATTRIBUTION" class="main"
         arquivo="0" escopo="1" id="5">
    <vertice label="if_uValido" element="DECISION" class="main" arquivo="0" escopo="1"
        id="6">
        <vertice label="novaConta_inicializa_novoU_100" element="STATEMENT" class="main"
             arquivo="0" escopo="2" id="7">
        <vertice label="novaConta_deposita_10" element="STATEMENT" class="main" arquivo=
            "0" escopo="2" id="8">
    </vertice>
    <vertice label="return_0" element="JUMP" class="main" arquivo="0" escopo="1" id="10"
        >
</vertice>

<arestas from = "main_2" to = "Usuario_novoU_22_3" event = "lambda">
<arestas from = "Usuario_novoU_22_3" to = "Conta_novaConta_4" event = "lambda">
<arestas from = "Conta_novaConta_4" to = "bool_uValido_novoU_verificaIdade_5" event = "
    lambda">
<arestas from = "bool_uValido_novoU_verificaIdade_5" to = "if_uValido_6" event = "lambda
    ">
<arestas from = "if_uValido_6" to = "novaConta_inicializa_novoU_100_7" event = "TRUE">
<arestas from = "novaConta_inicializa_novoU_100_7" to = "novaConta_deposita_10_8" event
    = "lambda">
<arestas from = "novaConta_deposita_10_8" to = "return_0_10" event = "lambda">
<arestas from = "if_uValido_6" to = "return_0_10" event = "FALSE">
<arestas from = "return_0_10" to = "Usuario_novoU_22_3" event = "lambda">
<arestas from = "bool_uValido_novoU_verificaIdade_5" to = "
    novaConta_inicializa_novoU_100_7" event = "lambda">
<arestas from = "novaConta_deposita_10_8" to = "final_" event = "lambda">
```

Fonte: Produção do autor.

Figura 3.9 - Representação da lista individual - arquivo conta.hpp.

```
<vertice label="using_namespace" element="STATEMENT" class="conta" arquivo="1" escopo="0
    " id="14">
<vertice label="Conta" element="CLASS" class="conta" arquivo="1" escopo="0" id="15">
    <vertice label="Usuario_usuario" element="STATEMENT" class="conta" arquivo="1"
        escopo="1" id="16">
    <vertice label="float_saldo" element="STATEMENT" class="conta" arquivo="1" escopo="1
        " id="17">
    <vertice label="void_inicializa_Usuario_usuario_float_s" element="STATEMENT" class="
        conta" arquivo="1" escopo="1" id="18">
    <vertice label="void_deposita_float_valor" element="STATEMENT" class="conta" arquivo
        ="1" escopo="1" id="19">
</vertice>
<arestas from = "using_namespace_131" to = "Usuario_usuario_151" event = "lambda">
<arestas from = "Usuario_usuario_151" to = "float_saldo_161" event = "lambda">
<arestas from = "float_saldo_161" to = "void_inicializa_Usuario_usuario_float_s_171"
    event = "lambda">
<arestas from = "void_inicializa_Usuario_usuario_float_s_171" to = "
    void_deposita_float_valor_181" event = "lambda">
<arestas from = "void_deposita_float_valor_181" to = "final_" event = "lambda">
```

Fonte: Produção do autor.

Figura 3.10 - Representação da lista individual - arquivo conta.cpp.

```
<vertice label="inicializa" element="FUNCTION" class="conta" arquivo="2" escopo="0" id="
    21">
    <vertice label="usuario_usuario" element="ATTRIBUTION" class="conta" arquivo="2"
        escopo="1" id="22">
    <vertice label="saldo_b" element="ATTRIBUTION" class="conta" arquivo="2" escopo="1"
        id="23">
    <vertice label="if_saldo_0" element="DECISION" class="conta" arquivo="2" escopo="1"
        id="24">
        <vertice label="erro_na_cricao_da_conta" element="STATEMENT" class="conta"
            arquivo="2" escopo="2" id="25">
    </vertice>
</vertice>
<vertice label="deposita" element="FUNCTION" class="conta" arquivo="2" escopo="0" id="28
    ">
    <vertice label="saldo_saldo_value" element="ATTRIBUTION" class="conta" arquivo="2"
        escopo="1" id="29">
</vertice>

<arestas from = "inicializa_21" to = "usuario_usuario_22" event = "lambda">
<arestas from = "usuario_usuario_22" to = "saldo_b_24" event = "lambda">
<arestas from = "saldo_b_24" to = "if_saldo_0_25" event = "lambda">
<arestas from = "if_saldo_0_25" to = "erro_na_criacao_da_conta_25" event = "TRUE">
<arestas from = "erro_na_criacao_da_conta_25" to = "usuario_usuario_22" event = "
    erro_na_cricao_da_conta">
<arestas from = "if_saldo_0_25" to = "usuario_usuario_22" event = "FALSE">
<arestas from = "saldo_b_24" to = "erro_na_criacao_da_conta_25" event = "lambda">
<arestas from = "erro_na_criacao_da_conta_25" to = "saldo_saldo_valor_29" event = "
    erro_na_cricao_da_conta">
<arestas from = "deposita_281" to = "saldo_saldo_valor_29" event = "lambda">
<arestas from = "saldo_saldo_valor_29" to = "final_" event = "lambda">
```

Fonte: Produção do autor.

3.3.4 Módulo Integrador

Como o nome indica, o módulo Integrador integra as listas geradas anteriormente e retorna duas listas, uma contendo todos os vértices que interajam com a lista do arquivo principal ou com algum arquivo secundário que interaja com o principal, e outra contendo todas as arestas dos vértices presentes na lista de vértices. Estas duas listas representem as integrações das classes, bem como as arestas que entre elas.

O processo relacionado a este módulo pode ser dividido em duas etapas principais, cada uma abordando uma etapa para realizar a integração. A primeira etapa, representada no Algoritmo 5, é responsável por realizar a integração dos vértices com base nos vértices do arquivo principal. Esta integração é feita via correspondência, onde para cada vértice presente em uma lista, denominada *vértices_integrados*, é verificado se algum dos vértices dos arquivos secundários está relacionado a ele. Se essa relação existir, o vértice será adicionado à *vértices_integrados* e, em seguida, todos os vértices dependentes dele também serão adicionados.

Por fim, ocorre a integração das arestas com base na lista final de vértices gerada

na etapa anterior, este processo é realizado pelo Algoritmo 6. Neste processo, serão percorridas todas as listas de arestas e todas as arestas nelas presentes, e será feita uma verificação para encontrar as arestas que estão relacionadas com a lista final de vértices. Quando uma aresta é encontrada, esta aresta é adicionada à lista de arestas finais e, em seguida, as arestas dependentes dessa aresta são adicionadas também, se houver.

Algoritmo 5: IntegradorV

Entrada: *vértices_integrados*, *lista_vértices*
Saída: *vértices_integrados*

```
1  início
2      para cada lista_vértices ∈ vértices_integrados faça
3          para cada vértice ∈ lista_vértices faça
4              se existeRelação(vértice, vértices_integrados) então
5                  vértices_integrados ← vértices_integrados ∪ vértice
6                  addSubVértices(vértice, vértices_integrados)
7              fim
8          fim
9      fim
10     retorna vértices_integrados
11 fim
```

Algoritmo 6: IntegradorA

Entrada: *vértices_integrados*, *arestas_integradas*, *lista_arestas*
Saída: *arestas_integradas*

```
1  início
2      para cada arestas ∈ lista_arestas faça
3          para cada aresta ∈ arestas faça
4              se aresta.origem ∈ vértices_integrados ∨ aresta.destino
                ∈ vértices_integrados então
5                  arestas_integradas ← arestas_integradas ∪ aresta
6                  addSubAresta(aresta, arestas_integradas)
7              fim
8          fim
9      fim
10     retorna arestas_integradas
11 fim
```

3.3.4.1 Exemplo de execução

Nesta subseção, serão apresentadas a lista de vértices e a de arestas integradas. Para representar os dados, será utilizado o mesmo formato da seção anterior. A Figura 3.11 representa a integração dos três arquivos contidos na seção anterior. Pode-se perceber a presença de vértices dos arquivos "principal.cpp" (0) e "conta.cpp" (2), indicado pelo atributo *arquivo*. Além disso, também é possível observar a integração entre eles por meio das arestas como, por exemplo, a linha 38 que liga os vértices *_novaconta_inicializa_novou_100_7*, pertencente ao arquivo "principal.cpp", ao *_inicializa_21*, pertencente ao arquivo "conta.cpp".

Figura 3.11 - Representação da lista integrada - Instância de problema 1.

```
<vertice label="_main" element="FUNCTION" class="main" arquivo="0" escopo="0" id="2">
    <vertice label="_usuario_novou_22" element="STATEMENT" class="main" arquivo="0"
        escopo="1" id="3">
    <vertice label="_conta_novaconta" element="STATEMENT" class="main" arquivo="0"
        escopo="1" id="4">
    <vertice label="_bool_uvalido_novou_verificaidade" element="ATTRIBUTION" class="main
        " arquivo="0" escopo="1" id="5">
    <vertice label="_if_uvalido" element="DECISION" class="main" arquivo="0" escopo="1"
        id="6">
        <vertice label="_novaconta_inicializa_novou_100" element="STATEMENT" class="main
            " arquivo="0" escopo="2" id="7">
        <vertice label="_novaconta_deposita_10" element="STATEMENT" class="main" arquivo
            ="0" escopo="2" id="8">
    </vertice>
    <vertice label="_return_0" element="JUMP" class="main" arquivo="0" escopo="1" id="10
        ">
</vertice>
<vertice label="_inicializa" element="FUNCTION" class="conta" arquivo="2" escopo="0" id=
    "21">
    <vertice label="_usuario_usuario" element="ATTRIBUTION" class="conta" arquivo="2"
        escopo="1" id="22">
    <vertice label="_saldo_b" element="ATTRIBUTION" class="conta" arquivo="2" escopo="1"
         id="23">
    <vertice label="_if_saldo_0" element="DECISION" class="conta" arquivo="2" escopo="1"
         id="24">
        <vertice label="erro_na_cricao_da_conta" element="STATEMENT" class="conta"
            arquivo="2" escopo="2" id="25">
    </vertice>
</vertice>
<vertice label="_deposita" element="FUNCTION" class="conta" arquivo="2" escopo="0" id="
    28">
    <vertice label="saldo_saldo_value" element="ATTRIBUTION" class="conta" arquivo="2"
        escopo="1" id="29">
</vertice>

<arestas from = "_main_2" to = "_usuario_novou_22_3" event = "lambda">
<arestas from = "_usuario_novou_22_3" to = "_conta_novaconta_4" event = "lambda">
<arestas from = "_conta_novaconta_4" to = "_bool_uvalido_novou_verificaidade_5" event =
    "lambda">
<arestas from = "_bool_uvalido_novou_verificaidade_5" to = "_if_uvalido_6" event = "
    lambda">
<arestas from = "_if_uvalido_6" to = "_novaconta_inicializa_novou_100_7" event = "TRUE">
<arestas from = "_novaconta_inicializa_novou_100_7" to = "_novaconta_deposita_10_8"
    event = "lambda">
<arestas from = "_novaconta_deposita_10_8" to = "_return_0_10" event = "lambda">
<arestas from = "_if_uvalido_6" to = "_return_0_10" event = "FALSE">
<arestas from = "_bool_uvalido_novou_verificaidade_5" to = "
    _novaconta_inicializa_novou_100_7" event = "lambda">
<arestas from = "_novaconta_deposita_10_8" to = "final_" event = "lambda">
<arestas from = "_inicializa_21" to = "_usuario_usuario_22" event = "lambda">
<arestas from = "_usuario_usuario_22" to = "_saldo_s_23" event = "lambda">
<arestas from = "_saldo_s_23" to = "_if_24" event = "lambda">
<arestas from = "_if_24" to = "_erro_na_criacao_da_conta_25" event = "TRUE">
<arestas from = "_saldo_s_23" to = "_erro_na_criacao_da_conta_25" event = "lambda">
<arestas from = "_deposita_28" to = "saldo_saldo_valor_29" event = "lambda">
<arestas from = "_novaconta_inicializa_novou_100_7" to = "_inicializa_21" event = "">
<arestas from = "_erro_na_criacao_da_conta_25" to = "_novaconta_inicializa_novou_100_7"
    event = "">
<arestas from = "_novaconta_deposita_10_8" to = "_deposita_28" event = "">
<arestas from = "saldo_saldo_valor_29" to = "_novaconta_deposita_10_8" event = "">
```

Fonte: Produção do autor.

3.3.5 Módulo Construtor

Após a integração de todos os vértices e arestas, o módulo Construtor realiza o procedimento de criação de um arquivo em uma Linguagem de Representação de Grafos (LRG) tal como .dot. É feita uma varredura sequencial dos vértices e arestas. Primeiramente, são gerados um cabeçalho do arquivo e o vértice inicial do diagrama. Depois, a varredura dos vértices é feita sequencialmente.

Terminados os vértices, é adicionado um vértice final ao diagrama. Aqui, as arestas são iteradas uma a uma. Cada aresta é colocada no formato da LRG e inserida após a declaração dos vértices no arquivo. O Algoritmo 7 descreve o procedimento completo da geração do grafo (G) em formato de um arquivo LRG.

Algoritmo 7: CONSTRUTOR

Entrada: $vértices\_integrados, arestas\_integradas$
Saída: G

```
1 início
2 |  G ← gerarCabecalho(vértices_integrados)
3 |  G ← G ⋃ gerarVérticeInicial()
4 |  para cada transição ∈ arestas_integradas faça
5 |  |  G ← G ⋃ criarArestas(arestas_integradas)
6 |  fim
7 |  G ← G ⋃ gerarVérticeFinal()
8 |  retorna G
9 fim
```

3.3.5.1 Exemplo de execução

Retornando ao exemplo de execução, o grafo das instâncias de problema 1 e 2 (vide Tabela 3.2) são mostrados nas Figuras 3.12 e 3.13, respectivamente. Na instância de problema ACC1_3, tem-se a integração dos arquivos “principal.cpp”, “conta.hpp” e “ conta.cpp”, enquanto todos os cinco arquivos são integrados e criam a instância de problema ACC2_3. Percebe-se claramente como o código-fonte é transformado no grafo. Observe que **Conta novaConta;** na Figura 3.2 cria vértice o *Conta_novaConta_4* enquanto **saldo_ = s;** na Figura 3.4 deriva o vértice *saldo_s_23*.

3.3.6 Módulo Gerador

O módulo Gerador é responsável por receber o grafo direcionado gerado anteriormente, representando a integração dos arquivos do SST, e realizar a execução dos algoritmos escolhidos para gerar as suítes de teste. Conforme mencionado anteriormente, as estratégias STV e STF são formas diferentes para gerar os casos de teste. As Seções 3.3.6.1 e 3.3.6.3 detalham as estratégias STV e STF, respectivamente.

3.3.6.1 STV

Como descrito anteriormente, as Definições 1, 2 e 3 se relacionam à forma STV. É importante lembrar que aqui, uma variável de decisão representa um passo de um caso de teste (abstrato). O módulo Gerador recebe o grafo integrado, e o transforma em uma matriz de adjacência, que é uma das formas de representação de grafos. A matriz de adjacência de um grafo com |V| vértices é uma matriz |V|×|V| de 0s e 1s, na qual a entrada na linha i e coluna j é 1 se e somente se a aresta (i,j) estiver no grafo, e 0 caso contrário. Como aqui foi necessário representar o peso da aresta, foi colocado o valor 1 se houver arestas, mas que sejam instruções de menor custo computacional, tal como uma simples atribuição de valor a uma variável, e 2 se houver instruções de maior custo, tal como uma chamada de função. O valor 0 é colocado se não houver arestas ligando os dois vértices do grafo.

Para o STV, foram consideradas as seguintes funções objetivo:

- **Esforço de execução**: esta função visa avaliar o esforço de execução associado a uma solução (suíte de teste). Tal função leva em conta a matriz de adjacência que foi mencionada acima, e apenas soma-se os pesos das arestas relacionadas a tal conjunto de testes. Observe que esta função está relacionada a propriedades não-funcionais e deve ser minimizada;
- **Inconsistência**: não foram adicionadas restrições às instâncias de problema. A ideia foi dar flexibilidade total aos algoritmos para criar as soluções (suítes de teste). Mas, geralmente é necessário que a sequência de valores das variáveis de decisão, em um caso de teste (abstrato), seja consistente com a sequência de vértices (arestas) do grafo, caso contrário, um caso de teste não executável será gerado. O objetivo desta função é verificar esta inconsistência ao contar o número de arestas "inadequadas" em uma solução. A função é para ser minimizada;
- **Cobertura de arestas**: essa função é relacionada a teste funcional e mos-

tra a cobertura de arestas do grafo que representa uma instância de problema (conjunto de arquivos integrados). Essa função precisa ser maximizada e, portanto, se uma suíte de teste, St_1, tem uma cobertura de arestas maior do que outra, St_2, então St_1 é melhor do que St_2, nesse sentido.

Com base em tais funções objetivo e no grafo integrado, um algoritmo de otimização (meta-heurística, hiper-heurística) pode ser executado. A população, Pop_i, de soluções não dominadas, pop_{ij}, devido a execução de um algoritmo a_i é então um conjunto de suítes de teste (casos de teste abstratos) para depois serem traduzidos em casos de teste executáveis. Desde que cada $pop_{ij} \in Pop_i$ é uma solução não dominada, um profissional pode aleatoriamente selecionar uma ou mais suítes de teste (soluções) para executar, embora existam técnicas que permitem selecionar uma solução de um conjunto de soluções não dominadas.

O Algoritmo 8 mostra a forma genérica do módulo Gerador para a estratégia STV. O módulo recebe um conjunto de algoritmos (meta e hiper-heurísticas) para avaliar (Alg) e, para cada algoritmo a_i, as populações (Pop_i) são geradas de acordo com os princípios de cada técnica, por um número máximo de iterações ($máximo\_iterações$). E cada algoritmo é executado por um número máximo de execuções ($máximo\_execuções$).

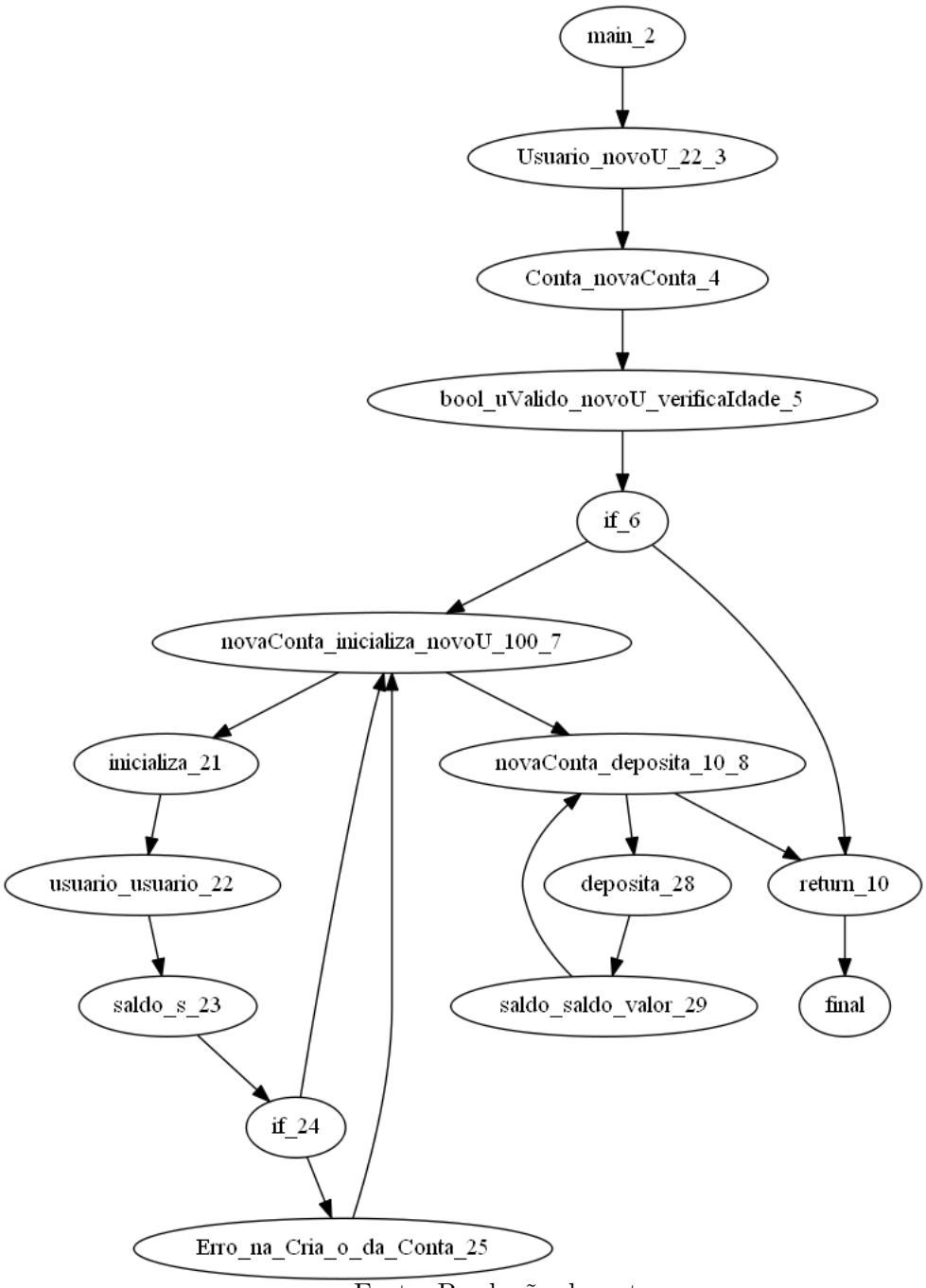
Uma vez que esses são problemas reais, e não problemas de *benchmark*, para calcular os indicadores de qualidade (I), cria-se a chamada **True Known Pareto Front** ($TKPF$) onde, para cada instância de problema, junta-se todas as populações finais (Pop_i) de todos os algoritmos, após todas as execuções, obtém-se as soluções não dominadas (MKAOUER; KESSENTINI, 2014) e remove-se as repetidas. Portanto, note que o módulo Gerador atua com uma instância de problema por vez. Além das populações ($\forall Pop_i$), que são conjuntos de suítes de teste, o módulo retorna os indicadores de qualidade (I).

Algoritmo 8: GERADOR - STV

Entrada: Alg, G
Saída: $\forall Pop_i, I$

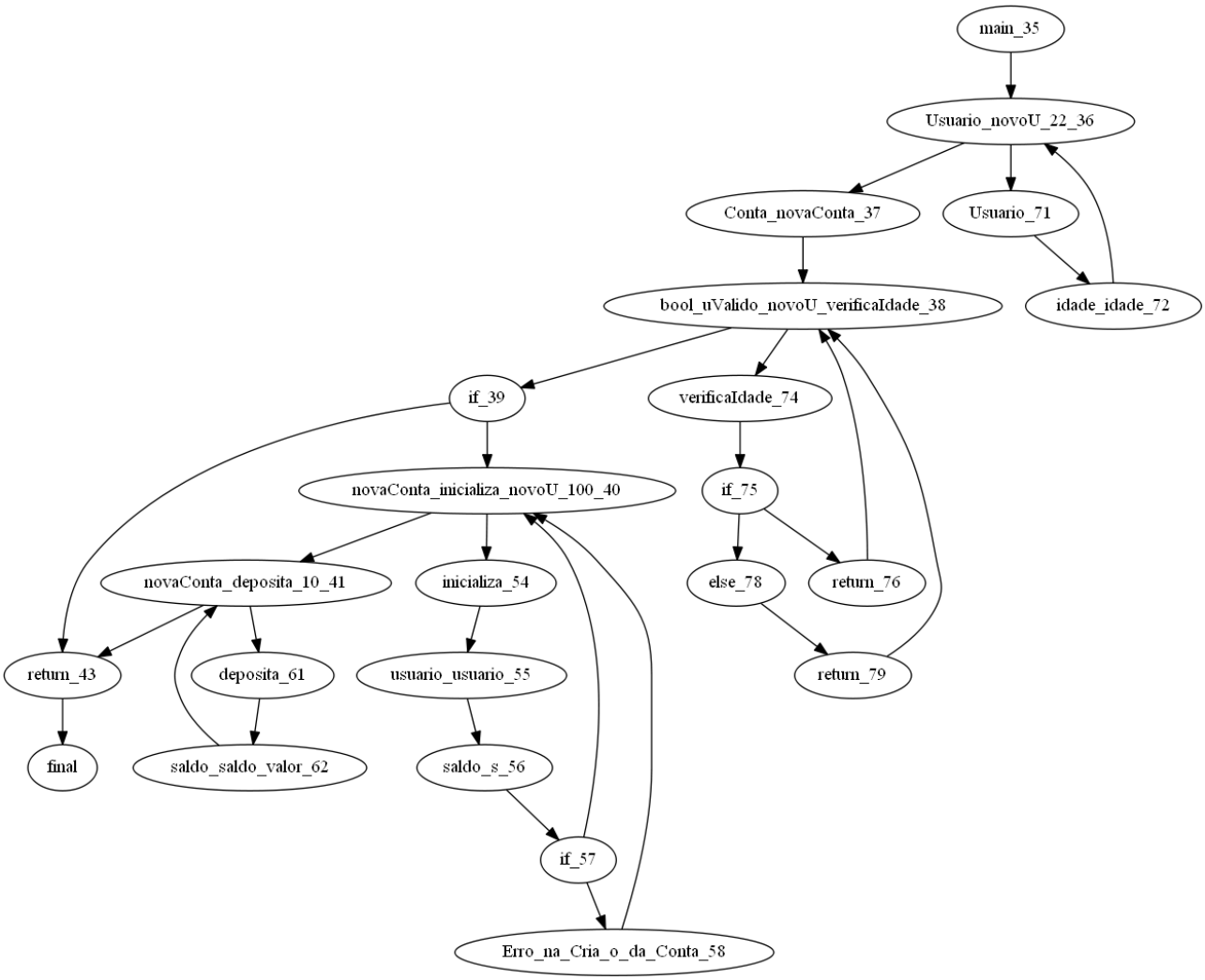
```
1  início
2      MA ← gerarMatrizAdjacência(G)
3      para cada a_i ∈ Alg faça
4          enquanto execuções ≤ máximo_execuções faça
5              Pop_i ← criarPopulaçãoInicial()
6              enquanto iterações ≤ máximo_interações faça
7                  Pop_i ← executarAlgoritmo(a_i, Pop_i, MA)
8              fim
9          fim
10     fim
11     TKPF ← criarTrueKnownParetoFront(∀Pop_i)
12     I ← calcularIndicadoresQualidade(TKPF, ∀Pop_i)
13     retorna ∀Pop_i, I
14 fim
```

Figura 3.12 - Grafo: instância ACC1_3.



Fonte: Produção do autor.

Figura 3.13 - Grafo: instância ACC2_3.



Fonte: Produção do autor.

3.3.6.2 Exemplo de execução - STV

A seguir, são exibidas partes das suítes de teste geradas via a estratégia STV. A Tabela 3.3 mostra suítes de teste geradas pelas quatro meta-heurísticas que foram consideradas nesse trabalho, para a instância de problema ACC1_3 (vide Tabela 3.2). A Tabela 3.4 se refere à instância de problema ACC2_3 (vide Tabela 3.2). Em ambos os casos, os vértices terminais, que determinam o final de um caso de teste e o início de um novo caso de teste, estão destacados em vermelho. Assim, pode-se observar a variação dos tamanhos dos casos de testes. Por exemplo, na Tabela 3.4 na linha referente ao algoritmo NSGA-III, pode-se observar que o vértice terminal 22 aparece 3 vezes, assim mostrando que tem pelo menos 3 casos de teste nesta suíte de teste, cada um com tamanho diferente. O primeiro caso de teste tem pelo menos 17 vértices, o segundo 5, e o terceiro apenas 2 vértices.

Tabela 3.3 - Parte das suítes de teste, instância ACC1_3: STV.

IBEA	... 10 8 6 2 9 9 3 3 3 1 12 10 0 7 15 13 12 1 4 13 13 11 7 15 ...
SPEA2	... 5 0 13 1 2 0 0 5 6 7 9 5 9 5 13 1 6 8 11 3 11 7 7 15 ...
NSGA-III	... 8 0 9 8 6 10 7 10 4 1 0 0 0 8 10 11 2 5 0 14 6 7 7 15 ...
MOMBI-II	... 3 10 8 9 0 0 14 6 2 13 8 13 0 12 0 8 3 0 8 4 0 0 7 15 ...

Tabela 3.4 - Parte das suítes de teste, instância ACC2_3: STV.

IBEA	... 1 15 17 5 8 15 4 15 13 16 1 3 11 19 12 14 18 8 4 6 6 6 12 22 ...
SPEA2	... 7 3 8 1 17 11 17 12 0 13 13 6 1 7 0 18 7 4 8 9 16 15 7 22 ...
NSGA-III	... 0 7 2 0 14 16 0 0 19 1 15 4 0 8 0 7 22 7 15 7 7 22 7 22 ...
MOMBI-II	... 16 6 8 3 8 21 1 13 17 17 18 14 17 16 0 2 2 1 5 17 18 0 7 22 ...

3.3.6.3 STF

A motivação para o desenvolvimento da estratégia STF foi perceber que, no caso de STV, diversos casos de teste eram incoerentes, a despeito de existir uma função objetivo para isso. Nesta seção, serão utilizadas as Definições 4, 5 e 6 fornecidas anteriormente. Lembrar que, agora, um caso de teste (abstrato) é uma sequência de vértices relacionada a um passeio direcionado, e é representado por uma variável de decisão.

Nessa estratégia, foram consideradas as funções objetivo de esforço de execução e

de cobertura de arestas, assim como no caso de STV, mas a função objetivo de inconsistência foi substituída pela de tamanho da suíte de teste:

- **Tamanho da suíte de teste**: essa é uma medida de custo que é simplesmente a soma do número de vértices de todos os casos de teste (variáveis de decisão) de uma suíte de teste (solução). Tal função deve ser minimizada, uma vez que, em geral, quanto menos eventos necessários para serem estimulados, melhor.

O Algoritmo 9 apresenta o módulo Gerador para a estratégia STF, o qual é muito similar ao da estratégia anterior. Porém, agora percebe-se que as entradas foram alteradas onde, além da lista de algoritmos(Alg) e do grafo integrado (G), é tomado como entrada, também, um parâmetro de restrição do número de trilhas direcionadas ($rest$). O objetivo de tal parâmetro é lidar com questões de escalabilidade que podem ocorrer com aplicativos não triviais.

Primeiramente, são obtidos os vértices inicial ($v\_inicial$) e final (v_final) do grafo integrado direcionado G. Os passeios direcionados (Pd) são obtidos em duas etapas. Primeiramente, é resolvido o problema do carteiro chinês, e com isso obtém-se um passeio fechado de comprimento mínimo, que visita todas as arestas do grafo pelo menos uma vez (EDMONDS; JOHNSON, 1973). Na segunda etapa, são criadas trilhas direcionadas (toda trilha direcionada também é um passeio direcionado), e então são unidas ao passeios direcionados obtidos pela resolução do problema do carteiro chinês.

A Figura 3.12 será utilizada para explicar a necessidade dessas duas etapas. Esta figura apresenta um grafo de integração obtido com base em arquivos C++.

Para resolver o problema do carteiro chinês, o grafo deve estar fortemente conectado. Caso não seja, como no exemplo da Figura 3.12, pode-se apenas adicionar uma aresta extra do vértice $final$ ao vértice $main\_2$ (o inicial). Cada caso de teste começa com o vértice inicial e termina no final. Portanto, uma solução típica do problema do carteiro chinês pode conter vários casos de teste, conforme mostrado no passeio fechado (pf) abaixo:

$$pf = \{final, main\_2, Usuario\_novoU\_22\_3, Conta\_novaConta\_4,$$
$$bool\_uValido\_novoU\_vericaIdade\_5, if\_6, return\_10, final, main\_2,$$
$$Usuario\_novoU\_22\_3, Conta\_novaConta\_4, \cdots, return\_10, final\}.$$

Assim, se faz necessária a divisão do passeio fechado e a geração de passeios direcionados, e suas sequências de vértices, que começam com $main\_2$ e terminam em $final$. Neste exemplo, os dois casos de teste (ct_1, ct_2) criados por meio da solução do problema do carteiro chinês são os seguintes:

$$ct_1 = \{main\_2, Usuario\_novoU\_22\_3, Conta\_novaConta\_4,$$
$$bool\_uValido\_novoU\_vericaIdade\_5, if\_6, return\_10, final\}.$$

$$ct_2 = \{main\_2, Usuario\_novoU\_22\_3, Conta\_novaConta\_4,$$
$$bool\_uValido\_novoU\_vericaIdade\_5, if\_6,$$
$$novaConta\_inicializa\_novoU\_100\_7,$$
$$initializa\_21, usuario\_usuario\_22, saldo\_s\_23, if\_24,$$
$$novaConta\_inicializa\_novoU\_100\_7,$$
$$initializa\_21, usuario\_usuario\_22, saldo\_s\_23, if\_24,$$
$$\cdots, novaConta\_inicializa\_novoU\_100\_7, return\_10, final\}.$$

Observe que pode-se obter vértices e arestas relacionadas nos casos de teste acima, conforme visto em ct_2. A motivação para criar casos de teste por meio da solução do problema do carteiro chinês é fornecer aos algoritmos de otimização um conjunto de casos de teste que, juntos, cubram todas as arestas do grafo pelo menos uma vez. Logo, essas são opções que as variáveis de decisão de uma solução podem contemplar. Mas observe que alguns casos de teste podem se tornar enormes (grande número de vértices) se forem considerados grafos grandes. Por exemplo, um caso de teste como ct_2 considerando um grafo contendo milhares de vértices e arestas pode ser muito grande, tornando-se inviável para ser executado na prática. Além disso, normalmente, pode-se ter poucos casos de teste derivados do problema do carteiro chinês, onde alguns são muito grandes, resultando em menos opções possíveis para as variáveis de decisão.

Este é o motivo de criar casos de teste adicionais obtendo apenas as trilhas direcionadas ($criarTrilhasDirecionadas$), para ter casos de teste menores e aumentar o número deles. Como esses casos extras são trilhas direcionadas, eles não contêm

arestas repetidas, mas ainda podem ter vértices repetidos. Como mencionado acima, o parâmetro *rest* serve para limitar o número máximo de trilhas direcionadas para lidar com problemas de escalabilidade.

Algoritmo 9: GERADOR - STF

Entrada: $Alg, G, rest$
Saída: $\forall Pop_i, I$

