RUI ANGELO MATNEI FILHO

GERANDO DADOS PARA O TESTE DE MUTAÇÃO DE LINHA DE PRODUTO DE SOFTWARE COM ALGORITMOS DE OTIMIZAÇÃO MULTIOBJETIVO

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientadora: Profa. Dra. Silvia Regina Vergilio

CURITIBA

2015

RUI ANGELO MATNEI FILHO

GERANDO DADOS PARA O TESTE DE MUTAÇÃO DE LINHA DE PRODUTO DE SOFTWARE COM ALGORITMOS DE OTIMIZAÇÃO MULTIOBJETIVO

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientadora: Profa. Dra. Silvia Regina Vergilio

CURITIBA

2015

M433g Matnei Filho, Rui Angelo
Gerando dados para o teste de mutação de linha de produto de software com algoritmos de otimização multiobjetivo/ Rui Angelo Matnei Filho. – Curitiba, 2015.
95 f. : il. color. ; 30 cm.

Dissertação - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-graduação em Informática, 2015.

Orientador: Silvia Regina Vergilio .
Bibliografia: p. 88-95.

1. Algoritmo - Otimização. 2. Genética- Software. 3. Engenharia de software - Testes. 4. Software - Desenvolvimento. I. Universidade Federal do Paraná. II.Vergilio, Silvia Regina. III. Título.

CDD: 006.3823

Ministério da Educação
Universidade Federal do Paraná
Programa de Pós-Graduação em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa de Dissertação de Mestrado em Informática, do aluno Rui Angelo Matnei Filho, avaliamos o trabalho intitulado, *"GERANDO DADOS PARA O TESTE DE MUTAÇÃO DE LINHA DE PRODUTO DE SOFTWARE COM ALGORITMOS DE OTIMIZAÇÃO MULTIOBJETIVO"*, cuja defesa foi realizada no dia 16 de abril de 2015, às 15:00 horas, no Departamento de Informática do Setor de Ciências Exatas da Universidade Federal do Paraná. Após a avaliação, decidimos pela:

(X) **aprovação** do candidato. ( ) **reprovação** do candidato.

Curitiba, 16 de abril de 2015.

Profª. Dra. Silvia Regina Vergilio
PPGInf/UFPR – Orientadora

Profª. Dra. Sandra Fabbri
UFSCar - Membro Externo

Prof. Dr. Marcos Didonet Del Fabro
PPGInf/UFPR – Membro Interno

MINISTÉRIO DA EDUCAÇÃO · UNIVERSIDADE FEDERAL DO PARANÁ · Setor de Ciências Exatas · Coordenação de Pós-Graduação em Informática

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

AG Algoritmo Genético

AGM Arcade Game Maker

ED Euclidean Distance

ER Error Ratio

FM Feature Model

FaMa FeAture Model Analyser

FMTS Feature Mutation Test Suite

HV Hypervolume

IBEA Indicator-Based Evolutionary Algorithm

LPS Linha de Produto de Software

MOEA Multi-objective Evolutionary Algorithm

MOGA Multi-objective Genetic Algorithm

NPGA Niched-Pareto Genetic Algorithm

NSGA-II Nondominated Sorting Genetic Algorithm II

OVM Orthogonal Variability Model

SBSE Search-Based Software Engineering

SPEA2 Strength Pareto Evolutionary Algorithm 2

RESUMO

O teste de mutação tem sido recentemente aplicado no teste de Linha de Produto de Software. A ideia consiste em selecionar produtos para o teste de acordo com operadores de mutação que representam possíveis defeitos em um diagrama de características (*feature model* - FM). Esses operadores associados ao escore de mutação são então usados para avaliação e geração de conjuntos de casos de teste. A geração de conjuntos de casos de teste que matem todos os mutantes e que, além disso, satisfaçam outros requisitos para o teste de software, tais como o menor número possível de produtos, é uma tarefa complexa. A fim de resolver esse problema, este trabalho propõe uma abordagem de otimização multiobjetivo que inclui uma representação para o problema, operadores de busca e uma função de avaliação que inclui três objetivos relacionados ao número de casos de teste, número de mutantes mortos e número de pares de características cobertos. A abordagem foi implementada com três algoritmos evolutivos multiobjetivos: NSGA-II, SPEA2 e IBEA. Foram realizados dois experimentos: um experimento (2M) com dois objetivos considerando tamanho do conjunto e número de mutantes mortos; e um experimento (3MP) com três objetivos considerando tamanho do conjunto, número de mutantes mortos e cobertura de pares de características (*pairwise testing*). A avaliação realizada analisou as soluções obtidas e comparou os algoritmos. De maneira geral todos obtiveram um bom desempenho com destaque para o tempo de execução do IBEA, o número de soluções do NSGA-II no experimento 2M e a unanimidade de melhores soluções obtidas pelo algoritmo SPEA2 no experimento 3MP. Uma vantagem dessa abordagem é oferecer ao testador um conjunto de boas soluções, com um número reduzido de produtos e altos valores de cobertura.

ABSTRACT

The mutation test has recently been applied to the Software Product Line testing. The idea is to select products for testing according to mutation operators that represent possible faults in the Feature Model - FM. These operators associated with the mutation score are then used for evaluation and generation of test case sets. The generation of test sets to kill all mutants, and also meet other testing requirements with the minimum possible number of products, is a complex task. To solve this problem, this work proposes a multiobjective optimization approach that includes a representation to the problem, search operators and a fitness function with three objectives, related to the number of test cases, number of mutants killed, and number of covered feature pairs. The approach was implemented with three multiobjective evolutionary algorithms: NSGA-II, SPEA2 and IBEA. Two experiments were conducted: one experiment (2M) with two objectives: set size and number of killed mutants; and other one (3MP) with three objectives: set size, number of killed mutants and feature coverage (pairwise testing). The evaluation analyzes the solutions obtained and compares the algorithms. In general all algorithms performed well. Among the main results we can mention: the IBEA runtime and the number of solutions obtained by NSGA-II in the experiment 2M and the best solutions obtained by SPEA2 in the experiment 3MP. An advantage of this approach is to offer the tester a set of good solutions, with a small number of products and high coverage values.

CAPÍTULO 1

INTRODUÇÃO

Uma Linha de Produto de Software (LPS) corresponde a um conjunto de produtos que compartilham características comuns e que satisfazem as necessidades de um determinado segmento de mercado. Engenharia de LPS consiste em uma abordagem para o desenvolvimento de *software* que utiliza plataformas e customização em massa [43, 55]. Essa abordagem tem impacto em negócios, organização, tecnologia e é uma forma comprovada de desenvolver uma vasta gama de produtos de *software* de maneira rápida e com baixos custos, enquanto fornece um *software* de alta qualidade [55].

No contexto de engenharia de LPS uma característica (*feature*) consiste em uma funcionalidade ou atributo do sistema visível ao usuário final, sendo que cada LPS possui um conjunto de características. Cada característica de uma LPS pode, ou não, ser selecionada de maneira a formar um determinado produto [43]. As características permitem ainda diferenciar os produtos entre si [55].