```
1  início
2      v_inicial ← obterVeticeInicial(G)
3      v_final ← obterVeticeFinal(G)
4      Pd ← obterCarteiroChinês(G)
5      Pd ← Pd ⋃ criarTrilhasDirecionadas(G, v_inicial, v_final, rest)
6      para cada a_i ∈ Alg faça
7          enquanto execuções ≤ máximo_execuções faça
8              Pop_i ← criarPopulaçãoInicial()
9              enquanto iterações ≤ máximo_interações faça
10                 Pop_i ← executarAlgoritmo(a_i, Pop_i, Pd)
11             fim
12         fim
13     fim
14     TKPF ← criarTrueKnownParetoFront(∀Pop_i)
15     I ← calcularIndicadoresQualidade(TKPF, ∀Pop_i)
16     retorna ∀Pop_i, I
17 fim
```

O procedimento para executar os casos de teste é similar ao apresentado na seção anterior. Em outras palavras, com a população, Pop_i, de soluções não dominadas, pode-se escolher uma ou mais soluções (suítes de teste) para depois serem traduzidas em casos de teste executáveis.

3.3.6.4 Exemplo de execução - STF

Retornando ao exemplo de execução, a seguir são exibidas partes das suítes de teste geradas via a estratégia STF. A Tabela 3.5 mostra suítes de teste geradas pelas quatro meta-heurísticas e três hiper-heurísticas que foram consideradas nesse trabalho, para a instância de problema ACC1_3 (vide Tabela 3.2). A Tabela 3.6 se refere à instância de problema ACC2_3 (vide Tabela 3.2). Pode-se observar a grande repetição dos casos de teste, isso ocorre devido ao fato de não ter sido adicionada nenhuma restrição a esse respeito e também pelo motivo das instancias de problemas serem simples e não terem gerados muitos passeis possíveis. Cada inteiro que aparece

nas tabelas é um valor de variável de decisão e que, portanto, corresponde a um caso de teste abstrato. Portanto, para a instância de problema ACC1_3, o número 4 na Tabela 3.5 representa o caso de teste a seguir:

$$
\begin{aligned}
ct_4 = \{&main\_2, Usuario\_novou\_22\_3, Conta\_novaConta\_4,\\
&bool\_uValido\_novoU\_verificaIdade\_5, if\_6,\\
&novaConta\_inicializa\_novoU\_100\_7, inicializa\_21,\\
&usuario\_usuario\_22, saldo\_s\_23, if\_24,\\
&erro\_na\_cria\_o\_da\_conta\_25, novaConta\_inicializa\_novou\_100\_7,\\
&novaConta\_deposita\_10\_8, return\_10, final\}.
\end{aligned}
$$

De forma similar, para a instância de problema ACC2_3, o número 7 na Tabela 3.6 representa o caso de teste a seguir:

$$
\begin{aligned}
ct_7 = \{&main\_35, Usuario\_novoU\_22\_36, Conta\_novaConta\_37,\\
&bool\_uvalido\_novoU\_verificaIdade\_38, if\_39, return\_43, final\}.
\end{aligned}
$$

Esses dois casos de teste para as instâncias de problema ACC1_3 e ACC2_3 estão destacados nas Figuras 3.14 e 3.15, respectivamente, onde cada um dos vértices em amarelo são os contemplados pelo caso de teste. Pode-se observar que o caso de teste para a instância ACC1_3 (Figura 3.14) realiza uma melhor cobertura de arestas no grafo em relação ao caso de teste para a instância ACC2_3 (Figura 3.15).

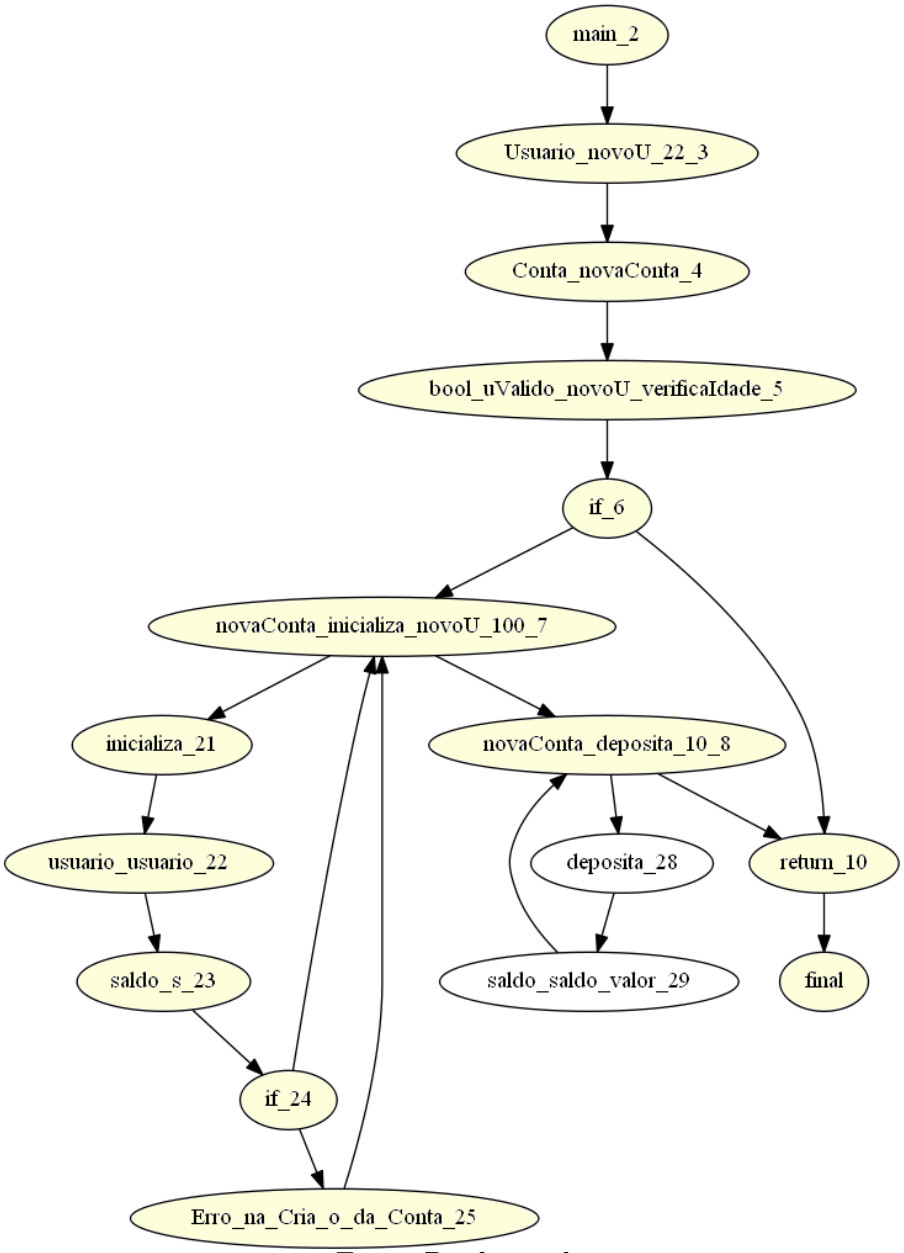
Tabela 3.5 - Parte das suítes de teste, instância ACC1_3: STF.

HH-CF	4	7	7	1	7	1	1	1	1	1
HHISE_M	7	7	1	1	1	1	0	7	7	1
HHISE_R	1	7	7	6	6	6	6	1	0	6
IBEA	1	1	4	1	1	1	1	1	1	1
MOMBI-II	1	1	1	1	7	1	7	4	7	1
NSGA-III	1	1	1	0	1	1	1	1	1	1
SPEA-II	1	1	1	1	1	1	1	1	1	4

Tabela 3.6 - Parte das suítes de teste, instância ACC2_3: STF.

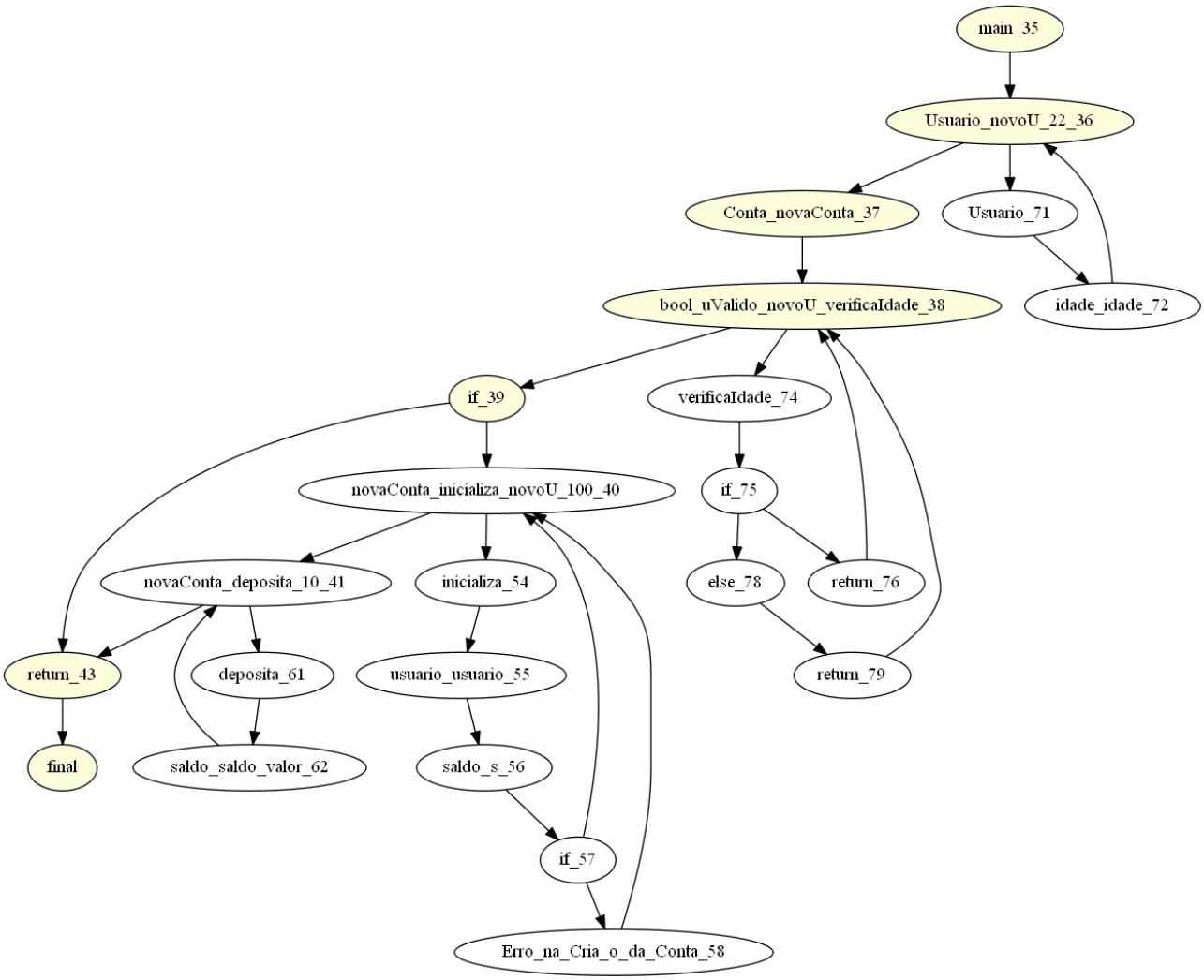
HH-CF	7	7	7	22	7	7	7	7	7	17
HHISE_M	22	7	7	2 0	7	7	2 2	7	7	7
HHISE_R	22	7	22	7	7	7	7	7	7	3
IBEA	4	7	7	7	7	7	7	7	7	7
MOMBI-II	7	7	7	7	7	7	7	22	6	6
NSGA-III	22	7	7	7	7	7	7	7	7	7
SPEA-II	7	7	7	7	7	7	7	7	7	0

Figura 3.14 - Caso de teste 4 - instância ACC1_3: STF.



Fonte: Produção do autor.

Figura 3.15 - Caso de teste 7 - instância ACC2_3: STF.



Fonte: Produção do autor.

3.4 Implementação do método InMeHy

O método InMeHy é um método genérico. Em outras palavras, o método em si não é uma ferramenta que foi desenvolvida em uma linguagem de programação particular. A notação algorítmica dos módulos serve para os seus melhores entendimentos, mas os mesmos podem ser implementados, a princípio, em qualquer linguagem de programação e/ou *framework*.

Duas ferramentas foram desenvolvidas e estão disponíveis livremente, sendo que uma implementa a estratégia STV (InMeHy_STV) (SALES, 2020) e a outra implementa

a estratégia STF (InMeHy_STF) (SALES, 2021). Naturalmente, como é usual, a concepção do método InMeHy se deu por uma definição inicial dos comportamentos/algoritmos de cada módulo e, conforme a implementação dos mesmos foi sendo realizada, esses algoritmos foram atualizados.

Para implementar o método, usou-se a linguagem de programação Java, o *framework* jMetal (DURILLO; NEBRO, 2011), a biblioteca JGrapht (MICHAIL et al., 2020) , o gerador de *parser* ANTLR (PARR, 2019) para gerar as árvores sintáticas, e a LRG é .dot.

3.5 Considerações finais sobre este capítulo

Neste capítulo foi apresentada o método InMeHy, descreveu-se cada um dos módulos que fazem parte do método, bem como qual processo é realizado para que seja possível realizar a geração de casos de Teste de integração, a partir de código-fonte em C++ e por meio de meta e hiper-heurísticas. Além da descrição do funcionamento também foi utilizado um exemplo de execução com o intuito de tornar mais claros os processos realizados em cada módulo. No próximo capítulo serão apresentadas algumas análises experimentais controladas realizadas para que fosse possível realizar uma avaliação do método.

4 EXPERIMENTOS CONTROLADOS

Neste capítulo, serão apresentadas as definições e as características dos experimentos controlados que foram conduzidos para avaliar o método InMeHy e os algoritmos de otimização. A avaliação será dividida em duas seções, uma para STV e outra para STF, Seções 4.1 e 4.2, respectivamente.

4.1 STV

4.1.1 Objetivo

O objetivo desta avaliação é identificar, utilizando suíte de testes com tamanho variáveis, qual dos quatro algoritmos de otimização é o melhor em relação à geração de casos de teste relacionados ao nível de Teste de integração. Foram consideradas classes de aplicativos C++ como os SSTs e os algoritmos IBEA, SPEA2, NSGA-III e MOMBI-II para gerar casos de teste. A razão pela escolha do algoritmo NSGA-III e não a sua versão anterior mais popular, NSGA-II, foi simplesmente porque se desejou observar o desempenho de um algoritmo já concebido para abordar problemas com inúmeros objetivos (NSGA-III) ao invés de usar o NSGA-II, o qual foi concebido para problemas multiobjetivo (embora o NSGA-II também tem sido usado para problemas com inúmeros objetivos).

Como mencionado anteriormente, levou-se em consideração três indicadores de qualidade para avaliar o desempenho dos algoritmos: hipervolume, indicador ϵ e IGD+.

4.1.2 Questões e variáveis de pesquisa

Foram definidas as seguintes questões de pesquisa (RQs) para serem respondidas:

RQ__1 - Qual dos algoritmos é o melhor em relação a cada indicador de qualidade?

RQ__2 - Existe um algoritmo que se destaca de todos os outros considerando todos os indicadores de qualidade?

As variáveis independentes são os algoritmos de otimização. As variáveis dependentes são os valores dos indicadores de qualidade: hipervolume, indicador ϵ e IGD +.

4.1.3 Problemas e níveis de integração

Aplicou-se o método InMeHy a dois softwares de geo-informática, desenvolvidos utilizando a linguagem de programação C++, em desenvolvimento no INPE, sendo eles, GeoDMA e TerraLib. O GeoDMA pode ser definido como uma caixa de ferramentas para integrar métodos de análise de imagens de sensoriamento remoto com técnicas de mineração de dados, com o objetivo de extrair informações e descoberta de conhecimento em grandes bancos de dados geográficos (KÖRTING et al., 2013). TerraLib, por sua vez, é uma biblioteca de Sistema de Informação Geográfica (SIG) para apoiar o desenvolvimento de aplicações geográficas customizadas (CÂMARA et al., 2008). É um aplicativo de código aberto, não trivial com mais de 1.400 classes.

Para cada um dos softwares escolhidos, foi definido valores diferentes de η. Para o GeoDMA, definiu-se $\eta = 2$ e um problema iniciado a partir do arquivo principal da aplicação "executar.cpp". Em outras palavras, foi realizada a integração dos arquivos de dois em dois. Dez arquivos foram considerados, resultando em cinco instâncias de problema (uma para cada etapa de integração). Tais instâncias de problemas foram denotadas como GEOx_2, uma vez que $\eta = 2$ e x é o número da instância. No entanto, a instância do problema GEO2_2 não alterou o grafo resultante, obtido na instância do problema anterior. Isso pode ocorrer devido a dois motivos, o primeiro sendo: os novos arquivos selecionados para integração não terem corpo, sendo por exemplo, arquivos de cabeçalho; ou por não estarem diretamente relacionados ao arquivo principal ou aos arquivos secundários. Portanto, a instância de problema GEO2_2 foi descartada da análise.

É importante ressaltar que o módulo Coletor simplesmente segue a sequência de arquivos a serem integrados, conforme definido na seção de importações dos arquivos contidos no SST, começando com o arquivo principal. Ou seja, o módulo não faz nenhum tipo de análise sobre a melhor forma de integrar as classes, já que o método InMeHy trata da geração de casos de teste e não da questão da integração e da ordem dos testes, como outros fizeram (GUIZZO et al., 2017).

Quanto à TerraLib, foram considerados dois problemas como estudos de caso. Um problema está relacionado à classe GeometryFactory que é responsável por instanciar as geometrias TerraLib de acordo com o tipo de geometria que recebe como parâmetro. O nível de integração foi definido como $\eta = 3$ (integração três por três). As instâncias do problema foram denotadas como TLGMx_3. No total, 14 instâncias de problemas foram criadas, mas TLGM9_3 foi eliminado devido a um dos motivos citados anteriormente, relacionado a instância GEO2_2. O outro problema

considera a classe TimePeriod, esta classe não possui associação com a classe GeometryFactory, sendo de módulos diferentes. Duas instâncias de problema foram criadas, denominadas como TLTP1_3 e TLTP2_3 ($\eta = 3$). Foram considerados 42 arquivos TerraLib no primeiro problema e seis arquivos no segundo problema, totalizando 48 arquivos. Vale ressaltar que o objetivo dos diferentes níveis de integração é apenas enfatizar a flexibilidade do método.

A Tabela 4.1 apresenta as características do três problemas selecionados, em termos de software, quantidade de instâncias e nível de integração adotado. A Tabela 4.2 mostra as características das 19 instâncias do problema dos três problemas em termos de vértices e arestas dos respectivos grafos. Como esperado, o número de vértices e arestas aumenta à medida que o processo de integração é realizado.

Tabela 4.1 - Características dos Problemas - STV.

Problema	Software	Quantidade de Instâncias geradas	Nível de Integração
Main	GeoDMA	5	2
GeometryFactory	TerraLib	14	3
TimePeriod	TerraLib	2	3
Total		52	

4.1.4 Parâmetros e execução dos algoritmos

A Tabela 4.3 mostra os valores dos principais parâmetros usados para todos os algoritmos (IBEA, SPEA2, NSGA-III, MOMBI-II). Alguns valores desses parâmetros são muito tradicionais na comunidade de otimização, como um tamanho de população de 100 e a execução de todas as abordagens para 30 tentativas para cada instância do problema.

No entanto, o tamanho da solução, ou seja, o número de variáveis de decisão, é grande (1.000). Isso ocorre porque o grafo pode se tornar muito grande ao se aprofundar no processo de integração e, portanto, é necessário ter um tamanho adequado para ser capaz de gerar um conjunto de testes significativo (solução). Este tamanho foi definido examinando o número de vértices das instâncias do problema (ver Tabela 4.2) e dando uma margem considerável.

Tabela 4.2 - Características das instâncias de problema úteis - STV.

Instâncias de Problema	Software	#Vértices	#Arestas
GEO1_2	GeoDMA	230	311
GEO3_2	GeoDMA	233	316
GEO4_2	GeoDMA	261	353
GEO5_2	GeoDMA	283	408
TLGM1_3	TerraLib	59	69
TLGM2_3	TerraLib	61	73
TLGM3_3	TerraLib	63	76
TLGM4_3	TerraLib	66	81
TLGM5_3	TerraLib	67	83
TLGM6_3	TerraLib	69	88
TLGM7_3	TerraLib	70	91
TLGM8_3	TerraLib	71	94
TLGM10_3	TerraLib	72	96
TLGM11_3	TerraLib	73	102
TLGM12_3	TerraLib	75	109
TLGM13_3	TerraLib	76	111
TLGM14_3	TerraLib	77	119
TLTP1_3	TerraLib	31	30
TLTP2_3	TerraLib	33	39

Tabela 4.3 - Avaliação experimental: valores dos parâmetros - STV.

Parâmetros	Valores
Tamanho da população	100
Número de variáveis de decisão	1.000
Probabilidade de Crossover	0,9
Probabilidade de Mutação	0,0125
Operador de Crossover	SBX
Operador de Mutação	Random
Quantidade de execuções	30
Número de Iterações	1.100

Legenda: SBX = Simulated Binary Crossover.

4.1.5 Tipos de avaliações

Foram realizadas dois tipos de avaliações. Para realizar a análise foram utilizados os valores normalizados dos indicadores. Deste ponto em diante, o **hipervolume normalizado por fronteira** será denominado simplesmente por **hipervolume**, h,

assim como ϵ, $I+$, correspondem ao indicador ϵ (normalizado por fronteira) e IGD +, respectivamente. Um ponto importante sobre os indicadores é a sua forma de avaliação, onde pode-se simplificar da seguinte forma, quanto maior o h, melhor é o algoritmo e quanto menor o ϵ e $I+$, melhor ele é.

A primeira forma de avaliação é a análise multi-domínio , que é adequada para obter evidências da capacidade de generalização dos algoritmos em todas as instâncias do problema, e não realizar uma avaliação caso a caso (uma instância de problema por vez). Em seguida, definiu-se um segundo nível de normalização dos indicadores. Por exemplo, o **hipervolume duplamente normalizado** (normalizado por fronteira e normalizado novamente) será denominado simplesmente de **hipervolume normalizado**, h_N, e é definido como abaixo (SANTIAGO JÚNIOR et al., 2020; LI et al., 2019):

$$h_N = \frac{h^{max}_{(\forall a,p)} - \overline{h_{(a_i,p)}}}{h^{max}_{(\forall a,p)} - h^{min}_{(\forall a,p)}} \tag{4.1}$$

onde $h^{max}_{(\forall a,p)}$ e $h^{min}_{(\forall a,p)}$ são os valores máximo e mínimo, respectivamente, do hipervolume, h , devido a todos os algoritmos a para uma instância de problema p, e $\overline{h_{(a_i,p)}}$ é o valor médio do hipervolume devido ao algoritmo a_i para p. No entanto, observe que a formulação de h_N é como um problema de maximização (maximizar hipervolume) transformado em um problema de minimização. Portanto, quanto menor o valor de h_N, melhor. Já para os demais indicadores, ϵ e $I+$, o valor normalizado é calculado de maneira padrão. Por exemplo, o IGD+ normalizado, $I+_N$, é obtido da seguinte forma:

$$I+_N = \frac{\overline{I+_{(a_i,p)}} - I+^{min}_{(\forall a,p)}}{I+^{max}_{(\forall a,p)} - I+^{min}_{(\forall a,p)}} \tag{4.2}$$

onde $I+^{max}_{(\forall a,p)}$ e $I+^{min}_{(\forall a,p)}$ são os valores máximo e mínimo, respectivamente, do IGD+, $I+$, devido a todos os algoritmos a para uma instância do problema p, e $\overline{I+_{(a_i,p)}}$ é o valor médio do IGD+ devido ao algoritmo a_i por p. Quanto menor $I+_N$, e também ϵ_N, melhor é o algoritmo.

A análise multi-domínio é então realizada calculando as médias dos indicadores de qualidade normalizados considerando todas as instâncias de problemas de todos os problemas: $\overline{h^{APR}_N}$, $\overline{\epsilon^{APR}_N}$, $\overline{I+^{APR}_N}$. Novamente, o algoritmo que obteve o menor desses valores é considerado o melhor no geral.

A segunda avaliação é uma análise estatística, mas agora utilizando os indicadores

(normalizados) h, ϵ e $I+$. É uma convicção que o uso do segundo grau de normalização pode mascarar os resultados do teste estatístico. Aplicou-se um Teste de permutação bicaudal (procedimento de inferência condicional) para comparação multi-grupo com nível de significância igual a 0,05. Além disso, agora é uma análise caso a caso, ou seja, será verificado para cada instância do problema se um algoritmo a_i era significativamente melhor ("$>$") do que um algoritmo a_j, se fosse pior ("$<$"), ou então se não houver diferença significativa ("$\sim$").

Esses dois tipos de análise fornecem perspectivas diferentes para determinar o melhor algoritmo de todos.

4.1.6 Validade

Os resultados apresentados estão associados à versão 6 do framework jMetal onde os algoritmos e indicadores de qualidade são implementados. Acredita-se que a replicação deste estudo por outros pesquisadores produzirá resultados semelhantes.

As instâncias de problema são conjuntos de arquivos/classes integrados dos softwares GeoDMA e TerraLib, portanto não havia nenhum fator humano/natureza/social nem eventos imprevistos que interrompessem a coleta das medidas, uma vez que passassem a ter validade interna. Consequentemente, o experimento controlado tem uma alta validade interna.

Mesmo que os dois estudos de caso sejam sistemas não triviais, tem-se uma ameaça à validade externa relacionada à população. As instâncias do problema selecionadas são interessantes, mas primeiro precisa-se integrar mais classes dos dois sistemas e, posteriormente, considerar outros aplicativos C ++. No entanto, os resultados deste experimento controlado são valiosos, conforme mostrado na Seção 5.

4.2 STF

4.2.1 Objetivo

O objetivo continua sendo o mesmo da seção anterior, identificar qual dos algoritmos de otimização é o melhor em relação à geração de casos de teste relacionados ao nível de Teste de integração, mas desta vez a análise será realizada utilizando suíte de casos de teste com tamanho fixo, e fazendo o uso além dos quartos citados anteriormente, também das hiper-heurísticas escolhidas, sendo elas HRISE_M, HRISE_R e Choice-Function. Levando em consideração os mesmos indicadores de qualidade.

4.2.2 Questões e variáveis de pesquisa

Foram utilizadas as mesmas questões de pesquisa (RQs) definidas anteriormente, mas agora adicionando mais uma questão:

RQ_3 - Qual dos três algoritmos de hiper-heurísticas, HRISE_M, HRISE_R e Choice-Function apresentam melhor desempenho?

As variáveis independentes e dependentes continuam as mesmas.

4.2.3 Problemas e níveis de integração

Assim como na seção anterior os problemas foram retirados dos softwares GeoDMA e TerraLib, entretanto alguns problemas e instâncias de problema foram alterados, além da alteração no nível de integração.

Quanto ao GeoDMA, manteve a utilização do problema iniciando a partir do arquivo principal, mas desta vez considerando $\eta = 4$. Ou seja, agora a integração é realizada de quatro em quatro arquivos. Dezesseis arquivos foram considerados, resultando em quatro instâncias de problema (uma para cada etapa de integração). Tais instâncias de problemas foram denominadas como GEOx_4, uma vez que $\eta = 4$. Neste caso foram geradas 4 instâncias de problema, utilizando 16 arquivos.

Já em relação ao TerraLib, foram considerados oito problemas. Os dois problemas considerados anteriormente, GeometryFactory e TimePeriod, também foram utilizados aqui, mas com nível de integração diferentes, desta vez utilizou-se $\eta = 4$ (integração de quatro em quatro). Além destes problemas foram adicionados os seguintes problemas que representam algumas geometrias na aplicação, sendo elas, CirculasString, LineString, MultiCurve, MultiPolygon e Point, e também problemas ligados ao tipo de datas, OrdinalPeriod e TimeInstant.

Como nos problemas do software TerraLib obteve-se mais problemas a analisar, não foi possível seguir o mesmo padrão do GeoDMA, deste modo, as instâncias do problema foram denominadas da seguinte maneira, TL(*sigla_do_problema*)x_4, onde a *sigla_do_problema* é formada pelas iniciais de cada palavra do problema, por exemplo CirculasString = CS, LineString = LS, como a exceção a esta nomenclatura, pode-se elencar o problema Point, uma vez que este problema tem apenas uma inicial será denominando de TLPx_4.

Na Tabela 4.4 pode-se observar que ao total foram geradas 71 instâncias de pro-

blema, mas destas apenas 42 foram consideradas úteis, devido aos mesmos casos explicados anteriormente. Algumas das instâncias não úteis são, por exemplo, algumas do problema TimePeriod, que gerou 5 instâncias de problema, entretanto apenas na primeira instância de problema observou-se a integração dos arquivos, as demais instâncias permaneceram com a mesma estrutura, desta forma foram descartadas na realização da análise experimental.

Um outro caso que pode ocorrer para a não utilização da instância de problema acontece quando a instância de problema em construção não gera muitos passeios por conter uma baixa complexidade, o que impossibilita a geração dos casos de teste utilizando as meta e hiper-heurísticas. Isto ocorreu em alguns problemas escolhidos, sendo assim das 42 instancias úteis, 3 delas eram muito simples, resultando assim em 39 instancias de problemas. Por exemplo, OrdinalPeriod e TimeInstant geraram duas instâncias úteis, mas quando foi realizada a geração dos casos de teste, as duas instâncias de TimeInstant e a primeira de OrdinalPeriod não tinham complexidade suficiente para gerar passeios distintos, então foram descartadas.

Tabela 4.4 - Características dos Problemas - STF.

Problema	Software	Instâncias geradas	Instâncias úteis	Nível de Integração
Main	GeoDMA	4	4	4
CompoundCurve	TerraLib	7	3	4
CircularString	TerraLib	6	5	4
GeometryCollection	TerraLib	5	2	4
GeometryFactory	TerraLib	7	7	4
LineString	TerraLib	6	4	4
MultiCurve	TerraLib	6	3	4
MultiPoligon	TerraLib	7	4	4
OrdinalPeriod	TerraLib	4	2	4
Point	TerraLib	5	3	4
PolyhedralSurface	TerraLib	5	2	4
TimeInstant	TerraLib	4	2	4
TimePeriod	TerraLib	5	1	4
Total		71	42	

A Tabela 4.5 mostra as características destas 39 instâncias do problema dos dez problemas em termos de vértices e arestas dos respectivos grafos. Como esperado, o número de vértices e arestas aumenta à medida que avança-se no processo de integração, além de mostrar a quantidade de passeios gerados para cada uma das

instâncias de problemas.

4.2.4 Parâmetros e execução dos algoritmos

Com a alteração da formação dos casos de teste algumas variáveis foram alteradas com relação a Tabela 4.3, a Tabela 4.6 mostra os novos valores dos principais parâmetros usados para todos os algoritmos (IBEA, SPEA2, NSGA-III, MOMBI-II, HRISE_M, HRISE_R e Choice-Function). Foi mantido o tamanho de população de 100, mas pode-se observar que o número de variáveis de decisão diminuiu bastante, isso se deve ao fato de agora cada individuo representar um caso de teste e não apenas faz parte do caso de teste.

4.3 Considerações finais sobre este capítulo

Neste capítulo foram apresentadas as avaliações experimentais que foram realizadas com o intuito de coletar dados para que fosse possível avaliar o método proposto. Foi apresentado os casos de teste com tamanhos variáveis e fixos, onde no primeiro foram utilizadas apenas as meta-heurísticas e no segundo foram utilizadas as meta-heurísticas juntamente com as hiper-heurísticas. Apresentou-se as questões de pesquisas formuladas para cada um dos dois casos, bem como os valores dos parâmetros passados para os algoritmos, os tipos de avaliações que forma utilizados e quais foram os estudos de caso escolhidos para realizar estas avaliações.

Tabela 4.5 - Características das instâncias de problema úteis - STF.

Instância de Problema	nome	Software	#Vértices	#Arestas	Passeios gerados
1	GEO1_4	GeoDMA	229	330	10.026
2	GEO2_4	GeoDMA	266	412	10.032
3	GEO3_4	GeoDMA	436	729	10.025
4	GEO4_4	GeoDMA	451	769	10.031
5	TLCC1_4	TerraLib	117	140	10.007
6	TLCC2_4	TerraLib	123	151	10.006
7	TLCC6_4	TerraLib	125	158	10.006
8	TLCS1_4	TerraLib	195	232	10.007
9	TLCS2_4	TerraLib	202	241	10.007
10	TLCS3_4	TerraLib	226	335	10.005
11	TLCS4_4	TerraLib	229	342	10.005
12	TLCS6_4	TerraLib	252	393	10.007
13	TLGC1_4	TerraLib	117	148	10.006
14	TLGC2_4	TerraLib	133	171	10.006
15	TLGF1_4	TerraLib	59	69	25
16	TLGF2_4	TerraLib	63	75	27
17	TLGF3_4	TerraLib	72	86	29
18	TLGF4_4	TerraLib	78	95	34
19	TLGF5_4	TerraLib	86	106	37
20	TLGF6_4	TerraLib	90	114	43
21	TLGF7_4	TerraLib	92	118	45
22	TLLS1_4	TerraLib	236	281	10.006
23	TLLS2_4	TerraLib	246	298	10.006
24	TLLS3_4	TerraLib	268	351	10.006
25	TLLS6_4	TerraLib	300	420	10.006
26	TLMC1_4	TerraLib	32	44	468
27	TLMC2_4	TerraLib	33	47	8.147
28	TLMC3_4	TerraLib	36	53	10.004
29	TLMP1_4	TerraLib	31	35	47
30	TLMP2_4	TerraLib	41	50	531
31	TLMP3_4	TerraLib	42	54	2.599
32	TLMP5_4	TerraLib	45	59	10.004
33	TLOP2_4	TerraLib	31	35	15
34	TLP1_4	TerraLib	92	114	10.007
35	TLP2_4	TerraLib	102	126	10.007
36	TLP3_4	TerraLib	110	148	10.007
37	TLPS1_4	TerraLib	100	126	10.006
38	TLPS2_4	TerraLib	117	149	10.006
39	TLTP1_4	TerraLib	31	37	65

Tabela 4.6 - Avaliação experimental: valores dos parâmetros - STF.

Parâmetros	Valores
Tamanho da população	100
Número de variáveis de decisão	10
Probabilidade de Crossover	0,9
Probabilidade de Mutação	0,0125
Operador de Crossover	SBX
Operador de Mutação	Integer Polynomial
Quantidade de execuções	20
Número de Iterações	1.000

Legenda: SBX = Simulated Binary Crossover.

5 RESULTADOS E ANÁLISES

Neste capitulo serão apresentados os resultados da aplicação do método com as duas variações de suítes de teste, STV e STF. Os tipos de avaliações que foram aplicados estão de acordo com o descrito na seção 4.1.5.

5.1 Caso STV

Nesta subseção serão apresentados e discutidos os resultados da avaliação experimental que foi realizado utilizando a estratégia STV (SALES; SANTIAGO JÚNIOR, 2020). A Tabela 5.1 mostra os resultados da análise entre domínios usando como métrica o hipervolume normalizado, h_N. Nesta tabela, $\overline{h_N^{GEO}}$, $\overline{h_N^{TLGM}}$, e $\overline{h_N^{TLTP}}$ são os valores médios de h_N para o GeoDMA, TerraLib GeometryFactory e TerraLib TimePeriod instâncias de problemas, respectivamente. A média de todas as 19 instâncias do problema é $\overline{h_N^{APR}}$ que é a medida mais importante que representa todas as instâncias do problema de todos os problemas. Em um plano de fundo cinza, mostra-se as melhores abordagens de desempenho, onde em negrito é o melhor algoritmo e em itálico é o segundo melhor.

Pode-se observar que o IBEA foi o melhor algoritmo para todos os problemas individuais (GEO, TLGM, TLTP). SPEA2 foi o segundo melhor considerando o problema GEO, mas NSGA-III foi o segundo melhor quando analisa-se os problemas TLGM e TLTP. Quanto a todas as instâncias de problemas de todos os problemas, o IBEA foi o melhor, o NSGA-III ficou na segunda posição, seguido pelo SPEA2.

Na Tabela 5.2 é feita a apresentação dos resultados considerando o ϵ normalizado (ϵ_N) e IGD + ($I+_N$), mas aqui é mostrado apenas o valor médio do indicador ϵ normalizado ($\overline{\epsilon_N^{APR}}$) e IGD + ($\overline{I+_N^{APR}}$) para todas as instâncias de problemas de todos os problemas. É possível ver que o IBEA é novamente a abordagem que obteve melhor resultado em ambos os indicadores de qualidade, o SPEA2 é o segundo e NSGA-III é o terceiro em ambos os indicadores também.

Tabela 5.1 - STV, análise multi-domínio: h_N.

	IBEA	SPEA2	NSGA-III	MOMBI-II
GEO1_2	0.10160208	0.2463683	0.4289033	0.9822310
GEO3_2	0.11461707	0.2715744	0.3999732	0.9784607
GEO4_2	0.10184463	0.2663812	0.3910215	0.9945737
GEO5_2	0.08240685	0.1877198	0.3780523	0.9727308
$\overline{h_N^{GEO}}$	**0.10011766**	*0.2430109*	0.3994876	0.9819990
TLGM1_3	0.08498497	0.3541394	0.2378632	0.9566168
TLGM2_3	0.06730111	0.3942012	0.2594165	0.9259537
TLGM3_3	0.08385168	0.4094094	0.2772726	0.9418491
TLGM4_3	0.14113080	0.4672814	0.3677585	0.9550806
TLGM5_3	0.04555045	0.4083426	0.3090646	0.9241510
TLGM6_3	0.07847350	0.4169585	0.2904767	0.8869645
TLGM7_3	0.08734122	0.4306034	0.3395103	0.8903919
TLGM8_3	0.08662291	0.4454747	0.3327799	0.9003701
TLGM10_3	0.10511715	0.4241922	0.3368683	0.8787976
TLGM11_3	0.07326611	0.4267992	0.3351720	0.8869524
TLGM12_3	0.08209617	0.4501841	0.3284261	0.9145412
TLGM13_3	0.07658653	0.4114851	0.2950148	0.9152460
TLGM14_3	0.10760963	0.4208683	0.3115025	0.9108875
$\overline{h_N^{TLGM}}$	**0.08614863**	0.4199953	*0.3093174*	0.9144463
TLTP1_3	0.10805197	0.4689310	0.3032761	0.8015967
TLTP2_3	0.09739999	0.3680910	0.2123916	0.8052245
$\overline{h_N^{TLTP}}$	**0.10272598**	0.4185110	*0.2578339*	0.8034106
$\overline{h_N^{APR}}$	**0.09083446**	0.3825792	*0.3228813*	0.9169800

Tabela 5.2 - STV, análise multi-domínio: ϵ_N e $I+_N$.

	IBEA	SPEA2	NSGA-III	MOMBI-II
$\overline{\epsilon_N^{APR}}$	**0.06923602**	*0.2767582*	0.2825046	0.9128542
$\overline{I+_N^{APR}}$	**0.02503317**	*0.18086536*	0.18984178	0.8465902

A avaliação estatística está resumida na Tabela 5.3, onde "$>$" significa que o algoritmo mais à esquerda foi significativamente melhor do que o mais à direita, "$<$" significa que o algoritmo mais à esquerda foi significativamente pior do que o mais à direita um, e "$\sim$" significa nenhuma diferença significativa. É importante ressaltar de que aqui não foi utilizado os indicadores normalizados. Além disso, o número que

aparece na tabela significa em quantas vezes, das 19 instâncias, o algoritmo mais à esquerda foi significativamente melhor do que o mais à direita, em quantas instâncias do problema não há diferença estatística, e também em quantas vezes o o mais à esquerda era o pior. Por exemplo, em relação a $I+$, pode-se observar que SPEA2 foi significativamente melhor do que NSGA-III em seis casos de problema, em quatro casos não há diferença estatística e em nove casos de problema o NSGA-III foi significativamente melhor do que SPEA2.

Tabela 5.3 - STV, análise estatística.

Comparação	$>$	$\sim$	$<$	$>$	$\sim$	$<$	$>$	$\sim$	$<$
	h			ϵ			$I+$		
IBEA × SPEA2	19	0	0	19	0	0	19	0	0
IBEA × NSGA-III	19	0	0	18	1	0	19	0	0
IBEA × MOMBI-II	19	0	0	19	0	0	19	0	0
SPEA2 × NSGA-III	4	0	15	12	5	2	6	4	9
SPEA2 × MOMBI-II	19	0	0	19	0	0	19	0	0
NSGA-III × MOMBI-II	19	0	0	19	0	0	19	0	0

Em termos de h, os resultados estatísticos concordam com os resultados anteriores com base na análise multi-domínio em que IBEA foi o melhor seguido por NSGA-III e SPEA2 foi o terceiro. O mesmo ocorre analisando ϵ na seguinte ordem: IBEA, SPEA2, NSGA-III. No entanto, em termos de $I+$, embora IBEA fosse novamente a estratégia principal, NSGA-III foi melhor do que SPEA2 (9 6 vitórias).

A resposta para RQ_1 é então clara. O IBEA foi absoluto, obtendo o melhor desempenho nos dois tipos de análise para cada indicador individualmente. Consequentemente, isso também responde RQ_2 como um único algoritmo (IBEA) alcançou o melhor desempenho para todos os indicadores. Este resultado é interessante porque é comum em avaliações envolvendo AEs que um determinado algoritmo obtenha o melhor desempenho para alguns, mas não para todos os indicadores.

Além disso, esses resultados mostram a necessidade de conduzir muito mais experimentos, não apenas dentro dos testes de software, mas também considerando outros problemas, para abordar a questão da generalização. Algoritmos considerados “antigos", como o IBEA e o SPEA2, ainda são extremamente competitivos e, como demonstrado por esta pesquisa, podem se destacar em relação às abordagens mais recentes, e até mesmo visando resolver problemas de Muitos Objetivos como NSGA-III e MOMBI -II. Finalmente, é importante notar o mau desempenho do MOMBI-II,

uma abordagem recente baseada no indicador R2 como um mecanismo de seleção individual. Em algumas instâncias do problema, por exemplo, aquelas do problema GeoDMA, a variabilidade das soluções em uma população é muito baixa com os valores das funções objetivo basicamente os mesmos. Não está claro o motivo para este fato requerer mais investigação.

5.2 Caso STF

Nesta subseção será apresentado os resultados utilizando STF. Na Tabela 5.4 pode-se observar a quantidade de casos de testes distintos, para cada instância de problema, presentes nas soluções de cada algoritmo e também levando em conta todos os algoritmos. Pode-se constatar que as hiper-heurísticas apresentam maior diversidade de casos de teste quando comparadas com as meta-heurísticas. Se utilizada a média de todas as instâncias de problema tem-se a seguinte ordem do algoritmo com mais diversidade para o com menos diversidade: HRISE_M, HRISE_R, HH-CF, SPEA-II, IBEA, NSGA-III e por fim MOMBI-II, este último sendo bem menor que as outras meta-heurísticas.

A Tabela 5.5 mostra os resultados da análise multi-domínio considerando todos os indicadores de qualidade normalizados. A média de todas as 39 instâncias de problemas. Em um plano de fundo cinza, mostra-se as abordagens que obtiveram melhor desempenho, onde em negrito é o melhor algoritmo e em itálico é o segundo melhor. De acordo com os resultados, pode-se observar que o SPEA2 obteve o melhor desempenho para todos os indicadores de qualidade normalizados, seguido pelo IBEA em todos os casos. Quanto ao hipervolume normalizado e IGD +, pode-se elencar a seguinte ordem após IBEA: NSGA-III, HRISE_R, HRISE_M, HH-CF e MOMBI-II. Considerando o indicador normalizado, a ordem após a segunda posição é: HRISE_R, HRISE_M, NSGA-III, HH-CF e MOMBI-II.

Tabela 5.4 - STF, diversidade de casos de teste.

IP	IBEA	SPEA-II	NSGA-III	MOMBI-II	HRISE_M	HRISE_R	HH-CF	Total
GEO1_4	286	578	144	67	991	964	697	2358
GEO2_4	309	560	241	64	1078	1154	722	2288
GEO3_4	274	615	102	48	1737	1614	1170	3619
GEO4_4	186	296	118	21	1759	1596	1252	3578
TLCC1_4	292	440	226	69	508	489	422	1542
TLCC2_4	241	394	240	102	479	499	363	142
TLCC6_4	237	481	258	103	507	465	420	1582
TLCS1_4	267	467	214	64	481	480	346	1268
TLCS2_4	182	293	137	59	459	472	351	1124
TLCS3_4	255	469	146	98	1132	1082	797	2456
TLCS4_4	294	491	176	73	981	1117	813	2541
TLCS6_4	118	268	170	60	713	725	602	1563
TLGC1_4	287	497	196	45	516	511	457	1433
TLGC2_4	197	266	175	66	427	435	391	1084
TLGF1_4	25	25	24	17	25	25	25	25
TLGF2_4	26	26	26	17	27	26	27	27
TLGF3_4	28	28	27	16	29	29	28	29
TLGF4_4	29	32	28	17	34	33	31	34
TLGF5_4	30	37	32	15	37	37	37	37
TLGF6_4	29	38	26	16	40	41	39	42
TLGF7_4	29	42	32	15	45	43	44	45
TLLS1_4	288	591	139	87	420	412	312	1351
TLLS2_4	242	462	124	81	553	566	377	1489
TLLS3_4	178	312	134	82	596	534	502	1348
TLLS6_4	198	264	151	80	466	464	320	1110
TLMC1_4	32	35	63	22	73	78	72	139
TLMC2_4	120	126	133	114	275	239	249	612
TLMC3_4	108	92	85	110	282	269	223	602
TLMP1_4	18	15	15	8	23	24	20	25
TLMP2_4	60	89	65	25	90	92	80	151
TLMP3_4	150	174	158	59	217	197	182	482
TLMP5_4	210	239	216	88	315	370	301	852
TLOP2_4	8	8	8	4	12	12	12	13
TLP1_4	232	369	195	72	539	572	441	1521
TLP2_4	240	384	146	42	614	549	479	1537
TLP3_4	270	296	157	100	710	687	526	1617
TLPS1_4	267	392	158	55	364	424	340	1168
TLPS2_4	229	290	202	60	366	416	280	1076
TLTP1_4	15	20	21	12	23	17	18	32
média	166	269	125	55	460	455	353	1075

Tabela 5.5 - STF, análise multi-domínio: h_N, ϵ_N e $I+_N$.

	IBEA	SPEA2	NSGA-III	MOMBI-II	HRISE_M	HRISE_R	HH-CF
$\overline{h_N^{APR}}$	*0.29792359*	**0.238519808**	0.45186977	0.8374130	0.50122582	0.49471330	0.6076469
$\overline{\epsilon_N^{APR}}$	*0.15705460*	**0.114554383**	0.31899868	0.6687436	0.26708378	0.25946213	0.37275394
$\overline{I+_N^{APR}}$	*0.021752925*	**1.182022e-02**	0.13053346	0.50062134	0.273439244	0.26669285	0.36972155

Na Tabela 5.6, apresenta-se os resultados da avaliação estatística em que, assim como na Tabela 5.3 da subseção anterior, "$>$" significa que o algoritmo mais à esquerda foi significativamente melhor do que o mais à direita, "$<$" significa que o algoritmo mais à esquerda foi significativamente pior que o mais à direita e " $\sim$" significa sem diferença significativa. O número que aparece na tabela significa em quantas vezes, das 39 instâncias do problema, o algoritmo mais à esquerda foi significativamente melhor do que o mais à direita, em quantas instâncias do problema não há diferença estatística, e também em quantas vezes o mais à esquerda foi o pior. Por exemplo, considerando h, IBEA não foi significativamente melhor do que SPEA2 em nenhuma instância de problema, em 22 instâncias não há diferença estatística e em 17 instâncias de problema SPEA2 foi significativamente melhor do que o IBEA.

Similarmente, na Tabela 5.6, pode-se ver que o SPEA2 é o melhor de todos seguido pelo IBEA em todos os indicadores. Mas, aqui vale analisar mais o resultado entre a HRISE_M, HRISE_R e NSGA-III (não parece ter um claro vencedor; analisar por indicador). Mas, certamente HRISE_M e HRISE_R são as melhores das HHs.

Em termos de h, os resultados estatísticos também apontam que o SPEA2 foi o melhor seguido pelo IBEA. No entanto, observe que há basicamente um empate em termos de desempenho entre NSGA-III, HRISE_M e HRISE_R (NSGA-III é ligeiramente melhor do que HRISE_R, mas há um empate entre NSGA-III e HRISE M). Olhando para os valores médios na Tabela 5.5, pode-se observar que o desempenho desses três algoritmos é próximo. Assim, também pode-se dizer que existe uma concordância entre a análise multi-domínio e esta avaliação estatística, que é uma análise caso a caso.

A respeito, pode-se dizer que há total concordância com a análise anterior com o SPEA2, sendo o IBEA o algoritmo de topo, seguido do HRISE_R, HRISE_M e NSGA-III. Existe um acordo total ao analisar I+ onde, novamente, SPEA2 foi

a abordagem superior e IBEA foi a segunda melhor. Além disso, observe que em termos de I+, NSGA-III é melhor do que HRISE_R e HRISE_M, que também é o caso na análise multi-domínio.

Tabela 5.6 - STF, análise estatística.

Comparação	$>$	$\sim$	$<$	$>$	$\sim$	$<$	$>$	$\sim$	$<$
	h			ϵ			$IGD+$		
IBEA × SPEA2	0	22	17	0	22	17	0	25	14
IBEA × NSGA-III	30	9	0	28	11	0	34	5	0
IBEA × MOMBI-II	39	0	0	39	0	0	39	0	0
IBEA × HRISE_M	23	9	7	21	15	3	37	2	0
IBEA × HRISE_R	24	9	6	21	14	4	37	2	0
IBEA × HH-CF	32	6	1	35	4	0	38	1	0
SPEA2 × NSGA-III	37	2	0	37	2	0	39	0	0
SPEA2 × MOMBI-II	39	0	0	39	0	0	39	0	0
SPEA2 × HRISE_M	27	6	6	28	10	1	38	1	0
SPEA2 × HRISE_R	26	7	6	29	8	2	38	1	0
SPEA2 × HH-CF	33	6	0	38	1	0	38	1	0
NSGA-III × MOMBI-II	39	0	0	39	0	0	39	0	0
NSGA-III × HRISE_M	14	11	14	14	8	17	25	2	12
NSGA-III × HRISE_R	16	8	15	14	9	16	25	2	12
NSGA-III × HH-CF	19	14	6	19	9	11	27	8	4
MOMBI-II × HRISE_M	2	7	30	4	4	31	9	11	19
MOMBI-II × HRISE_R	1	7	31	4	3	32	11	5	23
MOMBI-II × HH-CF	4	11	24	4	8	27	14	6	19
HRISE_M × HRISE_R	2	35	2	0	38	1	2	35	2
HRISE_M × HH-CF	18	21	0	16	23	0	17	22	0
HRISE_R × HH-CF	22	16	1	21	17	1	20	18	1

Assim, pode-se responder à **RQ_1** afirmando que o SPEA2 foi a melhor solução, obtendo a primeira posição em todas as avaliações. Consequentemente, isso responde também à **RQ_2**, pois um único algoritmo (SPEA2) obteve o melhor desempenho considerando todos os indicadores de qualidade. Com relação à **RQ_3**, HRISE_R e HRISE_M foram as melhores hiper-heurísticas com HRISE_R apresentando um desempenho ligeiramente melhor.

5.3 Validação das hipóteses

A seguir são relembradas as hipóteses que se desejava validar nessa pesquisa, apresentadas na Seção 1.2.

Hipótese 1: É viável realizar a geração de casos de Teste de integração para aplicações desenvolvidas em C++ considerando algoritmos de otimização.

Hipótese 2: As hiper-heurísticas apresentam um melhor desempenho do que as meta-heurísticas.

Em relação à **Hipótese 1**, pode-se dizer que essa foi aceita. Algoritmos de otimização foram perfeitamente viáveis para gerar casos de Teste de integração, conforme demonstrado anteriormente. Entretanto é importante ressaltar que naturalmente existe um problema de escalabilidade com basicamente qualquer abordagem que se baseia em modelos (no caso grafo) para gerar casos de teste. Sendo assim está abordagem é viável mas não escalável. Mas, perceber que o desafio era gerar casos de Teste de integração a partir do código-fonte em uma abordagem caixa branca. No caso do STF, conseguiu-se considerar um total de 85 arquivos e gerou-se grafos com 451 vértices e 769 arestas. Notar que o uso de passeios direcionados, nesse caso, além de manter a coerência dos casos de teste, também auxiliou no problema de escalabilidade, por meio da restrição do número máximo de trilhas direcionadas.

Com respeito à **Hipótese 2**, a mesma foi rejeitada no contexto dos experimentos realizados, considerando os algoritmos e estudos de caso selecionados. Em outras palavras, as hiper-heurísticas tiveram desempenho pior do que as meta-heurísticas, principalmente as clássicas (antigas). Além disso observe que, na seção anterior utilizando STV quando foram avaliadas as quatro meta-heurísticas apenas, o IBEA foi a abordagem com melhor desempenho, seguido pelo NSGA-III que foi ligeiramente melhor do que o SPEA2.

Na estratégia STF, como mencionado acima, o SPEA2 foi o melhor seguido pelo IBEA. Mas aqui está claro que o IBEA é muito melhor do que o NSGA-III, que pode ser considerada a terceira melhor abordagem. Observe que em ambas as seções (STV e STF), utilizou-se problemas com características semelhantes: otimização discreta, funções objetivo que são medidas sobre grafos, e duas das três funções objetivo são iguais.

Apesar de todos esses fatos, tem-se resultados diferentes ao analisar as melhores abordagens. Com esse raciocínio, pode-se destacar como uma das principais contribuições dessa pesquisa: é realmente difícil para um algoritmo de otimização atingir um alto nível de generalização. Hiper-heurísticas tais como HRISE_R e HRISE_M obtiveram melhores desempenhos para problemas de otimização contínua em comparação com meta-heurísticas como SPEA2 e IBEA (SANTIAGO JÚNIOR et al., 2020),

mas aqui as duas últimas foram as mais proeminentes.

Note-se também que, considerando os dois contextos STV e STF, pode-se dizer que, de um modo geral, as meta-heurísticas multiobjetivo mais antigas (IBEA, SPEA2) foram melhores do que aquelas para inúmeros objetivos (MAEAs) mais recentes (NSGA-III, MOMBI-II).

É importante enfatizar que ainda não se pode generalizar as conclusões sobre a resposta ao problema da generalização dos algoritmos de otimização. Em outras palavras, a rejeição da **Hipótese 2** não pode ser entendida como um fato de que as meta-heurísticas multiobjetivo (clássicas) são realmente melhores do que as hiper-heurísticas de seleção e do que as meta-heurísticas para inúmeros objetivos, para qualquer tipo de problema. Como mencionado anteriormente, existem diversos relatos na literatura que demonstram um melhor desempenho das hiper-heurísticas de seleção e dos algoritmos MAEAs em comparação com as MOEAs clássicas, assim como existe uma quantidade significativa de algoritmos propostos, tanto hiper-heurísticas de seleção como MAEAs.

Observe que essa conclusão, baseada em evidências empíricas, parece estar relacionada aos conhecidos **teoremas sem almoço grátis** (*no free lunch theorems*), que afirmam que quaisquer dois algoritmos são equivalentes quando a média dos seus desempenhos são obtidas considerando todos os problemas possíveis (WOLPERT; MACREADY, 2005). Portanto, é interessante verificar experimentalmente se as teorias propostas são válidas, e parece ser o caso aqui com base nos resultados que foram obtidos.

Dessa forma, é importante continuar com mais experimentos, considerando problemas de otimização discretos e contínuos, bem como problemas reais e teóricos, para obter uma resposta em termos de generalização na prática.

5.4 Considerações finais sobre este capítulo

Nesse capítulo foram apresentados os resultados e a análise da aplicação do método, em suas duas versões, utilizando os três indicadores de qualidade citados e os algoritmos escolhidos. Por meio dos resultados obtidos para cada instância de problema, pode-se concluir que a **Hipótese 1** foi aceita, mostrando assim que é viável realizar a geração de casos de testes a nível de integração por meio de algoritmos de otimização. Já com respeito a **Hipótese 2**, a mesma foi rejeitada no contexto dos experimentos realizados, uma vez que as meta-heurísticas, principalmente as clássicas, obtiveram

um resultado melhor do que as hiper-heurísticas selecionadas. É importante ressaltar que o resultado obtido na primeira experimentação não se repete no segundo, mesmo quando considerados apenas as meta-heurísticas. Além disso, a rejeição da **Hipótese 2** não deve ser vista como um demérito para as hiper-heurísticas ou para as meta-heurísticas para inúmeros objetivos.

6 TRANSFORMAÇÃO E EXECUÇÃO DOS CASOS DE TESTE

Conforme mencionado na Seção 7.1, as ferramentas desenvolvidas para apoiar o método InMeHy ainda não geram os casos de teste executáveis de forma automática. Isso se deve, principalmente, pela complexidade dos parâmetros que podem vir a ser utilizados nas aplicações. Além disso, não foi possível executar os casos de teste para as aplicações do INPE justamente devido ao acoplamento das classes das aplicações usadas como estudo de caso, que além de estarem prontas, não são triviais.

No entanto, esse capítulo apresenta uma forma de, com base nos casos de testes abstratos gerados pela ferramenta que implementa o método InMeHy, na versão STF, gerar os casos de teste executáveis de fato e executá-los, considerando estudos de casos mais simples. Com isso, pretende-se demonstrar a viabilidade do método proposto, caso o mesmo seja usado em um processo de desenvolvimento de software, onde o mesmo ainda esteja em construção (não "completo").

Este processo de iteração é feito a partir de algumas informações, como o conjunto de suítes de teste geradas pela ferramenta InMeHy, e, o arquivo principal utilizado para geração, como observado no Algoritmo 10. Após isso, é feita a verificação do arquivo passado como principal, para identificar se ele contém a função *main*, onde, caso contenha, os procedimentos contidos na função serão transformados de maneira a tornar possível realizar a chamada a cada um dos processos contidos nela e comparar seus retornos.

Após a verificação, mais uma iteração é realizada. Desta vez, sobre as suítes de testes contidas nos conjuntos anteriormente iterado, obtendo assim, cada caso de teste que deverá ser convertido. Neste momento, tal caso de teste ainda é representado por um número, por isso na linha 8, é realizada a busca do caso de teste pelo número contido na suíte de casos de teste. Com o caso de teste obtido, é necessário fazer a coleta das variáveis. A partir daí, será necessário aplicar alguma técnica para realizar a geração dos valores que vão ser passados para realizar os casos de teste (Algoritmo 11), como por exemplo, a técnica de análise de valor limite (MALDONADO et al., 2004). Por fim, são gerados os casos de teste para o conjunto de variáveis gerado, e estes casos de

testes são armazenados na variável *suíte_teste_executável*.

Algoritmo 10: EXECUTAR CASOS DE TESTE

Entrada: *conjunto_de_suíte_de_teste*, *arquivo_principal*
Saída: *suíte_teste_executável*