Características podem ser expressas na forma de modelos de características (*Feature Model* - FM), o que permite uma decomposição hierárquica de características que produz uma árvore de características. Uma árvore de características estrutura o conjunto de características de um sistema [43].

Com a crescente utilização de LPS nas indústrias há uma demanda por técnicas específicas de teste para esse contexto. Verificar se os produtos gerados através da LPS estão de acordo com o desejado constitui uma importante tarefa dentro da atividade de teste. Para essa tarefa, um produto pode ser considerado como um caso de teste e consiste na derivação customizada das características presentes em uma LPS [10].

Idealmente todos os produtos, ou seja, todas as combinações possíveis de características de uma LPS, deveriam ser testados. Entretanto, assumindo que um diagrama de características possui n características e que cada produto demora O(m) para ser

testado, no pior caso, um teste completo do diagrama de características em questão demoraria O(2^n x m). Isso é impraticável tanto em termos de recursos para gerar todos os testes necessários quanto em termos de tempo requerido para executá-los [15].

Nesse contexto, assim como no teste de *software* tradicional, critérios de teste são utilizados para selecionar e avaliar casos de teste de forma a aumentar as possibilidades de revelar defeitos e estabelecer um nível elevado de confiança na correção do produto [46]. Critérios como o teste *pairwise* [40] e o teste de mutação [17, 23] foram propostos com o intuito de selecionar conjuntos de casos de teste para o teste de LPS a partir do modelo de características. Diferentes abordagens utilizam critérios baseados no teste de mutação. Henard et al. [23] propõem que em uma expressão booleana, que nesse caso representa um FM em teste, sejam inseridas diferentes alterações com a finalidade de gerar FMs mutantes. Dois operadores de mutação foram propostos associados ao teste de similaridade. O objetivo é gerar dados de teste o mais dissimilares possível. Ferreira [17], por outro lado, propõe uma abordagem mais completa para a geração de mutantes. Primeiramente o autor faz um levantamento das diferentes classes de defeitos que podem ocorrer em um FM e, posteriormente, propõe dezesseis operadores de mutação capazes de revelar tais defeitos. Além disso, para auxiliar e automatizar sua proposta, o autor desenvolveu a ferramenta FMTS, sigla do inglês *Feature Mutation Test Suite*, que concentra funções como a geração dos mutantes, e cálculo do escore de mutação para um conjunto de teste.

Diferentes conjuntos de produtos que satisfazem os critérios expostos podem existir. Portanto, o desenvolvimento de estratégias para a seleção dos melhores conjuntos de produtos (casos de teste) que garantam a qualidade da LPS torna-se um objetivo relevante [15]. Isto porque a seleção dos melhores conjuntos é um problema de pesquisa que pode envolver diferentes fatores tais como tamanho mínimo e maior cobertura em determinados critérios. Dada a natureza complexa desse problema, ele pode ser enquadrado em uma área denominada Engenharia de Software Baseada em Busca [21], tradução do inglês *Search-Based Software Engineering* (SBSE), que consiste na utilização de algoritmos de otimização em problemas complexos da Engenharia de Software.

Algoritmos baseados em busca, como os algoritmos genéticos, são capazes de, em um

grande espaço de busca, encontrar uma solução que resolva o problema e satisfaça algumas restrições. Esses algoritmos vem sendo utilizados no contexto da engenharia de LPS, principalmente nos últimos dois anos, com os objetivos de configuração de produtos, evolução e adaptação do FM e também de seleção de produtos para teste [18, 20]. Trabalhos que abordam o teste de LPS possuem como objetivo minimização [58], priorização de casos de teste [59] e geração de produtos (casos de teste) [15, 24] considerando diferentes fatores como: número de casos de teste, cobertura *pairwise*, número de defeitos revelados e outros relativos a custo. O critério de teste de mutação é usado apenas no trabalho de Henard et. al. [25] o qual implementa o Algoritmo Evolutivo (1+1) e considera os operadores definidos em [23].

A maior limitação desses trabalhos é que eles utilizam funções de agregação e algoritmos mono-objetivos para lidar com o problema que é de fato multiobjetivo. Nesse sentido, o uso de algoritmos multiobjetivos é mais adequado. Tais algoritmos usam o conceito de dominância de Pareto [41] para achar um conjunto de soluções que melhor representa o *trade-off* entre os objetivos. No trabalho de Lopez-Herrejon et al. [34] os autores utilizam um algoritmo multiobjetivo para esse problema considerando dois objetivos: número de produtos e cobertura *pairwise*. Entretanto, não foram encontradas abordagens multiobjetivo que enfoquem o teste de mutação.

1.1 Justificativa

Diante do contexto apresentado, este trabalho é justificado através das seguintes motivações:

- O critério análise de mutantes é um dos mais eficazes em termos do número de defeitos revelados, entretanto um dos mais caros em termos de número de dados de teste necessários. Portanto, gerar automaticamente dados de teste para satisfazer esse critério ajuda a reduzir o custo e esforço de teste;
- Os conjuntos de dados gerados devem satisfazer diferentes fatores tais como um

tamanho menor possível, altos valores de escore e de cobertura de outros critérios de teste, tais como *pairwise*;

- Observa-se que o problema de geração de dados de teste para satisfazer o teste de mutação de LPS é multiobjetivo. Entretanto, na literatura não foram encontradas abordagens que oferecem um tratamento multiobjetivo ao problema.

1.2 Objetivo

O presente trabalho tem como objetivo geral a utilização de algoritmos evolutivos multiobjetivos para gerar conjuntos de produtos para o teste de LPS visando a satisfazer o teste de mutação. Sendo assim, uma abordagem de geração de dados de teste baseada em algoritmos multiobjetivos é proposta. A ideia é obter conjuntos que satisfaçam o teste de mutação e que sejam mínimos em relação ao critérios adotados além de satisfazer outros fatores que influenciam essa escolha tal como o critério *pairwise*.

A abordagem engloba duas características principais: i) introduz uma nova representação para a população, onde um indivíduo (solução) é um conjunto de produtos, diferentemente da maioria dos trabalhos existentes, onde um indivíduo representa apenas um produto; e ii) explora o uso de diferentes objetivos, relacionados à quantidade de mutantes mortos, à cobertura *pairwise* e quantidade de produtos. Tais objetivos não foram explorados em conjunto nos trabalhos relacionados encontrados.

A implementação da abordagem é feita com três diferentes algoritmos, tradicionalmente utilizados para a geração de casos de teste: NSGA-II [28], SPEA2 [14] e IBEA [62] disponíveis no *framework* jMetal. A implementação interage com a ferramenta FMTS. Essa ferramenta implementa os operadores de mutação propostos por Ferreira [17] e utilizados neste trabalho. Esses operadores foram selecionados, pois foram elaborados de acordo com classes de defeitos que podem ocorrer em um FM, fornecendo maior probabilidade de revelar defeitos e um conjunto mais amplo que o proposto por Henard et al. [23]. O desempenho dos algoritmos implementados é comparado de acordo com indica-

dores de qualidade utilizados na área de otimização. Além disso, as soluções encontradas são analisadas com relação aos valores de *fitness* obtidos.