```
1  início
2      para cada suíte_de_teste ∈ conjunto_de_suíte_de_teste faça
3          se contémMain(arquivo_principal) então
4              coleta procedimento do main;
5              simula chamadas aos procedimentos no main;
6          fim
7          para cada número_caso_de_teste ∈ suíte_de_teste faça
8              caso_de_teste ← buscaCasoDeTeste(número_caso_de_teste)
9              conjunto_de_variáveis ←
                buscaConjuntoDeVariáveis(caso_de_test)
10             para cada conjunto_de_variável ∈ conjunto_de_variáveis faça
11                 caso_de_teste_executável ←
                    gerarCTE(caso_de_teste, conjunto_de_variável)
12                 suíte_teste_executável ←
                    suíte_teste_executável ⋃ caso_de_teste_executável
13             fim
14         fim
15     fim
16     retorna suíte_teste_executável
17 fim
```

Algoritmo 11: BUSCACONJUNTODEVARIÁVEIS

Entrada: *caso_de_teste*
Saída: *conjunto_de_variáveis*

```
1  início
2      variáveis ← coletaVariáveis(caso_de_teste)
3      para cada variável ∈ variáveis faça
4          conjunto_de_variáveis[variável] ←
            aplicaTecnicaParaGerarVariáveis(variável)
5      fim
6      retorna conjunto_de_variáveis
7  fim
```

Para demonstrar a conversão dos casos de testes foram utilizados dois problemas, o primeiro sendo o Conta, já apresentado anteriormente na Tabela 3.1, e o segundo sendo uma implementação de uma estrutura de dados chamada PrQuadTree. Os arquivos que compõem este último, bem como suas respectivas inclusões, estão apresentados na Tabela 6.1. As características dos problemas, como quantidade de instâncias geradas e quantidade de instâncias úteis podem ser visualizados na Tabela 6.2. Por fim, as características das instâncias de problemas úteis, relacionadas a cada um dos problemas são apresentadas na Tabela 6.3.

Assim como aconteceu em alguns estudos de caso do Capítulo 4, a primeira instância do problema Conta foi descartada, pois era muito simples e não tinha passeios suficientes para que fossem geradas as suítes de casos de teste. Deste modo, nesta seção serão executados os casos de teste das 5 instâncias de problema presentes na Tabela 6.3.

Tabela 6.1 - Características do problema `PRQuadTree`: arquivos e inclusões.

Arquivo	Inclusões
principal.cpp	prquadtree.hpp
prquadtree.hpp	no.hpp
prquadtree.cpp	prquadtree.hpp
no.hpp	ponto.hpp
no.cpp	no.hpp
ponto.hpp	sem inclusões
ponto.cpp	ponto.hpp

Tabela 6.2 - Características dos Problemas.

Problema	Instâncias geradas	Instâncias úteis	Nível de Integração
Conta	3	2	2
PrQuadTree	3	3	2
Total	6	5	

Com os problemas já definidos, é então feita a aplicação da ferramenta. Com isso, foi obtido o conjunto de suítes de teste que contém os casos de teste abstratos, relacionado a cada instância de problema. A partir deste momento, pode-se iniciar o processo de geração dos casos de teste executáveis, de fato.

Tabela 6.3 - Características das instâncias de problema uteis.

Instância de Problema	nome	Software	#Vértices	#Arestas	Passeios gerados
2	ACC2_2	Conta	16	19	8
3	ACC3_2	Conta	23	29	33
4	PQT1_2	PRQuadTree	26	32	8
6	PQT2_2	PRQuadTree	65	92	9897
7	PQT3_2	PRQuadTree	73	103	10.003

A Tabela 6.4 mostra a diversidade dos casos de teste presentes nas suítes de teste de cada algoritmo, que também consideram a instância de problema como um todo. Assim como nos problemas utilizados para realizar a avaliação experimental, as hiper-heurísticas também tiveram maior diversidade em relação às meta-heurísticas. A única diferença, foi a ordem dos algoritmos, que neste caso têm-se: HRISE_M, HRISE_R, HH-CF, SPEA-II, NSGA-III, IBEA e por fim MOMBI-II. Ou seja, nesse caso, IBEA e NSGA-III trocaram de posição.

Tabela 6.4 - Diversidade de casos de teste.

IP	IBEA	SPEA-II	NSGA-III	MOMBI-II	HRISE_M	HRISE_R	HH-CF	Total
ACC2_2	8	7	8	5	8	8	8	8
ACC3_2	24	24	19	14	27	27	26	31
PQT1_2	6	6	6	5I	7	7	8	8
PQT2_2	128	157	143	91	409	405	392	932
PQT3_2	89	110	88	77	386	355	325	793
média	51	61	53	47	167	160	157	354

É importante enfatizar que o processo pode ser inicializado de duas maneiras de acordo com a entrada passada, a primeira sendo: receber como entrada para a geração dos casos de testes abstratos um arquivo principal, ou seja, um arquivo que contenha a função *main*, que é uma função especial em programas C++; ou, receber um arquivo que contenha a representação de uma classe. Levando isso em consideração, podemos ter duas estruturas para realizar a conversão dos casos de testes abstratos para casos de testes executáveis.

Visando demonstrar estes dois tipos de estruturas fez-se o uso dos dois problemas citados acima, Conta e PRQuadTree, onde para o problema Conta será utilizado como arquivo principal, para iniciar o processo de integração, o arquivo que realmente é

o arquivo principal do projeto, ou seja, o arquivo que contém a função *main*, já para o segundo exemplo será utilizada uma classe, sendo ela a classe PRQuadTree, deixando de lado o arquivo principal deste problema para exemplificar os dois casos possíveis .

Essa distinção se fez necessária devido a forma que foi escolhida para elaborar os casos de teste executáveis. A principal diferença está na hora de comparar os valores obtidos com os valores esperados, uma vez que normalmente uma função *main* tem como retorno apenas o valor 0, que significa que o programa foi executado com sucesso, ou algum outro valor que significa que o programa foi executado com erros.

Para contornar esta característica na formulação do teste é gerado dentro do teste o fluxo que deveria ser seguido no *main*, assim é possível realizar as comparações dos valores esperados conforme os testes vão evoluindo. Assim como é feito quando o teste é realizado passando uma classe como parâmetro.

6.1 Conta

Para realizar a conversão dos casos de testes abstratos em casos de testes executáveis serão utilizadas as últimas suítes de teste de cada algoritmo na sua última execução. No caso do problema Conta as suítes de testes estão representadas na Tabela 6.5. Cada número presente na coluna Suíte de teste representa um caso de teste, por exemplo o caso de teste 1 na instância de problema ACC2_2 representa o seguinte passeio no grafo:

$$ct_1 = \{\_main\_2, \_usuario\_novou\_22\_3, \_conta\_novaConta\_4, \\ \_bool\_uvalido\_novou\_verificaidade\_5, \_if\_uvalido\_6, \\ \_return\_0\_10, final\}.$$

Já o caso de teste 3 representa o seguinte passeio no grafo:

$ct_3 = \{\_main\_2, \_user\_novoU\_22\_3, \_conta\_novaConta\_4,$
$\_bool\_uValido\_novoU\_verificaIdade\_5, \_if\_uvalido\_6,$
$\_novaConta\_inicializa\_novoU\_100\_7, \_inicializa\_22, \_user\_user\_23,$
$\_saldo\_s\_24, \_if\_saldo\_0\_25, \_erro\_na\_cria\_o\_da\_Conta\_25\_conta\_26,$
$\_novaConta\_inicializa\_novoU\_100\_7, \_novaConta\_deposita\_10\_8,$
$\_deposita\_29, \_saldo\_saldo\_valor\_30, \_novaConta\_deposita\_10\_8,$
$\_return\_0\_10, final\}$.

Na Tabela 6.5 pode-se observar uma grande repetição dos casos de teste, mas eliminando as repetições, para a instância de problema ACC2_2 foram selecionados os seguintes casos de teste, por meio dos algoritmos aplicados, 1, 2, 4, 5, 6 e 7, já para a instância de problema ACC3_2 foram selecionados os casos de teste 5, 7, 8, 10, 12, 13, 17, 19, 21, 22 e 27. Dentre os estudos de caso escolhidos nesta seção apenas a instância de problema ACC3_2 não apresentou os casos de testes 1 e 2 nas suítes de casos de testes escolhidas. É importante ressaltar que alguns dos passeios gerados não condizem com o fluxo possível de acordo com as variáveis passadas, ou seja são passeios não executáveis, por exemplo o caso de teste 8 da instância de problema ACC3_2, este casos de teste consiste no seguinte passeio:

$ct_8 = \{main\_35, usuario\_novou\_22\_36, conta\_novaconta\_37,$
$bool\_uvalido\_novou\_verificaidade\_38, verificaidade\_74, if\_75,$
$else\_78, return\_79, bool\_uvalido\_novou\_verificaidade\_38,$
$if\_39, novaconta\_inicializa\_novou\_100\_40, inicializa\_54,$
$usuario\_usuario\_55, saldo\_s\_56, if\_57, erro\_na\_cria\_o\_da\_conta\_58,$
$novaconta\_inicializa\_novou\_100\_40, novaconta\_deposita\_10\_41,$
$deposita\_61, saldo\_saldo\_valor\_62, novaconta\_deposita\_10\_41,$
$return\_43, final\}$.

Mas uma vez que o passeio "*{verificaidade_74 -> if_75 -> else_78 -> return_79}*" é feito, não é possível realizar o passeio "*{if_39 -> novaconta_inicializa_novou_100_40 -> inicializa_54 -> usuario_usuario_55 -> saldo_s_56 -> if_57 -> erro_na_cria_o_da_conta_58 -> novaconta_inicializa_novou_100_40 -> nova-*

conta_deposita_10_41 -> deposita_61 -> saldo_saldo_valor_62 -> novaconta_deposita_10_41 -> return_43 }". Isso acontece porque os passeios são gerados com base no grafo, que contém todos os passeios possíveis, mas não há nenhum indicador de fluxos dependentes, então mesmo percorrendo um passeio que não possibilita um outro trajeto, este trajeto é adicionado pois ele está contido no grafo. O mesmo vale para o passeio 17 representada por:

$$ct_1 7 = \{main\_35, usuario\_novou\_22\_36, conta\_novaconta\_37, bool\_uvalido\_novou\_verificaidade\_38, verificaidade\_74, if\_75, return\_76, bool\_uvalido\_novou\_verificaidade\_38, if\_39, return\_43, final\}.$$

Uma vez que é realizado o passeio "*{verificaidade_74, if_75, return_76, bool_uvalido_novou_verificaidade_38}*", não é possível realizar o "*{if_39, return_43, final}*".

Tabela 6.5 - Parte da suíte de teste do problema Conta.

Instância de Problema	Algoritmo	Suite de Teste
ACC2_2	IBEA	1 1 6 1 1 1 1 1 1 1
	SPEA2	1 1 1 1 1 4 1 1 1 1
	NSGA-III	1 1 1 1 1 1 1 1 1 1
	MOMBI-II	7 1 6 1 7 1 7 6 1 1
	HH-CF	6 7 6 5 7 2 1 1 1 1
	HRISE-M	1 1 1 1 1 5 1 1 1 1
	HRISE-R	1 7 7 7 7 7 5 1 1 7
ACC3_2	IBEA	22 13 7 7 7 12 7 7 7 7
	SPEA2	7 7 7 7 7 7 8 7 7 7
	NSGA-III	7 7 10 7 7 7 7 7 7 7
	MOMBI-II	7 7 7 10 7 7 7 22 7 7
	HH-CF	7 7 7 22 7 22 22 22 7 5
	HRISE-M	7 7 7 7 19 7 27 7 17 7
	HRISE-R	22 7 7 7 7 7 22 7 12 21

6.1.1 Casos de teste executáveis

Como nesta situação, o caso de teste é referente a um arquivo contendo a função *main*, foi refeito o processo que ele deveria realizar, e utilizado o método *AreEquals*

para verificar se o valor das variáveis corresponde com o esperado. A Figura 6.1 representa o passeio 1 da instância de problema ACC2_2, o passeio completo pode ser visualizada no Apêndice B.1.1.1. Pode-se observar que, de acordo com a regra de negócio estipulada, menores de idade não podem inicializar uma nova conta, e o caso de teste 1 representa este passeio.

Figura 6.1 - Teste relativo ao caso de teste 1 da instância de problema ACC2_2 .

```
1  TEST_METHOD(Conta_2_2_trilha_1)
2  {
3          Usuario novoU(12);
4          Assert::AreEqual(12, novoU.idade_);
5
6          Conta novaConta;
7          Assert::AreEqual(float(NULL), novaConta.saldo_);
8
9          Assert::AreEqual(false, novoU.verificaIdade());
10
11 }
```

Fonte: Produção do autor.

A Figura 6.2 representa o passeio 2 também da instância de problema ACC2_2, o passeio pode ser visualizado no Apêndice B.1.1.2. Esse caso de teste percorre a o grafo quase que completamente, deixando de cobrir apenas 5 das arestas existentes.

Figura 6.2 - Teste relativo ao caso de teste 2 da instância de problema ACC2_2 .

```
1  TEST_METHOD(Conta_2_2_trilha_2)
2  {
3          Usuario novoU(20);
4          Assert::AreEqual(20, novoU.idade_);
5
6          Conta novaConta;
7          Assert::AreEqual(float(NULL), novaConta.saldo_);
8
9          Assert::AreEqual(true, novoU.verificaIdade());
10
11         newAccount.initialize(novoU, 1000);
12         Assert::AreEqual(float(1000.0), novaConta.balance_);
13
14         newAccount.deposit(200);
15         Assert::AreEqual(float(1200.0), novaConta.balance_);
16
17 }
```

Fonte: Produção do autor.

Já para a instância de problema ACC3_2 foram selecionados os passeios 22 e 13. O passeio 22 é semelhante ao passeio 1 da primeira instância de problema, e, assim

como ele, com as variáveis passadas ela retorna o esperado, a diferença entre os dois é que no passeio 22 é adicionado o trajeto que deve ser percorrido relacionado a classe usuário (usuario_71, idade_idade_72), é importante ressaltar que mesmo que novas arestas e vértices tenham sidos adicionadas, isso não impede que casos de teste iguais ao da instância anterior sejam gerados, pois aquele passeio continua existindo. O caso de teste executável pode ser visualizado na Figura 6.3 e o passeio esta presente no Apêndice B.1.2.1:

Figura 6.3 - Teste relativo ao caso de teste 22 da instância de problema ACC3_2 .

```
1  TEST_METHOD(Conta_3_2_trilha_22)
2  {
3          Usuario novoU(12);
4          Assert::AreEqual(12, novoU.idade_);
5
6          Conta novaConta;
7          Assert::AreEqual(float(NULL), novaConta.saldo_);
8
9          Assert::AreEqual(false, novoU.verificaIdade());
10
11 }
```

Fonte: Produção do autor.

Já para o passeio 13 disponibilizado no Apêndice B.1.2.2 pode-se perceber que a mensagem de erro é exibida, mas não ocorreu erro de fato, e a conta foi inicializada com valor negativo, desta forma é visto que é necessário refatorar o código-fonte inicial, uma vez que ele contém um erro na implementação, que permite a inicialização da conta com valor negativo, o que não condiz com a mensagem exibida. O caso de teste pode ser visto na Figura 6.4.

6.2 PrQuadTree

Da mesma forma que no problema anterior, neste também serão utilizadas as ultimas suítes de teste de cada algoritmo na sua ultima execução. No caso do problema PrquadTree as suítes de testes estão representadas na Tabela 6.6. Cada número presente na coluna “Suite de teste” representa uma caso de teste, por exemplo o caso de teste 5, da instancia de problema PQT1_2, representa o seguinte passeio no grafo:

Figura 6.4 - Teste relativo ao caso de teste 13 da instância de problema ACC3_2 .

```
TEST_METHOD(Conta_3_2_trilha_13)
{
        Usuario novoU(20);
        Assert::AreEqual(20, novoU.idade_);

        Conta novaConta;
        Assert::AreEqual(float(NULL), novaConta.saldo_);

        Assert::AreEqual(true, novoU.verificaIdade());

        novoConta.initialize(novoU, -100);
        Assert::AreEqual(float(NULL), novaConta.saldo_);

        novoConta.deposit(200);
        Assert::AreEqual(float(200.0), novaConta.saldo_);
}
```

Fonte: Produção do autor.

$$
\begin{aligned}
ct_5 = \{&\_prquadtree\_1, \_maxdepth\_2, \_root\_3, \_insert\_8,\\
&\_node\_nodetemp\_getleafat\_point\_getx\_point\_gety\_9,\\
&\_nodetemp\_addvalue\_point\_10, \_if\_nodetemp\_getdepth\_maxdepth\_11,\\
&\_list\_node\_list\_14, \_list\_push\_back\_nodetemp\_15, \_while\_16,\\
&\_nodetemp\_list\_back\_17, \_list\_pop\_back\_18,\\
&\_if\_nodetemp\_getlen\_1\_nodetemp\_getdepth\_maxdepth\_19,\\
&\_nodetemp\_subdivide\_20, \_list\_push\_back\_nodetemp\_getchild\_e\_22,\\
&\_getleafat\_48, \_node\_nodetemp\_root\_49, \_while\_50,\\
&\_return\_nodetemp\_58, final\}.
\end{aligned}
$$

Para cada instância de problema foram selecionadas os casos de testes sem repetição, resultando em 7 casos de teste para a instância 1, sendo os casos de teste números 0, 1, 2, 5, 6, 7 e 8. Para a instância de problema 2 foram obtidos 39 casos de testes, sendo os seguintes casos de teste 1, 2, 6, 126, 986, 1133, 1138, 1171, 1200, 1268, 1296, 1400, 1574, 1679, 4779, 4917, 4919, 5094, 5139, 5176, 5277, 5350, 5484, 5571, 5612, 5613, 5623, 5624, 5718, 6673, 8158, 8786, 9077, 9140, 9218, 9295, 9800, 9872 e 9877. Por fim, para a instância de problema 3 foram obtidos também 39 casos de teste distintos, sendo, 1, 2, 80, 122, 1133, 1172, 1250, 1256, 1296, 1575, 1679, 1828, 1994, 4917, 4919, 5094, 5176, 5267, 5350, 5380, 5415, 5484, 5569, 5583, 5607, 5613, 5624, 5626, 5718, 6434, 8416, 9141, 9218, 9246, 9757, 9761, 9835, 9851 e 9877.

Pode-se perceber que na primeira instância, PQT1_2, é grande a repetição do mesmo estudo de caso, pode-se observar que o caso de teste 7 está presente na suíte de teste de todos os algoritmos, aparecendo no mínimo duas vezes em cada para cada algoritmo. Pode-se observar que conforme a complexidade do estudo de caso vai aumentando, essa repetição fica cada vez menor, principalmente na mesma suíte de teste.

Tabela 6.6 - Parte da suíte de teste do problema PrQuadTree.

Instância de Problema	Algoritmo	Suíte de Teste
PQT1_2	IBEA	5 8 8 7 2 7 7 7 2 2
	SPEA2	7 2 7 8 7 7 7 8 8 7
	NSGA-III	7 7 7 7 7 0 7 7 8 8
	MOMBI-II	2 6 8 7 7 8 2 8 7 7
	HH-CF	8 2 2 2 8 2 5 7 8 7
	HRISE-M	2 8 7 7 8 2 8 7 7 5
	HRISE-R	8 2 8 7 2 7 7 8 1 8
PQT2_2	IBEA	5350 5484 1296 5718 5484 1296 2 9140 5718 5350
	SPEA2	5612 5484 4917 5484 5718 5484 1400 5350 6 5350
	NSGA-III	5718 4917 5350 5484 5718 1296 5484 5484 9877 126
	MOMBI-II	8786 1679 5612 5484 5571 1133 9800 1200 4779 1296
	HH-CF	5613 6673 5094 9877 9218 9877 1268 1171 2 1574
	HRISE-M	5623 5624 9077 9872 4917 986 4919 5571 2 5484
	HRISE-R	5277 5139 8158 1 9877 1138 9295 5718 5176 5571
PQT3_2	IBEA	5484 122 5484 5484 1133 9835 1256 9218 4917 5484
	SPEA2	1172 2 5094 5484 4919 1256 9757 1296 5484 9835
	NSGA-III	5718 5350 1296 1172 5613 9877 5484 5613 4917 5484
	MOMBI-II	5624 5607 5613 5718 5613 5415 5350 1256 1172 5569
	HH-CF	1575 1994 5718 1172 5380 5626 2 80 9851 5176
	HRISE-M	9141 9877 5267 8416 1250 1 1828 5484 5484 5484
	HRISE-R	5583 6434 9246 5350 80 1679 4917 5484 1256 9761

6.2.1 Casos de teste executáveis

No contexto abordado neste problema, o caso de teste é referente a um arquivo que contém a implementação da classe, então neste caso são realizadas as chamadas do métodos presentes na classe de acordo com o que está presente no passeio que representa o caso de teste. Foram selecionados os casos de teste 1 e 2, pois aparecem em todas as instâncias de problema. Assim como no problema da Conta também foi utilizado o *AreEquals* para verificar se o valor das variáveis corresponde com o

esperado. A imagem do caso de teste executável que o representa pode ser visualizada na Figura 6.5 e o passeio do caso de teste 1, da instância de problema PQT1_2, pode ser vista no Apêndice B.2.1.1. Neste caso de teste foram executada apenas as funções da classe PRQuadTree, utilizando *Assert* apenas para realizar a verificação do retorno do método de acordo com o passeio estipulado. O ponto b(linha 15) foi inserido na árvore contida no teste(linha 16) apenas para seguir o processo estipulado para a função *getleafat*, que necessitava de dois pontos na árvore.

Figura 6.5 - Teste relativo ao caso de teste 1 da instância de problema PQT1_2 .

```
1  TEST_METHOD(PrQuadtree_1_2_trilha_1)
2  {
3          //construtor
4          PRquadtree prquadtree(0.0, 100.0, 0.0, 100.0, 10);
5
6          Assert::AreEqual(10, prquadtree.maxDepth);
7
8          //insert
9          Point a(25.0, 40.0, 10);
10         prquadtree.insert(&a);
11         Assert::AreEqual(10, prquadtree.root->getVal()[0]->getValue());
12
13         //getleafat
14
15         Point b(60.0, 40.0, 20);
16         prquadtree.insert(&b);
17         Assert::AreEqual(a.getValue(), prquadtree.getLeafAt(25.0, 45.0)->getVal()[0]->
               getValue());
18 }
```

Fonte: Produção do autor.

Já para o caso de teste 2 da mesma instância de problema, PQT1_2, a imagem do caso de teste executável que o representa pode ser visualizada na Figura 6.6 e o passeio pode ser vista no Apêndice B.2.1.2. Neste caso, pode-se observar que tem 2 chamadas as função *insert*, isso ocorre devido a necessidade de ter dois nós na árvore para que seja realizado o processo descrito no passeio, então o trajeto do passeio esta presente na linha 12, os outros processos seguem como no passeio anterior.

Já para a instância de problema PQT2_2, o caso de teste pode-ser visualizado na Figura 6.7 e o passeio 1 sendo representado no Apêndice B.2.2.1. Neste caso de teste já se tem a interação com funções de outras classes, o processo dos métodos em teste continuam seguindo o descrito nos passeios e além disso são realizadas verificações adicionais nos métodos que são chamados para verificar se estão de fato funcionando conforme esperado. Nesta situação o método *getDepth*, pertencente a classe Node, foi utilizado na linha 11, e o método *addvalue*, também da classe node, que está presente no passeio realizado pelo *insert*.

Figura 6.6 - Teste relativo ao caso de teste 2 da instância de problema PQT1_2.

```
TEST_METHOD(PrQuadtree_1_2_trilha_2)
{
        //construtor
        PRquadtree prquadtree(0.0, 100.0, 0.0, 100.0, 0);

        Assert::AreEqual(0, prquadtree.maxDepth);

        //insert
        Point a(25.0, 40.0, 10);
        Point b(60.0, 40.0, 20);
        prquadtree.insert(&a);
        prquadtree.insert(&b);
        Assert::AreEqual(a.getValue(), prquadtree.root->getVal()[0]->getValue());
        Assert::AreEqual(b.getValue(), prquadtree.root->getVal()[1]->getValue());

        //getleafat
        Assert::AreEqual(a.getValue(), prquadtree.getLeafAt(10, 45.0)->getVal()[0]->
            getValue());
}
```

Fonte: Produção do autor.

Pode-se observar que não foi realizado o teste da funcionalidade *getleatat* neste caso de teste, isso ocorre pelo fato do vértice *__getdepth_156* ser chamado por mais de uma função, então quando ao invés de realizar o passeio que retorna o valor para a origem da chamada , o vértice *__if_nodetemp_getdepth_maxdepth_11*, ele segue o passeio para retornar para o vértice *__if_nodetemp_getlen_1_nodetemp_getdepth_maxdepth_19*, que não é sequência do vértice, de chamada inicial (*__if_nodetemp_getdepth_maxdepth_11*), por este motivo não há passeio a ser realizado na função *getleatat*, já que o fluxo do grafo foi cortado.

Figura 6.7 - Teste relativo ao caso de teste 1 da instância de problema PQT2_2 .

```
TEST_METHOD(PrQuadtree_2_2_trilha_1)
{
        //construtor
        PRquadtree prquadtree(0.0, 100.0, 0.0, 100.0, 3);
        Assert::AreEqual(3, prquadtree.maxDepth);

        //insert
        Point a(25.0, 40.0, 10);
        prquadtree.insert(&a);
        Assert::AreEqual(a.getValue(), prquadtree.root->getVal()[0]->getValue());
        Assert::AreEqual(float(0), prquadtree.root->getDepth());

        //getleafat

}
```

Fonte: Produção do autor.

Já para o passeio 2 a imagem pode ser vista na Figura 6.8 e o passeio 2 pode

ser visualizado no Apêndice B.2.2.2. Neste caso de teste também foi necessária a utilização de mais de um ponto na árvore, mas desta vez o motivo é diferente, neste caso a necessidade é pelo falto da descrição do processo passar pelo método *subdivide*, e para realizar se foi de fato feito esse passeio é checado na linha 21. Em sequência é feito o processo descrito para a função *getleatat* onde é feita a verificação se está retornando o ponto esperado.

Figura 6.8 - Teste relativo ao caso de teste 2 da instância de problema PQT2_2.

```
1  TEST_METHOD(PrQuadtree_2_2_trilha_2)
2  {
3          //construtor
4          PRquadtree prquadtree(0.0, 100.0, 0.0, 100.0, 3);
5          Assert::AreEqual(3, prquadtree.maxDepth);
6
7
8          //insert
9          Point a(25.0, 40.0, 10);
10
11         prquadtree.insert(&a);
12         Assert::AreEqual(10, prquadtree.root->getVal()[0]->getValue());
13
14         Assert::AreEqual(float(0), prquadtree.root->getDepth());
15
16         Assert::AreEqual(1, prquadtree.root->getLen());
17         Point b(25.0, 80.0, 20);
18
19         prquadtree.insert(&b);
20
21         Assert::AreEqual(b.getValue(), prquadtree.root->getChild(1)->getVal()[0]->
               getValue());
22
23         //getleafat
24         Assert::AreEqual(a.getValue(), prquadtree.getLeafAt(20.0, 45.0)->getVal()[0]->
               getValue());
25         Assert::AreEqual(float(0), prquadtree.root->getDepth());
26
27
28 }
```

Fonte: Produção do autor.

Para a instância de problema PQT3_2, o caso de teste 1 pode ser visualizado pela Figura 6.9 e o passeio corresponde está presente no Apêndice B.2.3.1. Pode-se observar a integração dos métodos pertencentes à classe *Point* no passeio gerado, *getY* e *getX*, para testar o retorno destes métodos foram adicionadas as linhas 10 e 11(Figura 6.9). Além disso também pode-se observar a presença de métodos pertencentes a classe *Node*, como por exemplo *__addvalue__17* e *__getdepth__156*. Neste caso de teste, ocorreu o mesmo problema da instancia PQT2_1, o fluxo também se perdeu no retorno para o vértice que realizou a chamada, mas nesse caso ocorreu na função *__getchild__159*, que retornou para o vértice *__nodetemp__nodetemp__getchild__e__53* ao invés de retornar para *__list__push__back__nodetemp__getchild__e__22*, sendo assim

neste passeio também não tem a chamada ao método *getleatat*.

Figura 6.9 - Teste relativo ao caso de teste 1 da instância de problema PQT3_2 .

```
1  TEST_METHOD(PrQuadtree_3_2_trilha_1)
2  {
3          //construtor
4          PRquadtree prquadtree(0.0, 100.0, 0.0, 100.0, 3);
5          Assert::AreEqual(3, prquadtree.maxDepth);
6
7          //insert
8          Point a(25.0, 40.0, 10);
9          prquadtree.insert(&a);
10         Assert::AreEqual(a.getY(), prquadtree.root->getVal()[0]->getY());
11         Assert::AreEqual(a.getX(), prquadtree.root->getVal()[0]->getX());
12         Assert::AreEqual(float(0.0), prquadtree.root->getDepth());
13         Assert::AreEqual(10, prquadtree.root->getVal()[0]->getValue());
14
15         //getleafat
16 }
```

Fonte: Produção do autor.

A Figura 6.10 representa o caso de teste do passeio 2 da última instância de problema, sendo o passeio apresentado no Apêndice B.2.3.2. Neste caso de teste pode-se observar que apesar da classe *Point* ter sido adicionada, os métodos pertencentes a ela, por exemplo *getX* e *getY*, não estão presente neste passeio. Neste caso de teste o processo descrito no método *insert* está sendo realizado pela segunda chamada ao método, o que realiza a a inserção do ponto b. A linha 15 representa a verificação da execução da passeio presente no método subdivide e a linha 16 o teste do retorno do método *getLen*, ambos métodos da classe *Node* existentes no caso de teste. Por fim é executado o método *getleafat*, assim como nos anteriores que continham o método.

6.3 Considerações finais sobre este capítulo

Neste capítulo foi apresentado uma maneira de superar a limitação da ferramenta de não realizar a geração automática do caso de teste executável. Para isso foi utilizados alguns exemplos simples para que fosse possível visualizar o processo. Selecionou-se dois problemas para serem testados que se enquadrassem nas duas formas possíveis de realização de casos de teste, sendo, a primeira utilizando um arquivo com a função *main* como principal e a segunda maneira utilizando um arquivo que represente uma classe. Pode se observar que passeios gerados são de grande valia para serem utilizadas como guias na criação de casos de teste, mesmo que em alguns casos o passeio gerado acabe se perdendo no retorno para a função correta, ou as vezes gerando passeios que não devem ser executados dado que um outro foi.

Figura 6.10 - Teste relativo ao caso de teste 2 da instância de problema PQT3_2.

```
TEST_METHOD(PrQuadtree_3_2_trilha_2)
{
        //construtor
        PRquadtree prquadtree(0.0, 100.0, 0.0, 100.0, 3);
        Assert::AreEqual(3, prquadtree.maxDepth);

        //insert
        Point a(25.0, 40.0, 10);
        prquadtree.insert(&a);
        Assert::AreEqual(10, prquadtree.root->getVal()[0]->getValue());


        Point b(25.0, 80.0, 20);
        prquadtree.insert(&b);
        Assert::AreEqual(float(0.0), prquadtree.root->getDepth());
        Assert::AreEqual(1, prquadtree.root->getChild(1)->getLen());
        Assert::AreEqual(b.getValue(), prquadtree.root->getChild(1)->getVal()[0]->
            getValue());