1.3 Organização do Trabalho

Este trabalho está dividido da seguinte maneira: o Capítulo 2 trata dos principais conceitos relacionados à LPS, abordando uma visão geral e as atividades essenciais. No Capítulo 3 são apresentados os principais conceitos relacionados ao teste, sendo abordado o teste de *software*, seus conceitos e técnicas, bem como o teste de Linha de Produto de *Software* com os critérios *pairwise* e mutação de variabilidades. O Capítulo 4 aborda os principais conceitos sobre Otimização Multiobjetivo bem como alguns dos algoritmos utilizados para esse propósito. No Capítulo 5 são apresentados os trabalhos relacionados existentes na literatura. No Capítulo 6 a abordagem proposta é introduzida e são discutidos seus principais aspectos de implementação. O Capítulo 7 apresenta resultados da avaliação conduzida, as questões de pesquisa formuladas, a organização dos experimentos e as avaliações realizadas. Por fim, o Capítulo 8 apresenta as considerações finais sobre o trabalho apresentado e mostra novas direções de pesquisa.

CAPÍTULO 2

LINHA DE PRODUTO DE SOFTWARE

Linha de Produto de *Software* (LPS) pode ser definida como um conjunto de produtos que compartilham características comuns [43]. De maneira geral, consiste em uma abordagem estratégica para o desenvolvimento de *software* que utiliza plataformas e customização em massa, sendo comumente dirigida por interesses de mercado como custo e tempo [55].

Engenharia de LPS impacta negócios, organização, tecnologia e é uma forma comprovada de desenvolver uma vasta gama de produtos de *software* de maneira rápida e com baixos custos, enquanto fornece um *software* de alta qualidade o que influencia diretamente na redução do tempo de desenvolvimento e do esforço de manutenção [43, 55].

Neste capítulo é dada uma visão geral sobre LPS e engenharia de LPS.

2.1 Visão Geral

No contexto de engenharia de LPS uma característica (*feature*) consiste em uma funcionalidade ou atributo do sistema que é visível ao usuário final [43]. Características podem ser expressas com o auxílio de um modelo de características (do inglês *Feature Models* - FM) permitindo uma decomposição hierárquica das mesmas que produz uma representação em estrutura de árvore, o diagrama de características. Uma árvore de características estrutura o conjunto de características de um sistema e deve ser capaz de representar os seguintes relacionamentos entre as características: (i) mandatórias (*mandatory*); (ii) opcionais (*optional*); (iii) alternativas (*alternative*); (iv) Relação OU (*OR*) [43].

De maneira geral, um diagrama de características é composto por quatro elementos principais: características, relações, restrições e cardinalidades. Assim a característica pode ser uma raiz (*root*), uma característica solitária (*solitary*) ou agrupada (*grouped*). Além disso, características possuem os atributos nome e domínio. Relações podem ser do tipo binária (*binary*) ou relações agrupadas (*set*) e, por fim, restrições podem ser do

tipo incluir (*depends*) ou excluir (*excludes*). Cardinalidades são aplicadas a características solitárias ou em relações agrupadas [17].

O ponto principal é que as características dos produtos que os diferenciam de outros produtos são fundamentais para a representação de variabilidade. Variabilidade é um conceito chave no contexto de engenharia de LPS. Ao invés de entender cada sistema individualmente, a engenharia de LPS olha para a linha de produto como um todo, tentando identificar as variações entre os sistemas. Essas variações são chamadas de variabilidades e devem ser definidas, representadas, exploradas, evoluídas; em outras palavras, devem ser gerenciadas em toda a engenharia de LPS [55].

Para representar as variabilidades existem alguns elementos que devem ser entendidos, eles são apresentados por [55] como:

1. Pontos de variação (*Variation point*): descrevem pontos nos quais existem diferenças nos sistemas finais (por exemplo, sistemas podem ser diferentes no que diz respeito aos sistemas operacionais que dependem);

2. Variantes (*Variant*): as diferentes possibilidades que existem para se resolver um ponto de variação são chamadas variantes;

Além do FM as características também podem ser representadas no Modelo de Variabilidades Ortogonais (do inglês *Orthogonal Variability Model - OVM*) [55]. Um OVM consiste em um modelo que relaciona as variabilidades definidas a outros diagramas de *software* como: Diagrama de Recursos, Diagrama de Casos de Uso, Diagrama de *Design*, Diagrama de Componentes e Diagramas de Teste. Além disso, provê uma visão geral das variabilidades em todos os artefatos de desenvolvimento de *software* [43].

A Figura 2.1 apresenta um exemplo de diagrama de características para a LPS *Arcade Game Maker* (AGM), proposta pelo Instituto de Engenharia de Software [53], utilizando alguns dos elementos apresentados para variabilidade. As características obrigatórias são representadas pela notação de um círculo preenchido (característica *services*) enquanto as características opcionais são representadas por um círculo não preenchido (característica *save*). Características alternativas são representadas através de um traço que une duas ou

mais características, por exemplo, a característica *rules* que possui como alternativas as opções *brickles, pong e bowling*. A cardinalidade [1..1] significa que das 3 características só é possível escolher uma. *Action* representa um ponto de variação onde *movement e collision* são suas variantes.

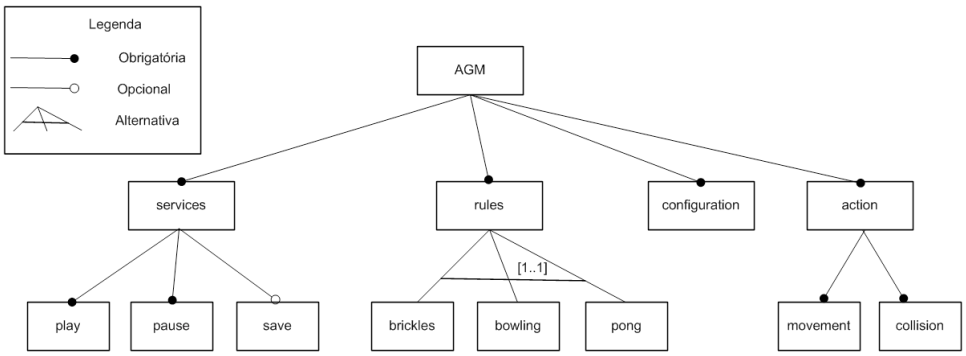

Figura 2.1: Diagrama de Características da LPS AGM (Adaptada de [53])

A Figura 2.2 apresenta um exemplo de OVM. No OVM o triângulo com a sigla VP representa um Ponto de Variação, os retângulos com a inicial V representam as Variantes para resolver tal ponto. Além disso, as linhas tracejadas com um arco representam uma escolha alternativa, ou seja, onde se pode escolher apenas uma opção. Nesse caso "Fechadura" seria um ponto de variação e "Teclado" e "Scanner Biométrico" suas variantes.