        //getleafat
        Assert::AreEqual(a.getValue(), prquadtree.getLeafAt(25.0, 45.0)->getVal()[0]->
            getValue());
}
```

Fonte: Produção do autor.

7 CONCLUSÃO

Dada a importância de Teste de software para obter produtos com alta qualidade, essa dissertação de mestrado contribuiu para a atividade de Teste de integração de software via algoritmos de otimização, nesse caso meta-heurísticas e hiper-heurísticas. Duas hipóteses foram formuladas, sendo uma relacionada a viabilidade das meta-heurísticas e hiper-heurísticas para gerar os casos de Teste de integração, a outra em termos de quais algoritmos teriam um melhor desempenho sob a ótica da comunidade de otimização, nesse caso conjecturando que as hiper-heurísticas teriam melhor desempenho do que as meta-heurísticas.

Um método, denominado InMeHy, foi concebido para alcançar o objetivo de viabilidade, assim como ferramentas foram implementadas para apoiar o método. Esse método pressupõe apenas a existência do código-fonte de aplicações desenvolvidas em C++ para gerar casos de Teste de integração. Essa característica é importante pois gera-se casos de teste baseando-se no artefato mais comum de um produto de software, ou seja, o seu código-fonte.

Duas estratégias, STV e STF, foram derivadas para gerar casos de teste, onde tais estratégias definem de forma diferente os conceitos de casos de teste abstratos, soluções das populações e variáveis de decisão. Na STV, cada solução é uma suíte de teste e a mesma pode ter um tamanho variável de casos de teste. Na STF, cada solução é uma suíte de teste e a mesma tem um tamanho fixo e predefinido de casos de teste. A primeira hipótese foi aceita, demonstrando a viabilidade dos algoritmos de otimização para gerar casos de Teste de integração a partir de código-fonte em C++.

Dois experimentos controlados foram realizados para abordar a hipótese sobre desempenho, que no fundo trata de analisar a capacidade de generalização dos algoritmos. Ambos os experimentos usaram como estudos de caso dois produtos não triviais, GeoDMA e TerraLib. No caso STF, 12 problemas e 39 instâncias de problemas foram avaliados.

O primeiro experimento foi realizado com a estratégia STV e as meta-heurísticas clássicas IBEA e SPEA2, assim como as meta-heurísticas para inúmeros objetivos, NSGA-III e MOMBI-II. Na segunda experimentação, além das quatro meta-heurísticas anteriores, as hiper-heurísticas de seleção HRISE_R, HRISE_M e HH-CF foram as escolhidas e, nesse caso, a estratégia foi a STF.

No primeiro experimento, o IBEA foi o melhor algoritmo, seguido pelo NSGA-III, que foi um pouco melhor do que o SPEA2. Já no segundo experimento, onde foram utilizados todos os algoritmos incluindo as hiper-heurísticas, o SPEA2 foi o algoritmo que apresentou o melhor resultado seguido pelo IBEA, as duas meta-heurísticas clássicas propostas há algum tempo. A meta-heurística NSGA-III foi considerada a terceira e, entre as hiper-heurísticas, HRISE_R foi considerada a melhor, com um desempenho um pouco superior à HRISE_M.

Portanto, a hipótese sobre desempenho foi rejeitada no contexto dos experimentos realizados onde, considerando os algoritmos e estudos de caso selecionados, pode-se afirmar que as meta-heurísticas clássicas (mais antigas) tiveram um melhor desempenho não somente comparadas às hiper-heurísticas de seleção mas, também, em relação às meta-heurísticas para inúmeros objetivos.

Não é possível generalizar as conclusões sobre a resposta ao problema da generalização dos algoritmos de otimização. Em outras palavras, não se pode afirmar que meta-heurísticas multiobjetivo (clássicas) são realmente melhores do que as hiper-heurísticas de seleção e do que as meta-heurísticas para inúmeros objetivos, para qualquer tipo de problema. Além disso, essa conclusão, baseada em experimentações, parece estar relacionada aos teoremas sem almoço grátis. Portanto, é interessante verificar experimentalmente se as teorias propostas são válidas, e parece ser o caso aqui com base nos resultados que foram obtidos.

Dessa forma, é importante que novos experimentos sejam realizados para tentar obter uma resposta em termos de generalização na prática.

7.1 Contribuições e limitações

As contribuições desse trabalho estão apoiadas em dois pontos principais. Primeiramente, a necessidade de desenvolver métodos de Teste de integração baseados em algoritmos de otimização. Segundo, a necessidade de realizar uma experimentação rigorosa para verificar o desempenho das meta-heurísticas e hiper-heurísticas.

Neste contexto, pode-se destacar, como primeira contribuição desse trabalho, o desenvolvimento de um método com duas estratégias para geração de casos de Teste de integração, assim como ferramentas que os apoiam. As ferramentas são capazes de, a partir apenas do código-fonte escrito na linguagem C++, gerar um grafo direcionado que representa a integração de arquivos do sistema. Por meio desse grafo, as ferramentas realizam a geração de casos de teste para o nível de integração, via

meta-heurísticas e hiper-heurísticas. Note-se que o grande ganho que o método InMeHy e suas ferramentas propiciam é o fato de gerar os casos de teste abstratos automaticamente, onde estes servem de guia para gerar os casos de teste executáveis. As ferramentas estão disponíveis livremente para uso conforme descrito no Capítulo 3. A segunda contribuição é a experimentação rigorosa, considerando até sete algoritmos de otimização e dois estudos de caso não triviais.

As ferramentas desenvolvidas geram, de fato, casos de teste abstratos e, dessa forma, se faz necessário convertê-los em casos de teste executáveis. Uma limitação relacionada as ferramentas está ligada ao seu tipo de entrada, uma vez que para tornar a conversão possível, as versões atuais das ferramentas definem algumas restrições no processo de geração do grafo de maneira automática. As restrições para realizar a conversão do código-fonte para grafo foram aplicadas nos arquivos que compõem o SST. Sendo assim, o SST precisa estar de acordo com algumas normas:

- As estruturas de laço, por exemplo "for" e "while", devem conter os limitadores de escopo, ou seja, tem que conter o caractere "{" para abrir o corpo do laço e o caractere "}" para delimitar o final do laço;
- A estrutura condicional "if" deve seguir a mesma regra acima, ou seja, deve conter os limitadores de escopo ("{", "}");
- Cada método nas classes deve ter um nome único, mesmo que tenha assinaturas diferentes;
- Operadores só são considerados no escopo da classe, eles não são contemplados no processo de integração.

Outra limitação das ferramentas é não levar em conta o valor das variáveis para realizar a geração dos casos de teste, o que pode resultar em passeios não executáveis, de acordo com o valor das variáveis. Por exemplo, caso exista uma ramificação no código que só acontece caso uma variável seja falsa, o passeio gerado pode conter essa ramificação, ainda que anteriormente esta variável esteja como verdadeira. Isso acontece devido ao fato de que, no grafo, este é um passeio possível, visto que as duas ramificações são consideradas na formação do mesmo.

O desenvolvimento do método e das ferramentas foram guiados para gerar casos de teste para duas aplicações não triviais do INPE, e que já estão "completas" e em funcionamento. Note que mesmo assim, o Teste de integração pode ser útil para

detectar defeitos críticos e difíceis de serem encontrados nas interfaces quando da montagem final do sistema. Uma outra limitação, portanto, é que as ferramentas ainda não geram os casos de teste executáveis de forma automática. Isso se deve, principalmente, pela complexidade dos parâmetros que podem vir a ser utilizados nas aplicações.

Além disso, mesmo que houvesse uma tradução manual dos casos de teste abstratos gerados automaticamente pelas ferramentas que apoiam o método InMeHy, para os casos de teste executáveis, não foi possível executar os casos de teste. Isso ocorre, justamente, devido ao acoplamento das classes das aplicações usadas como estudo de caso que, além de estarem prontas e integradas, são não triviais. No entanto, o Capítulo 6 demonstra como é possível traduzir os casos de teste abstratos gerados automaticamente e estimular os SSTs com os casos de teste executáveis. Para isso, foram considerados estudos de caso mais simples e utilizado o *framework* cppUnit.

Conforme mencionado anteriormente, os casos de testes abstratos gerados pelas ferramentas servem como guias para o testador na criação do caso de teste realmente executável. Desse modo, o testador não precisa se preocupar em elaborar a o caminho que o caso de teste deve percorrer, tendo apenas a responsabilidade de transcrever o passeio gerado para o caso de teste executável.

7.2 Trabalhos futuros

Diante do contexto apresentado e levando em consideração as limitações do método proposto e das ferramentas que o apoiam, a seguir destaca-se algumas atividades futuras para essa pesquisa:

- Alterar o processo de extração de informações do código-fonte e geração do grafo nas ferramentas, de maneira que as variáveis possam ser consideradas, desta forma evitando passeios válidos no grafo mas não executáveis no contexto do passeio;
- Automatizar o processo de conversão de casos de teste abstratos para executáveis, efetuando as modificações necessárias nas ferramentas;
- Realizar experimentos controlados com a execução dos casos de teste derivados pelos algoritmos de otimização e análise de efetividade de acordo com ótica da comunidade de Teste de software (e.g. via análise de mutantes);
- Realizar a adição de outros algoritmos para inúmeros objetivos e hiper-

heurísticas de seleção com a finalidade de verificar se as meta heurísticas clássicas continuam a ser as melhores, nos estudos de casos abordados;

- Considerar outros estudos de caso desenvolvidos em C++ e não somente SIGs;
- Investigar as razões para explicar porque os algoritmos mais recentes, hiper-heurísticas de seleção e meta-heurísticas para inúmeros objetivos, não tiveram o desempenho esperado;
- Fazer uso de ferramentas de referência, e.g. Evosuite, adaptando-as para o contexto de Teste de integração e comparar com as ferramentas que apoiam o método InMeHy.

REFERÊNCIAS BIBLIOGRÁFICAS

AFZAL, W.; TORKAR, R.; FELDT, R. A systematic review of search-based testing for non-functional system properties. **Information and Software Technology**, v. 51, p. 957–976, 06 2009. 10

ALMEIDA, C. P.; GONÇALVES, R. A.; VENSKE, S.; LüDERS, R.; DELGADO, M. Hyper-heuristics using multi-armed bandit models for multi-objective optimization. **Applied Soft Computing**, v. 95, p. 106520, 2020. ISSN 1568-4946. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1568494620304592>. 3

ALSEWARI, A. R. A.; ZAMLI, K. Z. Design and implementation of a harmony-search-based variable-strength t-way testing strategy with constraints support. **Information and Software Technology**, v. 54, n. 6, p. 553 – 568, 2012. ISSN 0950-5849. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0950584912000134>. 2

ASOUDEH, N.; LABICHE, Y. Multi-objective construction of an entire adequate test suite for an efsm. In: INTERNATIONAL SYMPOSIUM ON SOFTWARE RELIABILITY ENGINEERING, 25., 2014. **Proceedings...** Naples: IEEE, 2014. p. 288–299. ISSN 1071-9458. 2

ASSUNÇÃO, W. K. G. **Uma abordagem para integração e teste de módulos baseada em agrupamento e algoritmos de otimização multiobjetivos**. Dissertação (Mestrado em Informática) — Universidade Federal do Paraná, Curitiba, 2012. 11

ASSUNÇÃO, W. K. G.; COLANZI, T. E.; VERGILIO, S. R.; POZO, A. T. R. Estabelecendo sequências de teste de integração de classes: um estudo comparativo da aplicação de três algoritmos evolutivos multiobjetivos. In: SIMPÓSIO BRASILEIRA DE REDES DE COMPUTADORES E SISTEMAS DISTRUBUÍDOS, 29., 2011. **Anais...** Campo Grande: SBC, 2011. 16, 19

BALERA, J. M.; SANTIAGO JÚNIOR, V. A. A systematic mapping addressing hyper-heuristics within search-based software testing. **Information and Software Technology**, v. 114, p. 176 – 189, 2019. ISSN 0950-5849. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0950584919301430>. 1, 2, 3, 14

BANSAL, P.; SABHARWAL, S.; SIDHU, P. An investigation of strategies for finding test order during integration testing of object oriented applications. In: INTERNATIONAL CONFERENCE ON METHODS AND MODELS IN COMPUTER SCIENCE, 2009. **Proceedings...** New Delhi: IEEE, 2010. p. 1 – 8. 18

BRIAND, L.; LABICHE, Y.; LIU, Y. Combining uml sequence and state machine diagrams for data-flow based integration testing. In: MODELLING FOUNDATIONS AND APPLICATIONS. **Proceedings...** Berlin: Springer, 2012. p. 74–89. 18, 19

BRITISH BROADCASTING CORPORATION. **Boeing 737 Max: what went wrong?** abr 2019. Disponível em: <https://www.bbc.com/news/world-africa-47553174>. Acesso em: ago. 11, 2021. 1

BURKE, E. K.; GENDREAU, M.; HYDE, M.; KENDALL, G.; OCHOA, G.; ÖZCAN, E.; QU, R. Hyper-heuristics: a survey of the state of the art. **Journal of the Operational Research Society**, v. 64, n. 12, p. 1695–1724, Dec 2013. ISSN 1476-9360. Disponível em: <https://doi.org/10.1057/jors.2013.71>. 2, 13

BURKE, E. K.; HYDE, M. R.; KENDALL, G.; OCHOA, G.; ÖZCAN, E.; WOODWARD, J. R. A classification of hyper-heuristic approaches: revisited. In: _____. **Handbook of metaheuristics**. Cham: Springer, 2019. p. 453–477. ISBN 978-3-319-91086-4. Disponível em: <https://doi.org/10.1007/978-3-319-91086-4_14>. 2

CÂMARA, G.; VINHAS, L.; FERREIRA, K. R.; QUEIROZ, G. R. D.; SOUZA, R. C. M. D.; MONTEIRO, A. M. V.; CARVALHO, M. T. D.; CASANOVA, M. A.; FREITAS, U. M. D. Terralib: an open source gis library for large-scale environmental and socio-economic applications. In: _____. **Open source approaches in spatial data handling**. Berlin, Heidelberg: Springer, 2008. p. 247–270. ISBN 978-3-540-74831-1. Disponível em: <https://doi.org/10.1007/978-3-540-74831-1_12>. 1, 4, 6, 54

CIGNITI. **37 Epic software failures that mandate the need for adequate software testing**. 2021. Disponível em: <https://www.cigniti.com/blog/37-software-failures-inadequate-software-testing/>. Acesso em: ago. 11, 2021. 1

COLANZI, T.; ASSUNÇÂO, W.; VERGILIO, S.; POZO, A. Generating integration test orders for aspect-oriented software with multi-objective algorithms. In: LATINAMERICAN WORKSHOP ON ASPECT ORIENTED SOFTWARE, 2011. **Proceedings...** [S.l.], 2011. 18

DALMAU, J. de; GIGOU, J. **Ariane-5: learning from flight 501 and preparing for 502**. Fev. 1997. Disponível em: <http://www.esa.int/esapub/bulletin/bullet89/dalma89.htm>. Acesso em: ago. 11, 2021. 1

DEB, K.; GOEL, T. Controlled elitist non-dominated sorting genetic algorithms for better convergence. In: ZITZLER, E.; THIELE, L.; DEB, K.; COELLO, C.A.C.; CORNE, D. (ED.). **Evolutionary Multi-Criterion Optimization**. Berlin: Springer, 2001. p. 67–81. ISBN 978-3-540-44719-1. 15

DEB, K.; JAIN, H. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: solving problems with box constraints. **IEEE Transactions on Evolutionary Computation**, v. 18, n. 4, p. 577–601, 2014. 5, 13

DEB, K.; THIELE, L.; LAUMANNS, M.; ZITZLER, E. Scalable test problems for evolutionary multiobjective optimization. In: _____. **Evolutionary multiobjective optimization: theoretical advances and applications**. London: Springer, 2005. p. 105–145. ISBN 978-1-84628-137-2. Disponível em: <https://doi.org/10.1007/1-84628-137-7_6>. 3

DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. **Introdução ao teste de software**. [S.l.]: Elsevier, 2017. 9

DEVASENA, M. S. G.; VALARMATHI, M. L. Meta heuristic search technique for dynamic test case generation. **International Journal of Computer Applications**, v. 39, n. 12, p. 1–5, Feb 2012. 17, 19

DOKEROGLU, T.; SEVINC, E.; KUCUKYILMAZ, T.; COSAR, A. A survey on new generation metaheuristic algorithms. **Computers & Industrial Engineering**, v. 137, p. 106040, 2019. ISSN 0360-8352. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0360835219304991>. 2, 11

DRAKE, J. H.; KHEIRI, A.; ÖZCAN, E.; BURKE, E. K. Recent advances in selection hyper-heuristics. **European Journal of Operational Research**, v. 285, n. 2, p. 405 – 428, 2020. ISSN 0377-2217. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0377221719306526>. 2, 13

DURILLO, J. J.; NEBRO, A. J. Jmetal: a java framework for multi-objective optimization. **Advances in Engineering Software**, v. 42, p. 760–771, 2011. ISSN 0965-9978. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0965997811001219>. 52

EDMONDS, J.; JOHNSON, E. L. Matching, euler tours and the chinese postman. **Mathematical Programming**, v. 5, p. 88–124, 1973. Https://doi.org/10.1007/BF01580113. 45

ERAS, E.; SANTIAGO JÚNIOR, V. A.; BRASIL, L. B. R. Singularity: a methodology for automatic unit test data generation for c++ applications based on model checking counterexamples. In: BRAZILIAN SYMPOSIUM ON SYSTEMATIC AND AUTOMATED SOFTWARE TESTING, 4., 2019. **Proceedings...** New York, NY, USA: ACM, 2019. p. 72–79. ISBN 978-1-4503-7648-8. 6, 27

EVERSON, R. M.; FIELDSEND, J. E. Multiobjective optimization of safety related systems: an application to short-term conflict alert. **IEEE Transactions on Evolutionary Computation**, v. 10, n. 2, p. 187–198, Apr 2006. ISSN 1089-778X. 2

FERREIRA, T. N.; KUK, J. N.; POZO, A.; VERGILIO, S. R. Product selection based on upper confidence bound MOEA/D-DRA for testing software product lines. In: CONGRESS ON EVOLUTIONARY COMPUTATION, 2016. **Proceedings...** Vancouver, BC, Canada: IEEE, 2016. p. 4135–4142. 3

FERREIRA, T. N.; LIMA, J. A. P.; STRICKLER, A.; KUK, J. N.; VERGILIO, S. R.; POZO, A. Hyper-heuristic based product selection for software product line testing. **IEEE Computational Intelligence Magazine**, v. 12, n. 2, p. 34–45, May 2017. ISSN 1556-603X. 3

FONSECA, C. M.; FLEMING, P. J. Multiobjective optimization and multiple constraint handling with evolutionary algorithms. In: DAGSTUHL SEMINAR, 2005. **Proceedings...** Dagstuhl, Germany: IBFI, 2005. ISSN 1862-4405. Disponível em: <http://drops.dagstuhl.de/opus/volltexte/2005/237>. 15

FRASER, G.; ARCURI, A. Evosuite: On the challenges of test case generation in the real world. In: INTERNATIONAL CONFERENCE ON SOFTWARE TESTING, VERIFICATION AND VALIDATION, 6., 2013. **Proceedings...** USA: IEEE Computer Society, 2013. p. 362–369. ISBN 9780769549682. Disponível em: <https://doi.org/10.1109/ICST.2013.51>. 2, 19

GARVIN, B. J.; COHEN, M. B.; DWYER, M. B. Evaluating improvements to a meta-heuristic search for constrained interaction testing. **Empirical Software Engineering**, v. 16, n. 1, p. 61–102, Feb 2011. ISSN 1573-7616. Disponível em: <https://doi.org/10.1007/s10664-010-9135-7>. 2

GUIZZO, G.; FRITSCHE, G. M.; VERGILIO, S. R.; POZO, A. T. R. A hyper-heuristic for the multi-objective integration and test order problem. In: ANNUAL CONFERENCE ON GENETIC AND EVOLUTIONARY COMPUTATION, 2015. **Proceedings...** New York, NY, USA: ACM, 2015. p. 1343–1350. ISBN 978-1-4503-3472-3. Disponível em: <http://doi.acm.org/10.1145/2739480.2754725>. 3

GUIZZO, G.; VERGILIO, S. R.; POZO, A. T. R. Evaluating a multi-objective hyper-heuristic for the integration and test order problem. In: BRAZILIAN CONFERENCE ON INTELLIGENT SYSTEMS, 2015. **Proceedings...** [S.l.], 2015. p. 1–6. 3

GUIZZO, G.; VERGILIO, S. R.; POZO, A. T. R.; FRITSCHE, G. M. A multi-objective and evolutionary hyper-heuristic applied to the integration and test order problem. **Applied Soft Computing**, v. 56, p. 331 – 344, 2017. ISSN 1568-4946. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1568494617301357>. 3, 18, 19, 54

GÓMEZ, R. H.; COELLO, C. A. C. Improved metaheuristic based on the R2 indicator for many-objective optimization. In: ANNUAL CONFERENCE ON GENETIC AND EVOLUTIONARY COMPUTATION, 2015. **Proceedings...** New York, NY, USA: ACM, 2015. p. 679–686. ISBN 978-1-4503-3472-3. Disponível em: <http://doi.acm.org/10.1145/2739480.2754776>. 5, 11, 13

HARMAN, M.; JIA, Y.; ZHANG, Y. Achievements, open problems and challenges for search based software testing. In: INTERNATIONAL CONFERENCE ON SOFTWARE TESTING, VERIFICATION AND VALIDATION, 8., 2015. **Proceedings...** Graz, Austria: IEEE, 2015. p. 1–12. ISSN 2159-4848. 1, 10

ISHIBUCHI, H.; IMADA, R.; MASUYAMA, N.; NOJIMA, Y. Comparison of hypervolume, igd and igd+ from the viewpoint of optimal distributions of solutions. In: DEB, K.; GOODMAN, E.; COELLO, C. A. C.; KLAMROTH, K.; MIETTINEN, K. (ED.). **Evolutionary multi-criterion optimization**. Berlin: Spring, 2019. p. 332–345. 16

ISHIBUCHI, H.; MASUDA, H.; NOJIMA, Y. A study on performance evaluation ability of a modified inverted generational distance indicator. In: ANNUAL CONFERENCE ON GENETIC AND EVOLUTIONARY COMPUTATION, 2015. **Proceedings...** New York, NY, USA: ACM, 2015. p. 695–702. ISBN 9781450334723. Disponível em: <https://doi.org/10.1145/2739480.2754792>. 6, 16

JAKUBOVSKI FILHO, H. L.; LIMA, J. A. P.; VERGILIO, S. R. Automatic generation of search-based algorithms applied to the feature testing of software product lines. In: BRAZILIAN SYMPOSIUM ON SOFTWARE ENGINEERING, 31. **Proceedings...** New York, NY, USA: ACM, 2017. p. 114–123. ISBN 978-1-4503-5326-7. Disponível em: <http://doi.acm.org/10.1145/3131151.3131152>. 3

JIA, Y.; COHEN, M. B.; HARMAN, M.; PETKE, J. Learning combinatorial interaction test generation strategies using hyperheuristic search. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 37., 2015. **Proceedings...** Florence, Italy: IEEE, 2015. v. 1, p. 540–550. ISSN 0270-5257. 3

KHARI, M.; KUMAR, P. An extensive evaluation of search-based software testing: a review. **Soft Computing**, v. 23, n. 6, p. 1933–1946, Mar 2019. ISSN 1433-7479. Disponível em: <https://doi.org/10.1007/s00500-017-2906-y>. 1

KÖRTING, T. S.; GARCIA FONSECA, L. M.; CÂMARA, G. Geodma—geographic data mining analyst. **Computers & Geosciences**, v. 57, p. 133 – 145, 2013. ISSN 0098-3004. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0098300413000538>. 1, 4, 6, 54

LAKHOTIA, K.; HARMAN, M.; GROSS, H. Austin: a tool for search based software testing for the c language and its evaluation on deployed automotive systems. **Proceedings - 2nd International Symposium on Search Based Software Engineering, SSBSE 2010**, 09 2010. 2, 19

LEVESON, N. G.; TURNER, C. S. An investigation of the Therac-25 accidents. **Computer**, v. 26, n. 7, p. 18–41, 1993. 1

LI, W.; ÖZCAN, E.; JOHN, R. A learning automata-based multiobjective hyper-heuristic. **IEEE Transactions on Evolutionary Computation**, v. 23, n. 1, p. 59–73, Feb 2019. ISSN 1089-778X. 3, 57

LIMA, J. A. d. P. **Uma abordagem baseada em hiper-heurística e otimização multi-objetivo para o teste de mutação de ordem superior**. Dissertação (Mestrado em Informática) — Universidade Federal do Paraná, Curitiba, 2017. 12

LIMA, J. A. P.; VERGILIO, S. R. A multi-objective optimization approach for selection of second order mutant generation strategies. In: BRAZILIAN SYMPOSIUM ON SYSTEMATIC AND AUTOMATED SOFTWARE TESTING, 2., 2017. **Proceedings...** New York, NY, USA: ACM, 2017. p. 6:1–6:10. ISBN 978-1-4503-5302-1. Disponível em: <http://doi.acm.org/10.1145/3128473.3128479>. 3

LIN, J.; LUO, C.; CAI, S.; SU, K.; HAO, D.; ZHANG, L. Tca: an efficient two-mode meta-heuristic algorithm for combinatorial test generation (t). In: INTERNATIONAL CONFERENCE ON AUTOMATED SOFTWARE ENGINEERING, 30., 2015. **Proceedings...** Lincoln, NE, USA: IEEE, 2015. p. 494–505. 17, 19

MAASHI, M.; ÖZCAN, E.; KENDALL, G. A multi-objective hyper-heuristic based on choice function. **Expert Systems with Applications**, v. 41, n. 9, p. 4475 – 4493, 2014. ISSN 0957-4174. Disponível em: <http://www.sciencedirect.com/science/article/pii/S095741741400013X>. 3, 6, 15

MAHMOUD, T.; AHMED, B. S. An efficient strategy for covering array construction with fuzzy logic-based adaptive swarm optimization for software testing use. **Expert Systems with Applications**, v. 42, n. 22, p. 8753 – 8765, 2015. ISSN 0957-4174. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0957417415004893>. 2

MALDONADO, J. C.; BARBOSA, E. F.; VINCENZI, A. M.; DELAMARO, M. E.; SOUZA, S. d. R. S.; JINO, M. **Introducao ao teste de software (versão 2004-01)**. [S.l.]: ICMC-USP, 2004. 75

MARIANI, T.; GUIZZO, G.; VERGILIO, S. R.; POZO, A. T. R. Grammatical evolution for the multi-objective integration and test order problem. In: GENETIC AND EVOLUTIONARY COMPUTATION CONFERENCE, 31., 2016. **Proceedings...** New York, NY, USA: ACM, 2016. p. 1069–1076. ISBN 978-1-4503-4206-3. Disponível em: <http://doi.acm.org/10.1145/2908812.2908816>. 3, 18, 19

MCCAFFREY, J. D. An empirical study of pairwise test set generation using a genetic algorithm. In: INTERNATIONAL CONFERENCE ON INFORMATION TECHNOLOGY: NEW GENERATIONS, 2010. **Proceedings...** Las Vegas, NV, USA: IEEE, 2010. p. 992–997. 2

MCMINN, P. Search-based software testing: past, present and future. In: INTERNATIONAL CONFERENCE ON SOFTWARE TESTING, VERIFICATION AND VALIDATION WORKSHOPS, 4., 2011. **Proceedings...** Berlin, Germany: IEEE, 2011. p. 153–163. 10

MCMINN, P.; WRIGHT, C. J.; KINNEER, C.; MCCURDY, C. J.; CAMARA, M.; KAPFHAMMER, G. M. SchemaAnalyst: search-based test data generation for relational database schemas. In: INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE AND EVOLUTION, 2016. **Proceedings...** Raleigh, NC, USA: IEEE, 2016. p. 586–590. 2

MICHAIL, D.; KINABLE, J.; NAVEH, B.; SICHI, J. V. Jgrapht—a java library for graph data structures and algorithms. **ACM Transactions on Mathematical Software**, v. 46, n. 2, maio 2020. 52

MILLER, W.; SPOONER, D. Automatic generation of floating-point test data. **IEEE Transactions on Software Engineering**, v. 2, p. 223 – 226, 10 1976. 10

MKAOUER, M. W.; KESSENTINI, M. Model transformation using multiobjective optimization. In: HURSON, A. (Ed.). **Advances in Computers**. Elsevier, 2014. v. 92, p. 161 – 202. Disponível em: <http://www.sciencedirect.com/science/article/pii/B9780124202320000040>. 40

OGATA, S.; MATSUURA, S. A method of automatic integration test case generation from uml-based scenario. **WSEAS Transactions on Information Science and Applications**, v. 7, 12 2010. 3

ÖZCAN, E.; MISIR, M.; OCHOA, G.; BURKE, E. K. A reinforcement learning-great-deluge hyper-heuristic for examination timetabling. **International Journal of Applied Metaheuristic Computing**, v. 1, n. 1, p. 39–59, jan. 2010. ISSN 1947-8283. Disponível em: <https://doi.org/10.4018/jamc.2010102603>. 14

PANICHELLA, A.; KIFETEW, F. M.; TONELLA, P. Automated test case generation as a many-objective optimisation problem with dynamic selection of the targets. **IEEE Transactions on Software Engineering**, v. 44, n. 2, p. 122–158, 2018. 17, 19, 27

PARR, T. **ANTLR 4 documentation**. 2019. Disponível em: <https://github.com/antlr/antlr4/blob/master/doc/index.md>. Acesso em: 20 abr. 2020. 52

PENTA, M. D.; CANFORA, G.; ESPOSITO, G.; MAZZA, V.; BRUNO, M. Search-based testing of service level agreements. In: ANNUAL CONFERENCE ON GENETIC AND EVOLUTIONARY COMPUTATION, 9., 2007. **Proceedings...** New York, NY, USA: ACM, 2007. p. 1090–1097. ISBN 978-1-59593-697-4. Disponível em: <http://doi.acm.org/10.1145/1276958.1277174>. 2

PERKUSICH, M.; Chaves e Silva, L.; COSTA, A.; RAMOS, F.; SARAIVA, R.; FREIRE, A.; DILORENZO, E.; DANTAS, E.; SANTOS, D.; GORGÔNIO, K.; ALMEIDA, H.; PERKUSICH, A. Intelligent software engineering in the context of agile software development: a systematic literature review. **Information and Software Technology**, v. 119, p. 106241, 2020. ISSN 0950-5849. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0950584919302587>. 4

PETKE, J.; COHEN, M. B.; HARMAN, M.; YOO, S. Practical combinatorial interaction testing: empirical findings on efficiency and early fault detection. **IEEE Transactions on Software Engineering**, v. 41, n. 9, p. 901–924, Sept 2015. ISSN 0098-5589. 2

PINTE, F.; SAGLIETTI, F.; OSTER, N. Automatic generation of optimized integration test data by genetic algorithms. In: SOFTWARE ENGINEERING, 2008. **Proceedings...** München, 2008. 3, 18, 19, 27

PONZIO, P.; AGUIRRE, N.; FRIAS, M. F.; VISSER, W. Field-exhaustive testing. In: ACM SIGSOFT INTERNATIONAL SYMPOSIUM ON FOUNDATIONS OF SOFTWARE ENGINEERING, 24., 2016. **Proceedings...** New York, USA: ACM, 2016. p. 908–919. ISBN 9781450342186. Disponível em: <https://doi.org/10.1145/2950290.2950336>. 1

ROJAS, J. M.; VIVANTI, M.; ARCURI, A.; FRASER, G. A detailed investigation of the effectiveness of whole test suite generation. **Empirical Software Engineering**, v. 22, p. 852—893, 2017. 17, 19, 27

SAEED, A.; HAMID, S. H. A.; MUSTAFA, M. B. The experimental applications of search-based techniques for model-based testing: taxonomy and systematic literature review. **Applied Soft Computing**, v. 49, p. 1094 – 1117, 2016. ISSN

1568-4946. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1568494616304240>. 1

SALES, C. P. **INMEHY-STV**. 2020. Disponível em: <https://github.com/Pssales/InMeHy>. Acesso em: 13 out. 2020. 51

_____. **INMEHY-STF**. 2021. Disponível em: <https://github.com/Pssales/INMEHY-STF>. Acesso em: 08 abr. 2021. 52

SALES, C. P.; SANTIAGO JÚNIOR, V. A. Investigating multi and many-objective metaheuristics to support software integration testing. In: BRAZILIAN SYMPOSIUM ON SYSTEMATIC AND AUTOMATED SOFTWARE TESTING, 5., 2020. **Proceedings...** New York, USA: ACM, 2020. p. 1–10. ISBN 9781450387552. Disponível em: <https://doi.org/10.1145/3425174.3425175>. 5, 6, 21, 65, 135

SANTIAGO JÚNIOR, V. A. **SOLIMVA: a methodology for generating model-based test cases from natural language requirements and detecting incompleteness in software specifications**. Tese (Doutorado em Computação Aplicada) — Instituto Nacional de Pesquisas Espaciais (INPE), São José dos Campos, 2011. 3, 9

SANTIAGO JÚNIOR, V. A.; ÖZCAN, E.; CARVALHO, V. R. de. Hyper-heuristics based on reinforcement learning, balanced heuristic selection and group decision acceptance. **Applied Soft Computing**, v. 97, p. 106760, 2020. ISSN 1568-4946. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1568494620306980>. 2, 3, 5, 15, 57, 72

SANTIAGO JÚNIOR, V. A.; SALES, C. P. Metaheuristics and hyper-heuristics based on evolutionary algorithms for software integration testing. In: **Proceedings of the 5th International Joint Conference on Advances in Computational Intelligence**. [S.l.: s.n.], 2021. p. 1–22. Artigo em processo de publicação. 22, 135

SANTIAGO JÚNIOR, V. A.; SILVA, W.; VIJAYKUMAR, N. Shortening test case execution time for embedded software. In: CONFERENCE ON SECURE SYSTEM INTEGRATION AND REALIABILITY IMPROVEMENT, 2., 2018. **Proceedings...** Yokohama, Japan: IEEE, 2008. p. 81 – 88. ISBN 978-0-7695-3266-0. 1

SANTIAGO JÚNIOR, V. A.; VIJAYKUMAR, N. L. Generating model-based test cases from natural language requirements for space application software. **Software Quality Journal**, v. 20, n. 1, p. 77–143, 2012. DOI: 10.1007/s11219-011-9155-6. 1

SHARMA, C.; SIBAL, R. Applications of different metaheuristic techniques for finding optimal tst order during integration testing of object oriented systems and their comparative study. **arXiv.org**, 2014. 18

SHIN, Y.; CHOI, Y.; LEE, W. J. Integration testing through reusing representative unit test cases for high-confidence medical software. **Computers in Biology and Medicine**, v. 43, n. 5, p. 434 – 443, 2013. ISSN 0010-4825. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0010482513000450>. 3

TERRAAMAZON. **TerraAmazon**. 2020. Disponível em: <http://www.terraamazon.dpi.inpe.br/>. Acesso em: 22 abr. 2020. 4

TIOBE. **TIOBE index for August 2021**. 2021. Disponível em: <https://www.tiobe.com/tiobe-index/>. Acesso em: 11 ago. 2021. 4

TONELLA, P. Evolutionary testing of classes. **ACM SIGSOFT Software Engineering Notes**, v. 29, n. 4, p. 119–128, 07 2004. 17, 19

VINCENZI, A.; MALDONADO, J.; BARBOSA, E.; DELAMARO, M. Unit and integration testing strategies for c programs using mutation. **Software Testing, Verification and Reliability**, v. 11, p. 249 – 268, 12 2001. 3

VRIGAZOV, H. **grammars-v4/cpp**. 2019. Disponível em: <https://github.com/antlr/grammars-v4/tree/master/cpp>. Acesso em: 18 abr. 2020. 31

WEGENER, J.; BÜHLER, O. Evaluation of different fitness functions for the evolutionary testing of an autonomous parking system. In: DEB, K. (Ed.). **Genetic and evolutionary computation – GECCO 2004**. Berlin, Heidelberg: Springer, 2004. p. 1400–1412. ISBN 978-3-540-24855-2. 2

WHILE, L.; BRADSTREET, L.; BARONE, L.; HINGSTON, P. Heuristics for optimizing the calculation of hypervolume for multi-objective optimization problems. In: CONGRESS ON EVOLUTIONARY COMPUTATION, 2005. **Proceedings...** Edinburgh, UK: IEEE, 2005. v. 3, p. 2225 – 2232. ISBN 0-7803-9363-5. 15

WOLPERT, D.; MACREADY, W. Coevolutionary free lunches. **IEEE Transactions on Evolutionary Computation**, v. 9, n. 6, p. 721–735, 2005. 73

WU, H.; NIE, C.; KUO, F. C.; LEUNG, H.; COLBOURN, C. J. A discrete particle swarm optimization for covering array generation. **IEEE Transactions on Evolutionary Computation**, v. 19, n. 4, p. 575–591, Aug 2015. ISSN 1089-778X. 2

YANMEI, Z.; JIANG, S.; WANG, X.; CHEN, R.; ZHANG, M. An optimization algorithm applied to the class integration and test order problem. **Soft Computing**, v. 23, 02 2018. 16, 19

YOSHIDA, H.; Li, G.; KAMIYA, T.; GHOSH, I.; Rajan, S.; TOKUMOTO, S.; MUNAKATA, K.; UEHARA, T. Klover: automatic test generation for c and c++ programs, using symbolic execution. **IEEE Software**, v. 34, n. 5, p. 30–37, 2017. 4

ZAMLI, K. Z.; ALKAZEMI, B. Y.; KENDALL, G. A tabu search hyper-heuristic strategy for t-way test suite generation. **Applied Soft Computing**, v. 44, p. 57 – 74, 2016. ISSN 1568-4946. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1568494616301302>. 17, 19

ZAMLI, K. Z.; DIN, F.; KENDALL, G.; AHMED, B. S. An experimental study of hyper-heuristic selection and acceptance mechanism for combinatorial t-way test suite generation. **Information Sciences**, v. 399, p. 121 – 153, 2017. ISSN 0020-0255. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0020025517305820>. 3

ZHANG, Q.; ZHOU, A.; ZHAO, S.; SUGANTHAN, P. N.; LIU, W.; TIWARI, S. **Multiobjective optimization Test Instances for the CEC 2009 Special Session and Competition**. 2009. Disponível em: https://bit.ly/3CBpn2R, Acesso em: 11 ago. 2021. 3

ZITZLER, E.; KÜNZLI, S. Indicator-based selection in multiobjective search. In: YAO, X. ET AL. (ED.). **Parallel problem solving from nature**. Berlin: Springer, 2004. p. 832–842. 5, 12

ZITZLER, E.; LAUMANNS, M.; THIELE, L. Spea2: improving the strength pareto evolutionary algorithm for multiobjective optimization. In: YAO, X. ET AL. (ED.). **Parallel problem solving from nature**. Berlin: Springer, 2001. v. 3242, p. 742–751. 5, 12, 15

ZITZLER, E.; THIELE, L. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. **IEEE Transactions on Evolutionary Computation**, v. 3, n. 4, p. 257–271, 1999. 6, 15

ZITZLER, E.; THIELE, L.; LAUMANNS, M.; FONSECA, C. M.; FONSECA, V. G. da. Performance assessment of multiobjective optimizers: an analysis and review. **IEEE Transactions on Evolutionary Computation**, v. 7, n. 2, p. 117–132, abr. 2003. ISSN 1089-778X. Disponível em: <https://doi.org/10.1109/TEVC.2003.810758>. 6, 16

APÊNDICE A -EXEMPLOS DE ÁRVORES SINTÁTICAS

A.1 Árvore sintática do arquivo conta.hpp