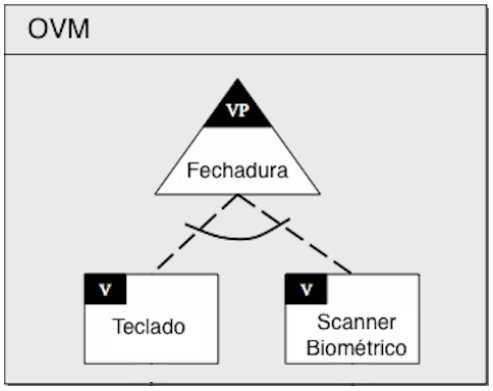

Figura 2.2: Exemplo de Diagrama de Variabilidade Ortogonal (adaptada de [43])

2.2 Atividades Essenciais

As atividades essenciais para montar uma LPS envolvem o Desenvolvimento do Núcleo de Artefatos (*Core Asset Development*) e o Desenvolvimento do Produto (*Product De-*

velopment) que utiliza os recursos do núcleo de artefatos. Ambas estão amparadas pelo Gerenciamento (*Management*) técnico-organizacional. O Desenvolvimento do Núcleo de Artefatos e o Desenvolvimento do Produto podem ocorrer de duas maneiras: novos produtos são construídos a partir do Núcleo de Artefatos ou o Núcleo de Artefatos é extraído de produtos existentes. Frequentemente, produtos e núcleo de artefatos são construídos em conjunto [53].

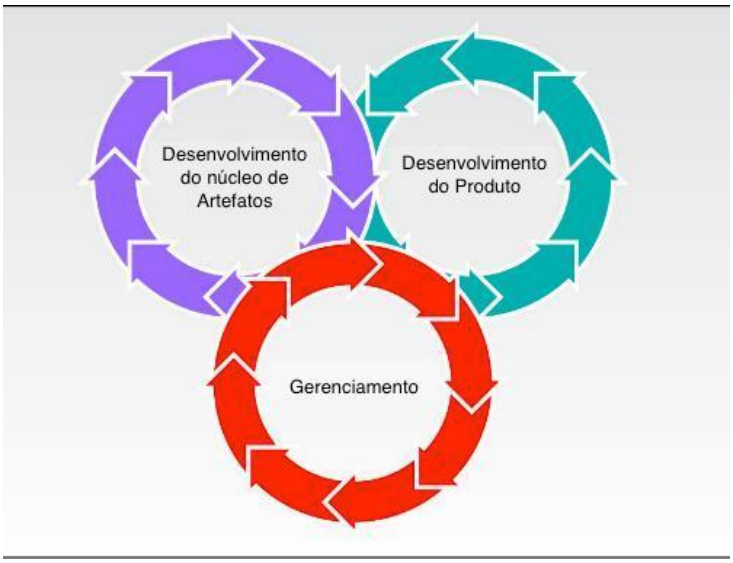

Figura 2.3: Atividades Essenciais de uma LPS (adaptada de [53])

A Figura 2.3 exibe a interação entre as atividades de uma LPS. As setas rotativas não indicam apenas que o Núcleo de Artefatos é utilizado para desenvolver produtos, mas também que revisões dos artefatos existentes ou até mesmo a criação de novos artefatos podem surgir do desenvolvimento de produtos. Além disso, a atividade de Gerenciamento assegura que atividades técnicas sejam realizadas segundo um planejamento [53].

As subseções seguintes detalham especificamente cada atividade descrita.

2.2.1 Desenvolvimento do Núcleo de Artefatos

O objetivo da atividade de Desenvolvimento do Núcleo de Artefatos é estabelecer uma base para o desenvolvimento de produtos. A atividade em questão acontece através de uma série de restrições e recursos, dentre quais destacam-se [53]:

- Restrições de Produto: são semelhanças e variações dos produtos da LPS, podem

ser: características comportamentais com as quais os produtos devem se integrar, limites computacionais e limites físicos;

- Restrições de Produção: normas específicas para o processo de desenvolvimento de *software* que devem ser levadas em consideração durante sua produção;
- Estratégia de Produção: define qual abordagem estratégica global deve ser usada para a produção de artefatos e produtos da LPS; e
- Ativos Pré-Existentes: utilização de artefatos que já existem.

Diante das entradas citadas acima as seguintes principais saídas são obtidas [53]:

- Escopo da LPS: representa quais produtos pertencem a LPS ou quais produtos a LPS pode produzir;
- Núcleo de Artefatos da LPS: constitui a base de artefatos da LPS a partir de onde os produtos da LPS são produzidos;
- Plano de Produção: apresenta como os produtos são produzidos a partir do núcleo de artefatos.

2.2.2 Desenvolvimento do Produto

O desenvolvimento do produto depende das saídas citadas na Seção 2.2.1 adicionada à descrição de cada produto. As entradas para o desenvolvimento de produtos são [53]:

- A descrição particular de um determinado produto;
- O escopo da linha de produto, que indica a viabilidade da inclusão do produto;
- O Núcleo de Artefatos do qual o produto é construído;
- Plano de produção que detalha como o núcleo de artefatos é utilizado para a construção do produto.

Os fabricantes de produto utilizam o núcleo de artefatos de acordo com o plano de produção para produzir produtos que atendem seus respectivos requerimentos. Os fabricantes ainda possuem como obrigação de fornecer o *feedback* para qualquer problema ou deficiência encontrada com o núcleo de artefatos, assim, a base do núcleo permanece viável [53].

2.2.3 Gerenciamento

A atividade de gerenciamento está dividida em duas: gerenciamento organizacional e gerenciamento técnico. O gerenciamento organizacional identifica as restrições de produção e determina a estratégia de produção. Já o gerenciamento técnico supervisiona o processo de desenvolvimento do núcleo de artefatos e a atividade de desenvolvimento de produtos, assegurando que todos os requisitos essenciais estejam envolvidos [53].

Os gerenciamentos organizacional e técnico contribuem ainda para a base do núcleo de artefatos. Para isso, disponibilizam o reúso desses artefatos de gestão, que são utilizados no desenvolvimento de produtos, em uma linha de produtos de *software* [53].

2.3 Framework FaMa

O *framework* FaMa (*Feature Model Analyser*) [16] consiste em uma ferramenta para análise automática de LPS, que é representada através de diagramas de características (FMs). A utilização dessa ferramenta evita a necessidade de se confiar inteiramente em testadores para verificar quando a saída de uma análise está ou não correta.

A ferramenta oferece várias formas de interface com o usuário. Pode-se utilizar o FaMa através de linha de comando, carregando modelos e realizando operações como busca de características, transformação de modelos e adição de configuração; através de um *web service* disponibilizado ou ainda como uma biblioteca de extensão. A extensão dos diagramas utilizados como entrada para a ferramenta é o XML [57].

A seguir são descritas as principais operações do *framework* [16]:

- Validação: valida o diagrama de características verificando se o mesmo não está vazio ou, em outras palavras, se existe pelo menos um produto;
- Produtos: gera uma lista de todos os produtos válidos para um diagrama de entrada;
- Número de Produtos: calcula o número de produtos válidos;
- Similaridade: calcula o número de aparições de uma característica em uma lista de produtos;
- Variabilidade: calcula o grau de variabilidade de um diagrama de características;
- Produto Válido: determina se um produto é válido para um diagrama;
- Configuração Válida: analisa se uma determinada configuração é válida. Uma configuração é um produto não acabado que pode precisar de mais recursos para ser um produto válido;
- Detecção de Erro: retorna um conjunto de erros de um diagrama de características;
- Explicações de Erro: quando um diagrama possui erros, essa operação procura explicações (relações) para os erros;
- Explicação de Produto Inválido: provê a possibilidade de reparar um produto inválido de um determinado diagrama de características;
- Principais características: verifica quais características estão presentes em todos os produtos;
- Características Variantes: retorna o conjunto de características que não estão presentes em todos os produtos.

Apesar de ter sido utilizado em outros trabalhos como o de Ferreira [17], o framework apresenta alguns problemas durante sua utilização. O primeiro deles é o problema de geração de arquivos temporários. Nesse ponto, durante os processos de avaliação internos, é gerada uma grande quantidade de arquivos temporários que pode chegar a afetar o

funcionamento da máquina. Outro aspecto é a limitação no framework do tempo de execução para geração de produtos, 3600 segundos, que para uma LPS de grande porte não é tempo suficiente para a geração de todos os seus produtos válidos. Essa é uma limitação interna do framework e não pode ser alterada. Apesar disso, pela sua prévia integração com a ferramenta de Ferreira (FMTS), o mesmo foi utilizado nesse trabalho visando a uma melhor integração com os aspectos já implementados.

2.4 Considerações Finais

A utilização de LPS é de fundamental importância para as empresas que querem se manter competitivas nos dias atuais. É importante assegurar que os produtos gerados a partir da LPS estejam de acordo com os requisitos esperados. Nesse quesito, o teste de LPS torna-se relevante para a garantia de que a LPS está gerando *software* de qualidade. O teste de LPS será abordado no próximo capítulo.

CAPÍTULO 3

TESTE DE LINHA DE PRODUTO DE SOFTWARE

A aplicação de uma abordagem sistemática, disciplinada e quantificável para o desenvolvimento, operação e manutenção de *software* é o objetivo da Engenharia de *Software*. Essa definição abrange vários aspectos técnicos como qualidade, necessidade de satisfação do cliente e importância de um processo maduro de *software*. Nesse contexto destaca-se como um aspecto relevante a qualidade. Um dos principais elementos da garantia da qualidade de *software* é o teste de *software* [45].

O desenvolvimento de *software* não se caracteriza por ser uma atividade simples, além disso, o tamanho do sistema bem como a quantidade de características pode tornar a atividade de desenvolvimento extremamente complexa. Assim, essa atividade está sujeita a diversos tipos de problemas que resultam em produtos finais diferentes do esperado [12].

As atividades de Validação, Verificação e Teste (VV&T) possuem como objetivo descobrir as diferenças citadas antes que o *software* seja liberado para utilização. Sua finalidade consiste portanto em garantir que tanto o modelo de referência quanto o *software* construído estejam de acordo com as especificações. Assim, a atividade de VV&T é considerada um elemento crítico de garantia de qualidade de *software* [12].

Segundo Pressman [45] existem forças propulsoras para que a atividade de teste seja cuidadosa e bem planejada, como o crescente destaque para *software* em sistemas e os "custos" associados a falhas de *software*. O autor ainda afirma que aproximadamente 40% do esforço total de um projeto é dedicado à atividade de teste.

Este capítulo apresenta conceitos básicos do teste de *software* e os principais critérios de teste que vem sendo utilizados no teste de LPS.

3.1 Conceitos da Atividade de Teste

Myers e Sandler [37] definem teste como o processo de avaliar um determinado programa com a finalidade de encontrar defeitos. Além disso, possui como objetivos secundários, demonstrar que um programa trabalha aparentemente como desejado e fornecer indicações de confiabilidade e qualidade de *software*. Assim, diferentemente da engenharia de *software* o processo de teste é visto como um processo destrutivo já que objetiva a elaboração de casos de testes com o intuito de revelar defeitos. Myers e Sandler [37] apontam alguns outros aspectos relativos ao teste:

- A atividade consiste na execução de um programa com o intuito de revelar defeitos;
- Um bom caso de teste é aquele capaz de elevar a probabilidade de encontrar defeitos ainda não descobertos;
- O teste é bem sucedido se revelar um defeito ainda não descoberto.

Contudo, a atividade de teste não pode garantir a ausência de defeitos, somente pode demonstrar que há defeitos presentes [37]. Segundo as normas da IEEE [26] existem algumas definições adotadas na literatura para os principais conceitos relacionados à atividade de teste:

- Defeito (*Fault*): passo, proceso ou definição de dados incorretos;
- Engano (*Mistake*): ação humana capaz de produzir defeito;
- Erro (*Error*): estado inconsistente ou inesperado de um programa em sua execução;
- Falha (*Failure*): resultado incorreto gerado pela execução do programa com relação ao resultado (saída) esperado.

Técnicas de teste sistematizam a atividade de teste buscando revelar a maior quantidade de defeitos com o menor custo possível. Dentre as principais destacam-se a técnica funcional, técnica estrutural, técnica baseada em defeitos e o teste combinatorial. Essas técnicas são abordadas na próxima seção.

3.2 Técnicas de Teste

Myers e Sandler [37] definem a diferença entre as técnicas de teste através das fontes utilizadas para definir os requisitos das atividades de teste. Segundo os autores cada técnica de teste procura explorar determinados tipos de defeitos, estabelecendo requisitos de teste para os quais valores específicos de entrada devem ser definidos com o intuito de exercitá-los.

A Técnica Funcional é utilizada para projetar casos de teste considerando o *software* testado como uma caixa-preta. Para testá-lo são fornecidas as entradas e avaliadas as saídas verificando sua conformidade com os requisitos especificados. Essa técnica caracteriza-se por desconsiderar os detalhes da implementação avaliando o *software* a partir de uma perspectiva do usuário [37].

A Técnica Estrutural, ao contrário do Teste Funcional, utiliza a estrutura interna do programa para a geração dos requisitos de teste, sendo portanto necessária a execução de suas partes ou de componentes elementares. A idéia principal é avaliar os caminhos lógicos do *software*, constituindo casos de teste que exercitam tanto conjuntos específicos de condições ou ainda laços, tal como, o uso de variáveis e definições. Exemplos desses critérios são os critérios de teste baseados em fluxo de controle e baseados em fluxo de dados [37].

O Teste Baseado em Defeitos utiliza defeitos conhecidos para gerar os requisitos de teste. A principal motivação dessa técnica são os possíveis erros que podem ser cometidos pelos programadores e projetistas de sistema durante o processo de desenvolvimento de *software*. Dentre os principais critérios dessa técnica destaca-se a Análise de Mutantes que gera casos de testes específicos para demonstrar presença ou ausência de defeitos [17].

A Análise de Mutantes é baseada em dois princípios: Hipótese do Programador Competente e Efeito do Acoplamento. O primeiro princípio considera que, por mais que os programas possuam defeitos, os programadores sempre constroem o *software* bem próximo do correto. Dessa maneira, pode-se afirmar que defeitos são introduzidos através de desvios sintáticos que mudam a semântica do programa, levando-o a um possível comportamento incorreto. O segundo princípio considera que defeitos complexos são compostos de defei-

tos simples, portanto casos de teste projetados para revelar defeitos simples são capazes de revelar defeitos complexos [45].