```
1  class reader.CPP14Parser$TranslationunitContext ->
       usingnamespacestd;classConta{Usuariousuario_;floatsaldo_;
       public:voidinicializa(Usuariousuario,floats);voiddeposita(
       floatvalor);};<EOF>
2  class reader.CPP14Parser$DeclarationseqContext ->
       usingnamespacestd;classConta{Usuariousuario_;floatsaldo_;
       public:voidinicializa(Usuariousuario,floats);voiddeposita(
       floatvalor);};
3  class reader.CPP14Parser$DeclarationseqContext ->
       usingnamespacestd;
4  class reader.CPP14Parser$DeclarationContext ->
       usingnamespacestd;
5  class reader.CPP14Parser$BlockdeclarationContext ->
       usingnamespacestd;
6  class reader.CPP14Parser$UsingdirectiveContext ->
       usingnamespacestd;
7  class org.antlr.v4.runtime.tree.TerminalNodeImpl -> using
8  class org.antlr.v4.runtime.tree.TerminalNodeImpl -> namespace
9  class reader.CPP14Parser$NamespacenameContext -> std
10 class reader.CPP14Parser$OriginalnamespacenameContext -> std
11 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> std
12 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> ;
13 class reader.CPP14Parser$DeclarationContext -> classConta{
       Usuariousuario_;floatsaldo_;public:voidinicializa(
       Usuariousuario,floats);voiddeposita(floatvalor);};
14 class reader.CPP14Parser$BlockdeclarationContext ->
       classConta{Usuariousuario_;floatsaldo_;public:
       voidinicializa(Usuariousuario,floats);voiddeposita(
       floatvalor);};
15 class reader.CPP14Parser$SimpledeclarationContext ->
       classConta{Usuariousuario_;floatsaldo_;public:
       voidinicializa(Usuariousuario,floats);voiddeposita(
       floatvalor);};
16 class reader.CPP14Parser$DeclspecifierseqContext ->
       classConta{Usuariousuario_;floatsaldo_;public:
       voidinicializa(Usuariousuario,floats);voiddeposita(
       floatvalor);}
```

```
17 class reader.CPP14Parser$DeclspecifierContext -> classConta{
     Usuariousuario_;floatsaldo_;public:voidinicializa(
     Usuariousuario,floats);voiddeposita(floatvalor);}
18 class reader.CPP14Parser$TypespecifierContext -> classConta{
     Usuariousuario_;floatsaldo_;public:voidinicializa(
     Usuariousuario,floats);voiddeposita(floatvalor);}
19 class reader.CPP14Parser$ClassspecifierContext -> classConta{
     Usuariousuario_;floatsaldo_;public:voidinicializa(
     Usuariousuario,floats);voiddeposita(floatvalor);}
20 class reader.CPP14Parser$ClassheadContext -> classConta
21 class reader.CPP14Parser$ClasskeyContext -> class
22 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> class
23 class reader.CPP14Parser$ClassheadnameContext -> Conta
24 class reader.CPP14Parser$ClassnameContext -> Conta
25 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> Conta
26 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> {
27 class reader.CPP14Parser$MemberspecificationContext ->
     Usuariousuario_;floatsaldo_;public:voidinicializa(
     Usuariousuario,floats);voiddeposita(floatvalor);
28 class reader.CPP14Parser$MemberdeclarationContext ->
     Usuariousuario_;
29 class reader.CPP14Parser$DeclspecifierseqContext -> Usuario
30 class reader.CPP14Parser$DeclspecifierContext -> Usuario
31 class reader.CPP14Parser$TypespecifierContext -> Usuario
32 class reader.CPP14Parser$TrailingtypespecifierContext ->
     Usuario
33 class reader.CPP14Parser$SimpletypespecifierContext ->
     Usuario
34 class reader.CPP14Parser$ThetypenameContext -> Usuario
35 class reader.CPP14Parser$ClassnameContext -> Usuario
36 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> Usuario
37 class reader.CPP14Parser$MemberdeclaratorlistContext ->
     usuario_
38 class reader.CPP14Parser$MemberdeclaratorContext -> usuario_
39 class reader.CPP14Parser$DeclaratorContext -> usuario_
40 class reader.CPP14Parser$PtrdeclaratorContext -> usuario_
41 class reader.CPP14Parser$NoptrdeclaratorContext -> usuario_
42 class reader.CPP14Parser$DeclaratoridContext -> usuario_
43 class reader.CPP14Parser$IdexpressionContext -> usuario_
44 class reader.CPP14Parser$UnqualifiedidContext -> usuario_
```

```
45 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> usuario_
46 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> ;
47 class reader.CPP14Parser$MemberspecificationContext ->
     floatsaldo_;public:voidinicializa(Usuariousuario,floats);
     voiddeposita(floatvalor);
48 class reader.CPP14Parser$MemberdeclarationContext ->
     floatsaldo_;
49 class reader.CPP14Parser$DeclspecifierseqContext -> float
50 class reader.CPP14Parser$DeclspecifierContext -> float
51 class reader.CPP14Parser$TypespecifierContext -> float
52 class reader.CPP14Parser$TrailingtypespecifierContext ->
     float
53 class reader.CPP14Parser$SimpletypespecifierContext -> float
54 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> float
55 class reader.CPP14Parser$MemberdeclaratorlistContext ->
     saldo_
56 class reader.CPP14Parser$MemberdeclaratorContext -> saldo_
57 class reader.CPP14Parser$DeclaratorContext -> saldo_
58 class reader.CPP14Parser$PtrdeclaratorContext -> saldo_
59 class reader.CPP14Parser$NoptrdeclaratorContext -> saldo_
60 class reader.CPP14Parser$DeclaratoridContext -> saldo_
61 class reader.CPP14Parser$IdexpressionContext -> saldo_
62 class reader.CPP14Parser$UnqualifiedidContext -> saldo_
63 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> saldo_
64 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> ;
65 class reader.CPP14Parser$MemberspecificationContext -> public
     :voidinicializa(Usuariousuario,floats);voiddeposita(
     floatvalor);
66 class reader.CPP14Parser$AccessspecifierContext -> public
67 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> public
68 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> :
69 class reader.CPP14Parser$MemberspecificationContext ->
     voidinicializa(Usuariousuario,floats);voiddeposita(
     floatvalor);
70 class reader.CPP14Parser$MemberdeclarationContext ->
     voidinicializa(Usuariousuario,floats);
71 class reader.CPP14Parser$DeclspecifierseqContext -> void
72 class reader.CPP14Parser$DeclspecifierContext -> void
73 class reader.CPP14Parser$TypespecifierContext -> void
74 class reader.CPP14Parser$TrailingtypespecifierContext -> void
```

```
75 class reader.CPP14Parser$SimpletypespecifierContext -> void
76 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> void
77 class reader.CPP14Parser$MemberdeclaratorlistContext ->
     inicializa(Usuariousuario,floats)
78 class reader.CPP14Parser$MemberdeclaratorContext ->
     inicializa(Usuariousuario,floats)
79 class reader.CPP14Parser$DeclaratorContext -> inicializa(
     Usuariousuario,floats)
80 class reader.CPP14Parser$PtrdeclaratorContext -> inicializa(
     Usuariousuario,floats)
81 class reader.CPP14Parser$NoptrdeclaratorContext -> inicializa
     (Usuariousuario,floats)
82 class reader.CPP14Parser$NoptrdeclaratorContext -> inicializa
83 class reader.CPP14Parser$DeclaratoridContext -> inicializa
84 class reader.CPP14Parser$IdexpressionContext -> inicializa
85 class reader.CPP14Parser$UnqualifiedidContext -> inicializa
86 class org.antlr.v4.runtime.tree.TerminalNodeImpl ->
     inicializa
87 class reader.CPP14Parser$ParametersandqualifiersContext -> (
     Usuariousuario,floats)
88 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> (
89 class reader.CPP14Parser$ParameterdeclarationclauseContext ->
      Usuariousuario,floats
90 class reader.CPP14Parser$ParameterdeclarationlistContext ->
     Usuariousuario,floats
91 class reader.CPP14Parser$ParameterdeclarationlistContext ->
     Usuariousuario
92 class reader.CPP14Parser$ParameterdeclarationContext ->
     Usuariousuario
93 class reader.CPP14Parser$DeclspecifierseqContext -> Usuario
94 class reader.CPP14Parser$DeclspecifierContext -> Usuario
95 class reader.CPP14Parser$TypespecifierContext -> Usuario
96 class reader.CPP14Parser$TrailingtypespecifierContext ->
     Usuario
97 class reader.CPP14Parser$SimpletypespecifierContext ->
     Usuario
98 class reader.CPP14Parser$ThetypenameContext -> Usuario
99 class reader.CPP14Parser$ClassnameContext -> Usuario
100 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> Usuario
101 class reader.CPP14Parser$DeclaratorContext -> usuario
```

```
102 class reader.CPP14Parser$PtrdeclaratorContext -> usuario
103 class reader.CPP14Parser$NoptrdeclaratorContext -> usuario
104 class reader.CPP14Parser$DeclaratoridContext -> usuario
105 class reader.CPP14Parser$IdexpressionContext -> usuario
106 class reader.CPP14Parser$UnqualifiedidContext -> usuario
107 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> usuario
108 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> ,
109 class reader.CPP14Parser$ParameterdeclarationContext ->
      floats
110 class reader.CPP14Parser$DeclspecifierseqContext -> float
111 class reader.CPP14Parser$DeclspecifierContext -> float
112 class reader.CPP14Parser$TypespecifierContext -> float
113 class reader.CPP14Parser$TrailingtypespecifierContext ->
      float
114 class reader.CPP14Parser$SimpletypespecifierContext -> float
115 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> float
116 class reader.CPP14Parser$DeclaratorContext -> s
117 class reader.CPP14Parser$PtrdeclaratorContext -> s
118 class reader.CPP14Parser$NoptrdeclaratorContext -> s
119 class reader.CPP14Parser$DeclaratoridContext -> s
120 class reader.CPP14Parser$IdexpressionContext -> s
121 class reader.CPP14Parser$UnqualifiedidContext -> s
122 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> s
123 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> )
124 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> ;
125 class reader.CPP14Parser$MemberspecificationContext ->
      voiddeposita(floatvalor);
126 class reader.CPP14Parser$MemberdeclarationContext ->
      voiddeposita(floatvalor);
127 class reader.CPP14Parser$DeclspecifierseqContext -> void
128 class reader.CPP14Parser$DeclspecifierContext -> void
129 class reader.CPP14Parser$TypespecifierContext -> void
130 class reader.CPP14Parser$TrailingtypespecifierContext -> void
131 class reader.CPP14Parser$SimpletypespecifierContext -> void
132 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> void
133 class reader.CPP14Parser$MemberdeclaratorlistContext ->
      deposita(floatvalor)
134 class reader.CPP14Parser$MemberdeclaratorContext -> deposita(
      floatvalor)
135 class reader.CPP14Parser$DeclaratorContext -> deposita(
```

```
    floatvalor)
136 class reader.CPP14Parser$PtrdeclaratorContext -> deposita(
    floatvalor)
137 class reader.CPP14Parser$NoptrdeclaratorContext -> deposita(
    floatvalor)
138 class reader.CPP14Parser$NoptrdeclaratorContext -> deposita
139 class reader.CPP14Parser$DeclaratoridContext -> deposita
140 class reader.CPP14Parser$IdexpressionContext -> deposita
141 class reader.CPP14Parser$UnqualifiedidContext -> deposita
142 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> deposita
143 class reader.CPP14Parser$ParametersandqualifiersContext -> (
    floatvalor)
144 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> (
145 class reader.CPP14Parser$ParameterdeclarationclauseContext ->
     floatvalor
146 class reader.CPP14Parser$ParameterdeclarationlistContext ->
    floatvalor
147 class reader.CPP14Parser$ParameterdeclarationContext ->
    floatvalor
148 class reader.CPP14Parser$DeclspecifierseqContext -> float
149 class reader.CPP14Parser$DeclspecifierContext -> float
150 class reader.CPP14Parser$TypespecifierContext -> float
151 class reader.CPP14Parser$TrailingtypespecifierContext ->
    float
152 class reader.CPP14Parser$SimpletypespecifierContext -> float
153 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> float
154 class reader.CPP14Parser$DeclaratorContext -> valor
155 class reader.CPP14Parser$PtrdeclaratorContext -> valor
156 class reader.CPP14Parser$NoptrdeclaratorContext -> valor
157 class reader.CPP14Parser$DeclaratoridContext -> valor
158 class reader.CPP14Parser$IdexpressionContext -> valor
159 class reader.CPP14Parser$UnqualifiedidContext -> valor
160 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> valor
161 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> )
162 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> ;
163 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> }
164 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> ;
165 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> <EOF>
```

A.2 Árvore sintática do arquivo conta.cpp

```
1  class reader.CPP14Parser$TranslationunitContext ->
       voidAccount::initialize(Useruser,floatb){user_=user;
       balance_=b;if(balance_<0){cout<<"error on create account
       !!!"<<endl;}}voidAccount::deposit(floatvalue){balance_=
       balance_+value;}<EOF>
2  class reader.CPP14Parser$DeclarationseqContext -> voidAccount
       ::initialize(Useruser,floatb){user_=user;balance_=b;if(
       balance_<0){cout<<"error on create account!!!"<<endl;}}
       voidAccount::deposit(floatvalue){balance_=balance_+value;}
3  class reader.CPP14Parser$DeclarationseqContext -> voidAccount
       ::initialize(Useruser,floatb){user_=user;balance_=b;if(
       balance_<0){cout<<"error on create account!!!"<<endl;}}
4  class reader.CPP14Parser$DeclarationContext -> voidAccount::
       initialize(Useruser,floatb){user_=user;balance_=b;if(
       balance_<0){cout<<"error on create account!!!"<<endl;}}
5  class reader.CPP14Parser$FunctiondefinitionContext ->
       voidAccount::initialize(Useruser,floatb){user_=user;
       balance_=b;if(balance_<0){cout<<"error on create account
       !!!"<<endl;}}
6  class reader.CPP14Parser$DeclspecifierseqContext -> void
7  class reader.CPP14Parser$DeclspecifierContext -> void
8  class reader.CPP14Parser$TypespecifierContext -> void
9  class reader.CPP14Parser$TrailingtypespecifierContext -> void
10 class reader.CPP14Parser$SimpletypespecifierContext -> void
11 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> void
12 class reader.CPP14Parser$DeclaratorContext -> Account::
       initialize(Useruser,floatb)
13 class reader.CPP14Parser$PtrdeclaratorContext -> Account::
       initialize(Useruser,floatb)
14 class reader.CPP14Parser$NoptrdeclaratorContext -> Account::
       initialize(Useruser,floatb)
15 class reader.CPP14Parser$NoptrdeclaratorContext -> Account::
       initialize
16 class reader.CPP14Parser$DeclaratoridContext -> Account::
       initialize
17 class reader.CPP14Parser$IdexpressionContext -> Account::
       initialize
18 class reader.CPP14Parser$QualifiedidContext -> Account::
       initialize
19 class reader.CPP14Parser$NestednamespecifierContext ->
```

```
    Account::
20 class reader.CPP14Parser$ThetypenameContext -> Account
21 class reader.CPP14Parser$ClassnameContext -> Account
22 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> Account
23 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> ::
24 class reader.CPP14Parser$UnqualifiedidContext -> initialize
25 class org.antlr.v4.runtime.tree.TerminalNodeImpl ->
    initialize
26 class reader.CPP14Parser$ParametersandqualifiersContext -> (
    Useruser,floatb)
27 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> (
28 class reader.CPP14Parser$ParameterdeclarationclauseContext ->
     Useruser,floatb
29 class reader.CPP14Parser$ParameterdeclarationlistContext ->
    Useruser,floatb
30 class reader.CPP14Parser$ParameterdeclarationlistContext ->
    Useruser
31 class reader.CPP14Parser$ParameterdeclarationContext ->
    Useruser
32 class reader.CPP14Parser$DeclspecifierseqContext -> User
33 class reader.CPP14Parser$DeclspecifierContext -> User
34 class reader.CPP14Parser$TypespecifierContext -> User
35 class reader.CPP14Parser$TrailingtypespecifierContext -> User
36 class reader.CPP14Parser$SimpletypespecifierContext -> User
37 class reader.CPP14Parser$ThetypenameContext -> User
38 class reader.CPP14Parser$ClassnameContext -> User
39 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> User
40 class reader.CPP14Parser$DeclaratorContext -> user
41 class reader.CPP14Parser$PtrdeclaratorContext -> user
42 class reader.CPP14Parser$NoptrdeclaratorContext -> user
43 class reader.CPP14Parser$DeclaratoridContext -> user
44 class reader.CPP14Parser$IdexpressionContext -> user
45 class reader.CPP14Parser$UnqualifiedidContext -> user
46 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> user
47 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> ,
48 class reader.CPP14Parser$ParameterdeclarationContext ->
    floatb
49 class reader.CPP14Parser$DeclspecifierseqContext -> float
50 class reader.CPP14Parser$DeclspecifierContext -> float
51 class reader.CPP14Parser$TypespecifierContext -> float
```

```
52 class reader.CPP14Parser$TrailingtypespecifierContext ->
     float
53 class reader.CPP14Parser$SimpletypespecifierContext -> float
54 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> float
55 class reader.CPP14Parser$DeclaratorContext -> b
56 class reader.CPP14Parser$PtrdeclaratorContext -> b
57 class reader.CPP14Parser$NoptrdeclaratorContext -> b
58 class reader.CPP14Parser$DeclaratoridContext -> b
59 class reader.CPP14Parser$IdexpressionContext -> b
60 class reader.CPP14Parser$UnqualifiedidContext -> b
61 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> b
62 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> )
63 class reader.CPP14Parser$FunctionbodyContext -> {user_=user;
     balance_=b;if(balance_<0){cout<<"error on create account
     !!!"<<endl;}}
64 class reader.CPP14Parser$CompoundstatementContext -> {user_=
     user;balance_=b;if(balance_<0){cout<<"error on create
     account!!!"<<endl;}}
65 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> {
66 class reader.CPP14Parser$StatementseqContext -> user_=user;
     balance_=b;if(balance_<0){cout<<"error on create account
     !!!"<<endl;}
67 class reader.CPP14Parser$StatementseqContext -> user_=user;
     balance_=b;
68 class reader.CPP14Parser$StatementseqContext -> user_=user;
69 class reader.CPP14Parser$StatementContext -> user_=user;
70 class reader.CPP14Parser$ExpressionstatementContext -> user_=
     user;
71 class reader.CPP14Parser$ExpressionContext -> user_=user
72 class reader.CPP14Parser$AssignmentexpressionContext -> user_
     =user
73 class reader.CPP14Parser$LogicalorexpressionContext -> user_
74 class reader.CPP14Parser$LogicalandexpressionContext -> user_
75 class reader.CPP14Parser$InclusiveorexpressionContext ->
     user_
76 class reader.CPP14Parser$ExclusiveorexpressionContext ->
     user_
77 class reader.CPP14Parser$AndexpressionContext -> user_
78 class reader.CPP14Parser$EqualityexpressionContext -> user_
79 class reader.CPP14Parser$RelationalexpressionContext -> user_
```

```
80 class reader.CPP14Parser$ShiftexpressionContext -> user_
81 class reader.CPP14Parser$AdditiveexpressionContext -> user_
82 class reader.CPP14Parser$MultiplicativeexpressionContext ->
     user_
83 class reader.CPP14Parser$PmexpressionContext -> user_
84 class reader.CPP14Parser$CastexpressionContext -> user_
85 class reader.CPP14Parser$UnaryexpressionContext -> user_
86 class reader.CPP14Parser$PostfixexpressionContext -> user_
87 class reader.CPP14Parser$PrimaryexpressionContext -> user_
88 class reader.CPP14Parser$IdexpressionContext -> user_
89 class reader.CPP14Parser$UnqualifiedidContext -> user_
90 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> user_
91 class reader.CPP14Parser$AssignmentoperatorContext -> =
92 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> =
93 class reader.CPP14Parser$InitializerclauseContext -> user
94 class reader.CPP14Parser$AssignmentexpressionContext -> user
95 class reader.CPP14Parser$ConditionalexpressionContext -> user
96 class reader.CPP14Parser$LogicalorexpressionContext -> user
97 class reader.CPP14Parser$LogicalandexpressionContext -> user
98 class reader.CPP14Parser$InclusiveorexpressionContext -> user
99 class reader.CPP14Parser$ExclusiveorexpressionContext -> user
100 class reader.CPP14Parser$AndexpressionContext -> user
101 class reader.CPP14Parser$EqualityexpressionContext -> user
102 class reader.CPP14Parser$RelationalexpressionContext -> user
103 class reader.CPP14Parser$ShiftexpressionContext -> user
104 class reader.CPP14Parser$AdditiveexpressionContext -> user
105 class reader.CPP14Parser$MultiplicativeexpressionContext ->
     user
106 class reader.CPP14Parser$PmexpressionContext -> user
107 class reader.CPP14Parser$CastexpressionContext -> user
108 class reader.CPP14Parser$UnaryexpressionContext -> user
109 class reader.CPP14Parser$PostfixexpressionContext -> user
110 class reader.CPP14Parser$PrimaryexpressionContext -> user
111 class reader.CPP14Parser$IdexpressionContext -> user
112 class reader.CPP14Parser$UnqualifiedidContext -> user
113 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> user
114 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> ;
115 class reader.CPP14Parser$StatementContext -> balance_=b;
116 class reader.CPP14Parser$ExpressionstatementContext ->
     balance_=b;
```