De maneira geral, esse critério insere pequenas alterações sintáticas em um programa P, gerando assim um conjunto de programas mutantes. Essas alterações são inseridas através de operadores de mutação que, em geral, estão associados a um tipo ou classe de defeitos que se pretende revelar em P. Todos os mutantes são executados utilizando-se um conjunto de casos de teste. Se o programa mutante P' apresenta resultados diferentes do programa P, ele é dito morto, caso contrário, é dito vivo. Quando um mutante permanece vivo é necessário verificar se P e P' são equivalentes. Nesse casos, não existe caso de teste capaz de diferenciar esses dois programas. A avaliação da adequação de um conjunto de casos de teste é calculada pelo escore de mutação que é a relação entre o número de mutantes mortos e o número total de mutantes não equivalentes gerados [27].

Outra técnica de teste bastante difundida é o Teste Combinatorial. Esse tipo de teste utiliza um *array* de cobertura de casos de teste gerados através de um mecanismo de amostragem para que defeitos sejam revelados através da interação de parâmetros do *software* em teste [38]. Segundo Kuhn et.al. [32], em geral, a combinação entre apenas dois parâmetros não usuais é o suficiente para revelar alguns defeitos, o que justifica a popularidade de técnicas como o *pairwise*.

Utilizando o *pairwise* é possível gerar um subconjunto muito menor de possibilidades capaz de cobrir todos os pares de parâmetros de um *software* [40]. Assim, sua utilização é válida, pois apesar de requer menos esforço que um teste exaustivo apresenta uma cobertura tão boa quanto.

3.3 Teste em Linha de Produto de *Software*

Cohen et al. [10] afirmam que existe uma quantidade significativa de trabalhos explorando características únicas de abstrações de arquiteturas de *software* com o objetivo de validar sistemas. Nesse caso a validação pode ser de vários tipos como detecção de incompatibilidade de componentes, planejamento de teste de integração e utilização de noções de modelos de cobertura de *software* para adequação de conjuntos de teste. Em contrapar-

tida existem relativamente poucos trabalhos que exploram as características únicas de abstrações de LPS, a identificação de elementos comuns e variabilidade, para validação.

Como mencionado no capítulo anterior, Linhas de Produto de *Software* são geralmente representadas por FMs. Em tais diagramas as características são organizadas de maneira hierárquica, sendo que diferentes combinações de características geram diferentes produtos. Ensan et al. [15] afirmam que idealmente o teste de uma LPS requer que todos os possíveis produtos resultantes da combinação de suas características sejam testados. Entretanto, isso só seria viável em LPSs com pequeno número de características, pois à medida que cresce o número de características de uma LPS, cresce exponencialmente o número de possíveis produtos.

Além disso, Ensan et al. [15] enfatizam que o tempo necessário para o teste acompanha o crescimento exponencial. Considerando uma LPS com n características onde cada característica demora $O(m)$ para ser testada, um teste completo dessa LPS demoraria $O(2^n \times m)$. Portanto, o teste exaustivo torna-se impraticável tanto em questão de tempo para execução dos testes como em recursos necessários para a geração dos casos de teste. Esse problema aponta para os principais fatores de interesse quando se fala em teste de LPS: tempo e esforço necessários para testar todos os possíveis produtos. Esse problema relaciona-se com o objetivo do presente trabalho, visto que, a idéia é a seleção de um subconjunto de produtos capaz de reduzir o tempo e o esforço para o teste, com a maior cobertura possível em uma LPS.

A maioria dos trabalhos voltados especificamente para o teste de variabilidades e para a seleção de produtos está relacionada com teste combinatorial de características obtidas através de diagramas como FM ou OVM [17].

O teste *pairwise* é bastante utilizado para seleção de produtos para teste de LPS. De maneira geral, o teste *pairwise* trabalha com a cobertura de combinações feitas a partir de pares de características. O trabalho proposto por Cohen et al. [10] utiliza o *pairwise* como critério de cobertura para a seleção de produtos.

Outros exemplos de trabalhos que utilizam *pairwise* são os trabalhos de Lamancha e Usaola [33] e o de Oster et al. [40]. O primeiro utiliza a técnica em conjunto com a

construção de matrizes de pares de características a partir de regras que representam os relacionamentos entre as características. O segundo combina a utilização da técnica *pairwise* com teste baseado em modelos para a redução do tamanho de uma LPS, para cobertura de interações entre características.

Por outro lado, Ferreira [17] apresenta seu trabalho baseado em mutação de variabilidades para a seleção de produtos para o teste de LPS. Operadores de mutação são propostos para descrever erros comuns associados ao FM. Além disso, o autor propõe um processo de teste para aplicar os operadores propostos e realizar a seleção de produtos para teste. Tudo isso é feito através de uma ferramenta chamada FMTS (*Feature Mutation Test Suit*).

Henard et al. [23] introduziram dois operadores de mutação, eles são definidos para gerar casos de teste dissimilares para revelar defeitos nos FMs. Entretanto, uma limitação desse trabalho é que os operadores não são orientados para erros comuns que podem estar presentes na FM. Outra limitação é que o trabalho não explora o uso de tais operadores para a geração de conjuntos de teste; eles são utilizados apenas para a avaliação desses conjuntos. Nesse sentido, o trabalho de Ferreira [17] se apresenta como uma abordagem mais completa pois além de levantar as classes de defeitos comuns e elaborar dezesseis operadores para revelá-los, ainda propõe a utilização desses operadores para a geração de casos de teste. Por esse motivo ele será utilizado neste trabalho.

As técnicas *Pairwise* e Mutação de Variabilidades são de especial interesse para este trabalho e são descritas nas próximas subseções.

3.3.1 Teste *Pairwise*

Segundo Ferreira [17] a maior parte dos trabalhos voltados para a seleção de produtos para teste estão baseados no teste de combinação de características obtidas a partir de modelos, tais como o FM e o OVM.

Tai e Lei [54] afirmam que teste *pairwise* é um critério de teste baseado em especificações. Assim, para cada par de parâmetros de entrada do sistema cada combinação de valores válidos desses dois parâmetros deve ser coberta por, pelo menos, um caso de teste.

Os autores ainda exemplificam o conceito de teste *pairwise* considerando um sistema com os valores e parâmetros descritos a seguir:

- Parâmetro A possui os valores A_1 e A_2;
- Parâmetro B possui os valores B_1 e B_2;

Para os parâmetros A e B, o conjunto de teste *pairwise* (A_1, B_1), (A_1, B_2), (A_2, B_1), (A_2, B_2) é o único existente. Esse conjunto leva em consideração todas as combinações dois a dois dos valores dos parâmetros, cobrindo dessa maneira, todos os possíveis casos de teste [54]. O mesmo princípio pode ser considerado no contexto de LPS, em que ao invés de parâmetros e valores considera-se como entrada as características de um diagrama de características. Vide exemplo da Seção 3.3.1.1.

Oster et al. [40] apresentam o teste *pairwise* como uma técnica de geração de casos de teste (produtos) para LPS, baseado na observação de que a maioria dos defeitos são causados através da interação de, pelo menos, duas características. A geração de casos de teste utilizando a técnica garante a cobertura de todas as combinações de duas características.

Cohen et al. [9] apresentam o algoritmo combinatorial AETG, para geração de casos de teste, baseado nos critérios de teste combinatorial como o *pairwise*. Segundo os autores esse algoritmo é capaz de gerar casos de teste que cobrem todas as n-combinações válidas de características. O tamanho de um conjunto de casos de teste gerado pelo AETG cresce de forma logarítmica em relação ao número de características de entrada. Isso permite aos testadores definir modelos de teste com dezenas de parâmetros (características). Em muitas aplicações a utilização do algoritmo resultou em uma redução significativa do custo de desenvolvimento.

Nesse contexto, destaca-se a utilização da ferramenta chamada *Combinatorial Tool* [44]. A ferramenta em questão possui implementados vários algoritmos baseados em combinação para a geração de casos de teste. As implementações englobam algoritmos como *All Combinations*, alguns algoritmos *pairwise* como AETG, dentre outros. Além disso, trata-se de uma ferramenta gratuita e *on-line*, o que incentiva sua utilização.

3.3.1.1 Exemplo de Geração de Casos de Teste

Para ilustrar a aplicação de teste *pairwise* utilizando o algoritmo AETG, o exemplo a seguir utiliza a LPS AGM, apresentada na Figura 2.1. Utilizando a ferramenta *Combinatorial Tool* foram gerados os conjuntos de combinações das Tabelas 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, de acordo com a técnica *pairwise*:

Tabela 3.1: Pares de combinações entre as subcaracterísticas de Services e Rules

Services - Rules
Play - Brickles
Play - Pong
Play - Bowling
Pause - Brickles
Pause - Pong
Pause - Bowling
Save - Brickles
Save - Pong
Save - Bowling

Tabela 3.2: Pares de combinações entre as subcaracterísticas de Services e Configuration

Services - Configuration
Play - Configuration
Pause - Configuration
Save - Configuration

Tabela 3.3: Pares de combinações entre as subcaracterísticas de Services e Action

Services - Action
Play - Movement
Play - Collision
Pause - Movement
Pause - Collision
Save - Movement
Save - Collision

Tabela 3.4: Pares de combinações entre as subcaracterísticas de Rules e Configuration

Rules - Configuration
Brickles - Configuration
Pong - Configuration
Bowling - Configuration

Tabela 3.5: Pares de combinações entre as subcaracterísticas de Rules e Action

Rules - Action
Brickles - Movement
Brickles - Collision
Pong - Movement
Pong - Collision
Bowling - Movement
Bowling - Collision

Tabela 3.6: Pares de combinações entre as subcaracterísticas de Configuration e Action

Configuration - Action
Configuration - Movement
Configuration - Collision

Depois de gerados os pares de combinações, o algoritmo AETG foi executado resultando nos produtos descritos na Tabela 3.7. Vale ressaltar que o resultado considera apenas as variabilidades de um diagrama de características e não suas relações de inclusão e exclusão.

Tabela 3.7: Resultado da Execução do AETG

Produto	**Características**
1	Play, Brickles, Configuration, Movement
2	Play, Pong, Configuration, Collision
3	Play, Bowling, Configuration, Movement
4	Pause, Brickles, Configuration, Collision
5	Pause, Pong, Configuration, Movement
6	Pause, Bowling, Configuration, Collision
7	Save, Brickles, Configuration, Movement
8	Save, Pong, Configuration, Collision
9	Save, Bowling, Configuration, Movement

3.3.2 Teste de Mutação de Variabilidades

Ferreira [17] afirma que os trabalhos existentes para a seleção de produtos (casos de teste) de uma LPS, a partir do diagrama de características, não consideram os defeitos que podem estar presentes. Segundo o autor, o teste de mutação pode aumentar a probabilidade de revelar defeitos, aumentando assim a confiança de que os produtos gerados atendem às especificações desejadas. Baseado nesses argumentos o autor propõe uma abordagem

de geração e avaliação de conjuntos de casos de teste, baseada em mutação do diagrama de características de uma Linha de Produto de *Software*.

Para atingir o objetivo do trabalho Ferreira [17] propõe a análise das possíveis classes de defeitos que podem ocorrer em um diagrama de características. Considerando essas classes de defeitos foram propostos operadores de mutação capazes de revelar tais defeitos. As classes de defeito identificadas são:

- Atribuição Incorreta da Cardinalidade de uma Característica Solitária: Ocorre quando a cardinalidade de uma característica solitária é definida incorretamente;
- Atribuição Incorreta de Elementos de uma Relação Agrupada: Pressupõe-se que uma das características agrupadas não deveria fazer parte da relação de agrupamento, ou ainda, que uma característica solitária deveria ser incluída na relação de agrupamento;
- Existência de uma Relação de Agrupamento: Corresponde aos defeitos associados a relações de agrupamento incorretamente criadas;
- Atribuição Incorreta de Cardinalidade de uma Característica Agrupada: Um engano, ao definir os valores mínimos e máximos para a cardinalidade, pode resultar em diagramas que permitam programas com menos ou mais produtos do que o especificado;
- Restrições Incorretas: Engloba as seguintes situações: (i) uma das características de uma restrição foi incorretamente selecionada; (ii) a restrição não deveria existir; (iii) uma restrição está ausente no diagrama de características.

Os operadores de mutação propostos, de acordo com as classes, são os seguintes:

1. Atribuição incorreta da cardinalidade de uma característica solitária;
    - DFL: Decrementa o valor mínimo da cardinalidade de uma característica solitária;

- IFL: Incrementa o valor mínimo da cardinalidade de uma característica solitária;
- DFU: Decrementa o valor máximo da cardinalidade de uma característica solitária;
- IFU: Incrementa o valor máximo da cardinalidade de uma característica solitária;

2. Atribuição incorreta de elementos de uma relação agrupada;

- AFS: Adiciona uma característica solitária a uma relação de agrupamento;
- RFS: Remove uma característica de uma relação de agrupamento;

3. Existência de uma relação de agrupamento;

- RSR: Remove uma relação de agrupamento;

4. Atribuição incorreta da cardinalidade de uma característica agrupada;

- DRL: Decrementa o valor mínimo da cardinalidade de uma característica agrupada;
- IRL: Incrementa a valor mínimo da cardinalidade de uma característica agrupada;
- DRU: Decrementa o valor máximo da cardinalidade de uma característica agrupada;
- IRU: Incrementa a valor máximo da cardinalidade de uma característica agrupada;

5. Restrições incorretas;

- FDC: Inverte as características de uma relação de dependência;
- RDC: Remove uma restrição de dependência;
- REC: Remove uma restrição de exclusão;

- CDC: Cria uma restrição de dependência;
- CEC: Cria uma restrição de exclusão.

A Figura 3.1 apresenta um exemplo de aplicação do operador de mutação RDC. Nesse caso a aplicação do operador sobre a restrição RX removeu-a do diagrama. Para "matar" esse diagrama mutante é necessário gerar um produto x, tal que x seja válido para o diagrama original e inválido para o mutante, ou ao contrário, x seja inválido para o diagrama original e válido para o mutante. Neste caso o diagrama mutante é dito "morto". Nesse contexto, dizer que determinado produto é válido corresponde a dizer que ele respeita as restrições de um determinado diagrama. No caso de ambos diagramas mutante e original gerarem o mesmo conjunto de produtos diz-se que o diagrama mutante é equivalente.