```
117 class reader.CPP14Parser$ExpressionContext -> balance_=b
118 class reader.CPP14Parser$AssignmentexpressionContext ->
      balance_=b
119 class reader.CPP14Parser$LogicalorexpressionContext ->
      balance_
120 class reader.CPP14Parser$LogicalandexpressionContext ->
      balance_
121 class reader.CPP14Parser$InclusiveorexpressionContext ->
      balance_
122 class reader.CPP14Parser$ExclusiveorexpressionContext ->
      balance_
123 class reader.CPP14Parser$AndexpressionContext -> balance_
124 class reader.CPP14Parser$EqualityexpressionContext ->
      balance_
125 class reader.CPP14Parser$RelationalexpressionContext ->
      balance_
126 class reader.CPP14Parser$ShiftexpressionContext -> balance_
127 class reader.CPP14Parser$AdditiveexpressionContext ->
      balance_
128 class reader.CPP14Parser$MultiplicativeexpressionContext ->
      balance_
129 class reader.CPP14Parser$PmexpressionContext -> balance_
130 class reader.CPP14Parser$CastexpressionContext -> balance_
131 class reader.CPP14Parser$UnaryexpressionContext -> balance_
132 class reader.CPP14Parser$PostfixexpressionContext -> balance_
133 class reader.CPP14Parser$PrimaryexpressionContext -> balance_
134 class reader.CPP14Parser$IdexpressionContext -> balance_
135 class reader.CPP14Parser$UnqualifiedidContext -> balance_
136 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> balance_
137 class reader.CPP14Parser$AssignmentoperatorContext -> =
138 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> =
139 class reader.CPP14Parser$InitializerclauseContext -> b
140 class reader.CPP14Parser$AssignmentexpressionContext -> b
141 class reader.CPP14Parser$ConditionalexpressionContext -> b
142 class reader.CPP14Parser$LogicalorexpressionContext -> b
143 class reader.CPP14Parser$LogicalandexpressionContext -> b
144 class reader.CPP14Parser$InclusiveorexpressionContext -> b
145 class reader.CPP14Parser$ExclusiveorexpressionContext -> b
146 class reader.CPP14Parser$AndexpressionContext -> b
147 class reader.CPP14Parser$EqualityexpressionContext -> b
```

```
148 class reader.CPP14Parser$RelationalexpressionContext -> b
149 class reader.CPP14Parser$ShiftexpressionContext -> b
150 class reader.CPP14Parser$AdditiveexpressionContext -> b
151 class reader.CPP14Parser$MultiplicativeexpressionContext -> b
152 class reader.CPP14Parser$PmexpressionContext -> b
153 class reader.CPP14Parser$CastexpressionContext -> b
154 class reader.CPP14Parser$UnaryexpressionContext -> b
155 class reader.CPP14Parser$PostfixexpressionContext -> b
156 class reader.CPP14Parser$PrimaryexpressionContext -> b
157 class reader.CPP14Parser$IdexpressionContext -> b
158 class reader.CPP14Parser$UnqualifiedidContext -> b
159 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> b
160 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> ;
161 class reader.CPP14Parser$StatementContext -> if(balance_<0){
      cout<<"error on create account!!!"<<endl;}
162 class reader.CPP14Parser$SelectionstatementContext -> if(
      balance_<0){cout<<"error on create account!!!"<<endl;}
163 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> if
164 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> (
165 class reader.CPP14Parser$ConditionContext -> balance_<0
166 class reader.CPP14Parser$ExpressionContext -> balance_<0
167 class reader.CPP14Parser$AssignmentexpressionContext ->
      balance_<0
168 class reader.CPP14Parser$ConditionalexpressionContext ->
      balance_<0
169 class reader.CPP14Parser$LogicalorexpressionContext ->
      balance_<0
170 class reader.CPP14Parser$LogicalandexpressionContext ->
      balance_<0
171 class reader.CPP14Parser$InclusiveorexpressionContext ->
      balance_<0
172 class reader.CPP14Parser$ExclusiveorexpressionContext ->
      balance_<0
173 class reader.CPP14Parser$AndexpressionContext -> balance_<0
174 class reader.CPP14Parser$EqualityexpressionContext ->
      balance_<0
175 class reader.CPP14Parser$RelationalexpressionContext ->
      balance_<0
176 class reader.CPP14Parser$RelationalexpressionContext ->
      balance_
```

```
177 class reader.CPP14Parser$ShiftexpressionContext -> balance_
178 class reader.CPP14Parser$AdditiveexpressionContext ->
      balance_
179 class reader.CPP14Parser$MultiplicativeexpressionContext ->
      balance_
180 class reader.CPP14Parser$PmexpressionContext -> balance_
181 class reader.CPP14Parser$CastexpressionContext -> balance_
182 class reader.CPP14Parser$UnaryexpressionContext -> balance_
183 class reader.CPP14Parser$PostfixexpressionContext -> balance_
184 class reader.CPP14Parser$PrimaryexpressionContext -> balance_
185 class reader.CPP14Parser$IdexpressionContext -> balance_
186 class reader.CPP14Parser$UnqualifiedidContext -> balance_
187 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> balance_
188 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> <
189 class reader.CPP14Parser$ShiftexpressionContext -> 0
190 class reader.CPP14Parser$AdditiveexpressionContext -> 0
191 class reader.CPP14Parser$MultiplicativeexpressionContext -> 0
192 class reader.CPP14Parser$PmexpressionContext -> 0
193 class reader.CPP14Parser$CastexpressionContext -> 0
194 class reader.CPP14Parser$UnaryexpressionContext -> 0
195 class reader.CPP14Parser$PostfixexpressionContext -> 0
196 class reader.CPP14Parser$PrimaryexpressionContext -> 0
197 class reader.CPP14Parser$LiteralContext -> 0
198 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> 0
199 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> )
200 class reader.CPP14Parser$StatementContext -> {cout<<"error on
      create account!!!"<<endl;}
201 class reader.CPP14Parser$CompoundstatementContext -> {cout<<"
      error on create account!!!"<<endl;}
202 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> {
203 class reader.CPP14Parser$StatementseqContext -> cout<<"error
      on create account!!!"<<endl;
204 class reader.CPP14Parser$StatementContext -> cout<<"error on
      create account!!!"<<endl;
205 class reader.CPP14Parser$ExpressionstatementContext -> cout<<
      "error on create account!!!"<<endl;
206 class reader.CPP14Parser$ExpressionContext -> cout<<"error on
      create account!!!"<<endl
207 class reader.CPP14Parser$AssignmentexpressionContext -> cout
      <<"error on create account!!!"<<endl
```

```
208 class reader.CPP14Parser$ConditionalexpressionContext -> cout
      <<"error on create account!!!"<<endl
209 class reader.CPP14Parser$LogicalorexpressionContext -> cout<<
      "error on create account!!!"<<endl
210 class reader.CPP14Parser$LogicalandexpressionContext -> cout
      <<"error on create account!!!"<<endl
211 class reader.CPP14Parser$InclusiveorexpressionContext -> cout
      <<"error on create account!!!"<<endl
212 class reader.CPP14Parser$ExclusiveorexpressionContext -> cout
      <<"error on create account!!!"<<endl
213 class reader.CPP14Parser$AndexpressionContext -> cout<<"error
       on create account!!!"<<endl
214 class reader.CPP14Parser$EqualityexpressionContext -> cout<<"
      error on create account!!!"<<endl
215 class reader.CPP14Parser$RelationalexpressionContext -> cout
      <<"error on create account!!!"<<endl
216 class reader.CPP14Parser$ShiftexpressionContext -> cout<<"
      error on create account!!!"<<endl
217 class reader.CPP14Parser$ShiftexpressionContext -> cout<<"
      error on create account!!!"
218 class reader.CPP14Parser$ShiftexpressionContext -> cout
219 class reader.CPP14Parser$AdditiveexpressionContext -> cout
220 class reader.CPP14Parser$MultiplicativeexpressionContext ->
      cout
221 class reader.CPP14Parser$PmexpressionContext -> cout
222 class reader.CPP14Parser$CastexpressionContext -> cout
223 class reader.CPP14Parser$UnaryexpressionContext -> cout
224 class reader.CPP14Parser$PostfixexpressionContext -> cout
225 class reader.CPP14Parser$PrimaryexpressionContext -> cout
226 class reader.CPP14Parser$IdexpressionContext -> cout
227 class reader.CPP14Parser$UnqualifiedidContext -> cout
228 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> cout
229 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> <<
230 class reader.CPP14Parser$AdditiveexpressionContext -> "error
      on create account!!!"
231 class reader.CPP14Parser$MultiplicativeexpressionContext -> "
      error on create account!!!"
232 class reader.CPP14Parser$PmexpressionContext -> "error on
      create account!!!"
233 class reader.CPP14Parser$CastexpressionContext -> "error on
```

```
    create account!!!"
234 class reader.CPP14Parser$UnaryexpressionContext -> "error on
    create account!!!"
235 class reader.CPP14Parser$PostfixexpressionContext -> "error
    on create account!!!"
236 class reader.CPP14Parser$PrimaryexpressionContext -> "error
    on create account!!!"
237 class reader.CPP14Parser$LiteralContext -> "error on create
    account!!!"
238 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> "error on
     create account!!!"
239 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> <<
240 class reader.CPP14Parser$AdditiveexpressionContext -> endl
241 class reader.CPP14Parser$MultiplicativeexpressionContext ->
    endl
242 class reader.CPP14Parser$PmexpressionContext -> endl
243 class reader.CPP14Parser$CastexpressionContext -> endl
244 class reader.CPP14Parser$UnaryexpressionContext -> endl
245 class reader.CPP14Parser$PostfixexpressionContext -> endl
246 class reader.CPP14Parser$PrimaryexpressionContext -> endl
247 class reader.CPP14Parser$IdexpressionContext -> endl
248 class reader.CPP14Parser$UnqualifiedidContext -> endl
249 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> endl
250 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> ;
251 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> }
252 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> }
253 class reader.CPP14Parser$DeclarationContext -> voidAccount::
    deposit(floatvalue){balance_=balance_+value;}
254 class reader.CPP14Parser$FunctiondefinitionContext ->
    voidAccount::deposit(floatvalue){balance_=balance_+value;}
255 class reader.CPP14Parser$DeclspecifierseqContext -> void
256 class reader.CPP14Parser$DeclspecifierContext -> void
257 class reader.CPP14Parser$TypespecifierContext -> void
258 class reader.CPP14Parser$TrailingtypespecifierContext -> void
259 class reader.CPP14Parser$SimpletypespecifierContext -> void
260 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> void
261 class reader.CPP14Parser$DeclaratorContext -> Account::
    deposit(floatvalue)
262 class reader.CPP14Parser$PtrdeclaratorContext -> Account::
    deposit(floatvalue)
```

```
263 class reader.CPP14Parser$NoptrdeclaratorContext -> Account::
      deposit(floatvalue)
264 class reader.CPP14Parser$NoptrdeclaratorContext -> Account::
      deposit
265 class reader.CPP14Parser$DeclaratoridContext -> Account::
      deposit
266 class reader.CPP14Parser$IdexpressionContext -> Account::
      deposit
267 class reader.CPP14Parser$QualifiedidContext -> Account::
      deposit
268 class reader.CPP14Parser$NestednamespecifierContext ->
      Account::
269 class reader.CPP14Parser$ThetypenameContext -> Account
270 class reader.CPP14Parser$ClassnameContext -> Account
271 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> Account
272 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> ::
273 class reader.CPP14Parser$UnqualifiedidContext -> deposit
274 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> deposit
275 class reader.CPP14Parser$ParametersandqualifiersContext -> (
      floatvalue)
276 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> (
277 class reader.CPP14Parser$ParameterdeclarationclauseContext ->
       floatvalue
278 class reader.CPP14Parser$ParameterdeclarationlistContext ->
      floatvalue
279 class reader.CPP14Parser$ParameterdeclarationContext ->
      floatvalue
280 class reader.CPP14Parser$DeclspecifierseqContext -> float
281 class reader.CPP14Parser$DeclspecifierContext -> float
282 class reader.CPP14Parser$TypespecifierContext -> float
283 class reader.CPP14Parser$TrailingtypespecifierContext ->
      float
284 class reader.CPP14Parser$SimpletypespecifierContext -> float
285 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> float
286 class reader.CPP14Parser$DeclaratorContext -> value
287 class reader.CPP14Parser$PtrdeclaratorContext -> value
288 class reader.CPP14Parser$NoptrdeclaratorContext -> value
289 class reader.CPP14Parser$DeclaratoridContext -> value
290 class reader.CPP14Parser$IdexpressionContext -> value
291 class reader.CPP14Parser$UnqualifiedidContext -> value
```

```
292 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> value
293 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> )
294 class reader.CPP14Parser$FunctionbodyContext -> {balance_=
      balance_+value;}
295 class reader.CPP14Parser$CompoundstatementContext -> {
      balance_=balance_+value;}
296 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> {
297 class reader.CPP14Parser$StatementseqContext -> balance_=
      balance_+value;
298 class reader.CPP14Parser$StatementContext -> balance_=
      balance_+value;
299 class reader.CPP14Parser$ExpressionstatementContext ->
      balance_=balance_+value;
300 class reader.CPP14Parser$ExpressionContext -> balance_=
      balance_+value
301 class reader.CPP14Parser$AssignmentexpressionContext ->
      balance_=balance_+value
302 class reader.CPP14Parser$LogicalorexpressionContext ->
      balance_
303 class reader.CPP14Parser$LogicalandexpressionContext ->
      balance_
304 class reader.CPP14Parser$InclusiveorexpressionContext ->
      balance_
305 class reader.CPP14Parser$ExclusiveorexpressionContext ->
      balance_
306 class reader.CPP14Parser$AndexpressionContext -> balance_
307 class reader.CPP14Parser$EqualityexpressionContext ->
      balance_
308 class reader.CPP14Parser$RelationalexpressionContext ->
      balance_
309 class reader.CPP14Parser$ShiftexpressionContext -> balance_
310 class reader.CPP14Parser$AdditiveexpressionContext ->
      balance_
311 class reader.CPP14Parser$MultiplicativeexpressionContext ->
      balance_
312 class reader.CPP14Parser$PmexpressionContext -> balance_
313 class reader.CPP14Parser$CastexpressionContext -> balance_
314 class reader.CPP14Parser$UnaryexpressionContext -> balance_
315 class reader.CPP14Parser$PostfixexpressionContext -> balance_
316 class reader.CPP14Parser$PrimaryexpressionContext -> balance_
```

```
317 class reader.CPP14Parser$IdexpressionContext -> balance_
318 class reader.CPP14Parser$UnqualifiedidContext -> balance_
319 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> balance_
320 class reader.CPP14Parser$AssignmentoperatorContext -> =
321 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> =
322 class reader.CPP14Parser$InitializerclauseContext -> balance_
      +value
323 class reader.CPP14Parser$AssignmentexpressionContext ->
      balance_+value
324 class reader.CPP14Parser$ConditionalexpressionContext ->
      balance_+value
325 class reader.CPP14Parser$LogicalorexpressionContext ->
      balance_+value
326 class reader.CPP14Parser$LogicalandexpressionContext ->
      balance_+value
327 class reader.CPP14Parser$InclusiveorexpressionContext ->
      balance_+value
328 class reader.CPP14Parser$ExclusiveorexpressionContext ->
      balance_+value
329 class reader.CPP14Parser$AndexpressionContext -> balance_+
      value
330 class reader.CPP14Parser$EqualityexpressionContext ->
      balance_+value
331 class reader.CPP14Parser$RelationalexpressionContext ->
      balance_+value
332 class reader.CPP14Parser$ShiftexpressionContext -> balance_+
      value
333 class reader.CPP14Parser$AdditiveexpressionContext ->
      balance_+value
334 class reader.CPP14Parser$AdditiveexpressionContext ->
      balance_
335 class reader.CPP14Parser$MultiplicativeexpressionContext ->
      balance_
336 class reader.CPP14Parser$PmexpressionContext -> balance_
337 class reader.CPP14Parser$CastexpressionContext -> balance_
338 class reader.CPP14Parser$UnaryexpressionContext -> balance_
339 class reader.CPP14Parser$PostfixexpressionContext -> balance_
340 class reader.CPP14Parser$PrimaryexpressionContext -> balance_
341 class reader.CPP14Parser$IdexpressionContext -> balance_
342 class reader.CPP14Parser$UnqualifiedidContext -> balance_
```

```
343 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> balance_
344 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> +
345 class reader.CPP14Parser$MultiplicativeexpressionContext ->
      value
346 class reader.CPP14Parser$PmexpressionContext -> value
347 class reader.CPP14Parser$CastexpressionContext -> value
348 class reader.CPP14Parser$UnaryexpressionContext -> value
349 class reader.CPP14Parser$PostfixexpressionContext -> value
350 class reader.CPP14Parser$PrimaryexpressionContext -> value
351 class reader.CPP14Parser$IdexpressionContext -> value
352 class reader.CPP14Parser$UnqualifiedidContext -> value
353 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> value
354 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> ;
355 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> }
356 class org.antlr.v4.runtime.tree.TerminalNodeImpl -> <EOF>
```

APÊNDICE B - PASSEIOS UTILIZADOS PARA O PROCESSO DE GERAÇÃO DE CASOS DE TESTES EXECUTÁVEIS.

B.1 Problema conta

B.1.1 ACC2_2

B.1.1.1 Caso de teste 1

"*__main__1, __user__newuser__22__2, __account__newaccount__3, __bool__validuser__newuser__verifyage__4, __if__uvalido__5, __return__0__9, final*"

B.1.1.2 Caso de teste 2

" *__main__1, __user__newuser__22__2, __account__newaccount__3, __bool__validuser__newuser__verifyage__4, __if__uvalido__5, __newaccount__initialize__newuser__100__6, __initialize__21, __user__user__22, __balance__b__23, __if__balance__0__24, __newaccount__initialize__newuser__100__6, __initialize__21, __user__user__22, __balance__b__23, __if__balance__0__24, __error__on__create__account__25, __newaccount__initialize__newuser__100__6, __newaccount__deposit__10__7, __deposit__28, balance__balance__value__29, __newaccount__deposit__10__7, __return__0__9, final* "

B.1.2 ACC__3__2

B.1.2.1 Caso de teste 22

"*__main__35, usuario__novou__22__36, usuario__71, idade__idade__72, usuario__novou__22__36, conta__novaconta__37, bool__uvalido__novou__verificaidade__38, if__39, return__43, final*"

B.1.2.2 Caso de teste 13

"*__main__35, usuario__novou__22__36, conta__novaconta__37, verificaidade__74, if__75, bool__uvalido__novou__verificaidade__38, return__76, bool__uvalido__novou__verificaidade__38, if__39, novaconta__inicializa__novou__100__40, inicializa__54, usuario__usuario__55, saldo__s__56, if__57, erro__na__cria__o__da__conta__58, novaconta__inicializa__novou__100__40, novaconta__deposita__10__41, deposita__61, saldo__saldo__valor__62, novaconta__deposita__10__41, return__43, final* "

B.2 Problema prquadtree

B.2.1 PQT1_2

B.2.1.1 Caso de teste 1

"*__prquadtree__1, __maxdepth__2, __root__3, __insert__8,__node_nodetemp_getleafat_point_getx_point_gety_9, __nodetemp_addvalue_point_10, __if_nodetemp_getdepth_maxdepth_11, __list_node_list_14, __list_push_back_nodetemp_15, __while_16, __nodetemp_list_back_17, __list_pop_back_18, __if_nodetemp_getlen_1_nodetemp_getdepth_maxdepth_19, __nodetemp_subdivide_20, __for_21, __list_push_back_nodetemp_getchild_e_22, __getleafat_48, __node_nodetemp_root_49, __while_50, __for_51, __if_nodetemp_getchild_e_isinregion_x_y_52, __for_51, __if_nodetemp_getchild_e_isinregion_x_y_52, __nodetemp_nodetemp_getchild_e_53, __break_54, __return_nodetemp_58, final*"

B.2.1.2 Caso de teste 2

"*__prquadtree__1, __maxdepth__2, __root__3, __insert__8,__node_nodetemp_getleafat_point_getx_point_gety_9, __nodetemp_addvalue_point_10, __if_nodetemp_getdepth_maxdepth_11, __list_node_list_14, __list_push_back_nodetemp_15, __while_16, __nodetemp_list_back_17, __list_pop_back_18, __if_nodetemp_getlen_1_nodetemp_getdepth_maxdepth_19, __while_16, __nodetemp_list_back_17, __list_pop_back_18, __if_nodetemp_getlen_1_nodetemp_getdepth_maxdepth_19, __nodetemp_subdivide_20, __for_21, __list_push_back_nodetemp_getchild_e_22, __getleafat_48, __node_nodetemp_root_49, __while_50, __for_51, __if_nodetemp_getchild_e_isinregion_x_y_52, __nodetemp_nodetemp_getchild_e_53, final*"

B.2.2 PQT2_2

B.2.2.1 Caso de teste 1

"*__prquadtree__1, __maxdepth__2, __root__3, __insert__8, __node_nodetemp_getleafat_point_getx_point_gety_9, __nodetemp_addvalue_point_10, __addvalue_178, __assert_isleaf_179, __point_pointtemp_new_point_len_1_180, __for_181, __pointtemp_i_val_i_182, __pointtemp_len_point_184, __if_len_0_185, __val_pointtemp_188, __len_189, __nodetemp_addvalue_point_10, __if_nodetemp_getdepth_maxdepth_11, __getdepth_156, __return_depth_157, __if_nodetemp_getlen_1_nodetemp_getdepth_maxdepth_19, __getdepth_156, __return_depth_157, __if_nodetemp_getlen_1_nodetemp_getdepth_maxdepth_19, __nodetemp_subdi-*

vide_20, _subdivide_76, _node_newchild_4_77, _for_78, _if_len_85, _isleaf_-false_88, _for_89, _nodetemp_subdivide_20, _for_21, _list_push_back_node-temp_getchild_e_22, _getchild_159, _assert_isleaf_160, _return_child_e_161, _nodetemp_nodetemp_getchild_e_53, _break_54, _return_nodetemp_58, final "

B.2.2.2 Caso de teste 2

"_prquadtree_1, _maxdepth_2, _root_3, _insert_8, _node_nodetemp_getlea-fat_point_getx_point_gety_9, _nodetemp_addvalue_point_10, _if_nodetemp_-getdepth_maxdepth_11, _getdepth_156, _return_depth_157, _if_nodetemp_-getdepth_maxdepth_11, _list_node_list_14, _list_push_back_nodetemp_15, _while_16, _nodetemp_list_back_17, _list_pop_back_18, _if_nodetemp_ge-tlen_1_nodetemp_getdepth_maxdepth_19, _getlen_191, _assert_isleaf_192, _return_len_193, _if_nodetemp_getlen_1_nodetemp_getdepth_maxdepth_19, _nodetemp_subdivide_20, _subdivide_76, _node_newchild_4_77, _for_78, _for_79, _if_newchild_e_isinregion_val_i_getx_val_i_gety_80, _for_79, _-if_newchild_e_isinregion_val_i_getx_val_i_gety_80, _newchild_e_addvalue_-val_i_81, _for_79, _for_78, _if_len_85, _delete_val_86, _isleaf_false_88, _-for_89, _child_e_newchild_e_90, _for_89, _nodetemp_subdivide_20, _for_21, _list_push_back_nodetemp_getchild_e_22, _getleafat_48, _node_nodetemp_-root_49, _while_50, _for_51, _if_nodetemp_getchild_e_isinregion_x_y_52, _nodetemp_nodetemp_getchild_e_53, final "

B.2.3 PQT3_2

B.2.3.1 Caso de teste 1

"_prquadtree_1, _maxdepth_2, _root_3,_prquadtree_5, _delete_root_6, _in-sert_8, _node_nodetemp_getleafat_point_getx_point_gety_9, _gety_249, _re-turn_y_250, _node_nodetemp_getleafat_point_getx_point_gety_9, _getx_246, _return_x_247, _node_nodetemp_getleafat_point_getx_point_gety_9, _node_-123, _left_124, _width_125, _down_126, _height_127, _isleaf_128, _depth_-129, _len_130, _node_nodetemp_getleafat_point_getx_point_gety_9, _node-temp_addvalue_point_10, _addvalue_178, _assert_isleaf_179, _point_point-temp_new_point_len_1_180, _point_241, _x_x_242, _y_y_243, value_va-lue_244, _point_pointtemp_new_point_len_1_180, _for_181, _pointtemp_-len_point_184, _if_len_0_185, _val_pointtemp_188, _len_189, _nodetemp_-addvalue_point_10, _if_nodetemp_getdepth_maxdepth_11, _getdepth_156, _re-turn_depth_157, _if_nodetemp_getdepth_maxdepth_11, _list_node_list_14, _-

list_push_back_nodetemp_15, _while_16, _nodetemp_list_back_17, _list_pop_-back_18, _if_nodetemp_getlen_1_nodetemp_getdepth_maxdepth_19, _getlen_-191, _assert_isleaf_192, _return_len_193, _if_nodetemp_getlen_1_nodetemp_-getdepth_maxdepth_19, _nodetemp_subdivide_20, _for_21, _list_push_back_-nodetemp_getchild_e_22, _getchild_159, _assert_isleaf_160, _return_child_e_-161, _nodetemp_nodetemp_getchild_e_53, _break_54, _return_nodetemp_58, fi-nal"

B.2.3.2 Caso de teste 2

"*_prquadtree_1, _maxdepth_2, _root_3, _insert_8, _node_nodetemp_getlea-fat_point_getx_point_gety_9, _nodetemp_addvalue_point_10, _if_nodetemp_-getdepth_maxdepth_11, _getdepth_156, _return_depth_157, _if_nodetemp_-getdepth_maxdepth_11, _list_node_list_14, _list_push_back_nodetemp_15, _while_16, _nodetemp_list_back_17, _list_pop_back_18, _if_nodetemp_ge-tlen_1_nodetemp_getdepth_maxdepth_19, _getlen_191, _assert_isleaf_192, _return_len_193, _if_nodetemp_getlen_1_nodetemp_getdepth_maxdepth_19, _nodetemp_subdivide_20, _subdivide_76, _node_newchild_4_77, _for_78, _for_79, _if_newchild_e_isinregion_val_i_getx_val_i_gety_80, _for_79, _-if_newchild_e_isinregion_val_i_getx_val_i_gety_80, _newchild_e_addvalue_-val_i_81, _for_79, _for_78, _if_len_85, _delete_val_86, _isleaf_false_88, _-for_89, _child_e_newchild_e_90, _for_89, _nodetemp_subdivide_20, _for_21, _list_push_back_nodetemp_getchild_e_22, _getleafat_48, _node_nodetemp_-root_49, _while_50, _for_51, _if_nodetemp_getchild_e_isinregion_x_y_52, _for_51, _if_nodetemp_getchild_e_isinregion_x_y_52, _nodetemp_nodetemp_-getchild_e_53, final* "

APÊNDICE C - PUBLICAÇÕES

O seguinte artigo, relacionado a essa dissertação de mestrado, foi aceito e publicado em conferência nacional:

SALES, C. P.; SANTIAGO JÚNIOR, V. A. Investigating multi and many-objective metaheuristics to support software integration testing. In: Proceedings of the 5th Brazilian Symposium on Systematic and Automated Software Testing. New York, NY, USA: Association for Computing Machinery, 2020. (SAST 20), p. 1–10. ISBN 9781450387552 (SALES; SANTIAGO JÚNIOR, 2020).

O seguinte artigo, relacionado a essa dissertação de mestrado, foi aceito em uma conferência internacional e está em processo de publicação.

SANTIAGO JÚNIOR, V. A.; SALES, C. P. Metaheuristics and hyper-heuristics based on evolutionary algorithms for software integration testing. In: Proceedings of the 5th International Joint Conference on Advances in Computational Intelligence. [S.l.: s.n.], 2021. p. 1–22. Artigo em processo de publicação (SANTIAGO JÚNIOR; SALES, 2021).