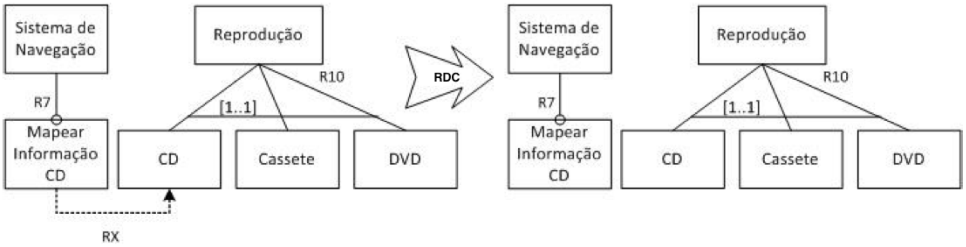

Figura 3.1: Exemplo de aplicação do operador RDC [17]

Resumidamente o processo de teste proposto por [17] inicia-se a partir de um diagrama de características válido fornecido, sendo dividido em três partes: geração de mutantes, geração de casos de teste (produtos) e a execução dos casos de teste. A etapa de geração de mutantes consiste na aplicação dos operadores de mutação citados, de acordo com uma porcentagem definida pelo usuário, ao diagrama de características de entrada. Essa etapa gera como saída um conjunto de diagramas mutantes.

A etapa de geração de casos de teste tem como saída um conjunto de produtos (casos de teste) válidos para o diagrama de características fornecido como entrada ou para algum dos diagramas mutantes.

A partir dos diagramas mutantes gerados e do conjunto de casos de teste, a etapa de execução dos casos de teste avalia os produtos gerados, utilizando os mutantes produzidos. Isso consiste em checar se produtos fornecidos são válidos tanto para o diagrama de

entrada como para os diagramas mutantes. Caso o resultado de ambas avaliações seja diferente considera-se que o caso de teste "matou" o mutante [17].

Cada produto (ou caso de teste) é executado para todos os diagramas mutantes com o objetivo de satisfazer o critério análise de mutantes, com um conjunto reduzido de casos de teste. Caso o conjunto gerado não satisfaça esse critério, a etapa de criação identifica os diagramas equivalentes e gera novos casos de teste, visando maximar o escore de mutação [17]. A Figura 3.2 apresenta o processo de teste proposto por Ferreira [17].

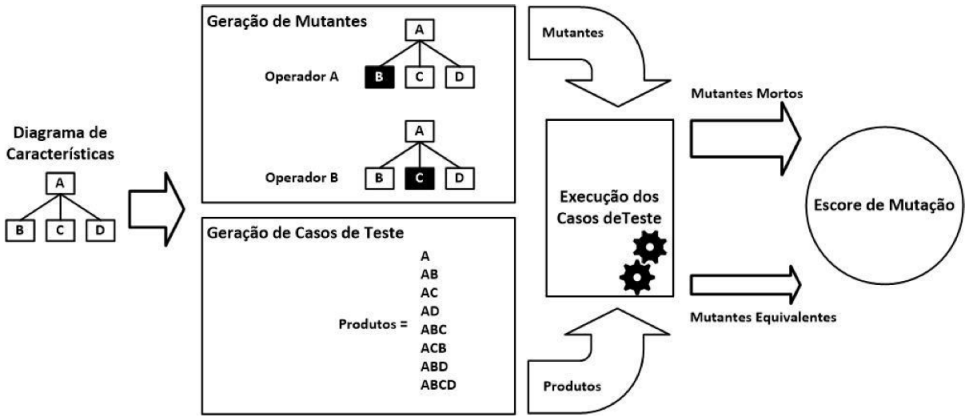

Figura 3.2: Processo de teste proposto em [17]

Para facilitar e automatizar a abordagem de teste utilizada, Ferreira [17] desenvolveu uma ferramenta chamada FMTS sigla que vem do inglês *Feature Mutation Test Suite*. A ferramenta recebe como entrada um diagrama de características no formato "XML". Segundo o autor, a utilização da FMTS possibilita que, para determinado diagrama de características, seja possível realizar o processo de teste de mutação de características. O processo envolve alguns passos como (i) aplicação dos operadores de mutação e a geração de diagramas mutantes; (ii) submissão de conjuntos de casos de teste; (iii) avaliação do conjunto gerado. A ferramenta possui ainda outras funcionalidades como a avaliação de um conjunto de teste existente e módulo de apoio à identificação de diagramas equivalentes. A validação dos produtos e mutantes é feita com o auxílio do *framework* FaMa.

3.4 Considerações Finais

A atividade de teste destaca-se por estar relacionada à qualidade de *software*. No contexto de LPS essa realidade não é diferente. O teste em LPS está em destaque, uma vez que esse contexto está em relevância atualmente. Dentre os critérios para teste de LPS estudados, destacam-se o teste *pairwise* bastante difundido e o teste de mutação de variabilidades, que apresenta resultados promissores. Além disso, verifica-se a existência de ferramentas que auxiliam a utilização de tais critérios, respectivamente, o algoritmo AETG implementado pela ferramenta *on-line CombinatorialTool* e a ferramenta FMTS que implementa o teste de mutação de variabilidades.

Entretanto, satisfazer o critério análise de mutantes, isto é, gerar dados de teste para matar os mutantes não é uma tarefa fácil. Ela envolve diferentes fatores como por exemplo número de casos de teste, restrição de custo e cobertura de outros critérios. Por isso essa tarefa é tratada neste trabalho com o uso de algoritmos multiobjetivos, assunto do próximo capítulo.

CAPÍTULO 4

OTIMIZAÇÃO MULTIOBJETIVO

Muitos problemas do mundo real, como os problemas de custo *versus* recursos de uma indústria, mesmo possuindo objetivos múltiplos e conflitantes, podem ser representados por meio de um modelo simplificado de tratamento [1]. Nesse caso, o modelo possui um único objetivo avaliado através da função objetivo f(X), onde X corresponde a um vetor n-dimensional $X = [x_1, x_2, ..., x_n]$ de variáveis de decisão, pertencentes a um universo de busca Ω. Essa é uma maneira de reduzir um problema multiobjetivo a um problema mono-objetivo ou de objetivo único [7].

Em alguns casos, entretanto, é necessária a otimização simultânea dos objetivos interdependentes e muitas vezes conflitantes de um determinado problema. Nesse caso, não há apenas uma única solução mas um conjunto de soluções que representam um possível compromisso entre os diferentes objetivos [1, 7]. A otimização multiobjetivo desses problemas se dá por meio de algoritmos de otimização multiobjetivos. Existem diferentes tipos desses algoritmos. Os mais conhecidos e utilizados são os baseados na evolução, chamados algoritmos evolutivos. Na área de SBSE, a maioria dos trabalhos utilizam algoritmos evolutivos [21]. Esses trabalhos demonstram um melhor desempenho desses algoritmos na resolução dos problemas da área. Por esse motivo, algoritmos evolutivos são utilizados no presente trabalho.

Neste capítulo são fornecidos os principais conceitos relativos à otimização de problemas multiobjetivos, algoritmos genéticos, algoritmos evolutivos multiobjetivos e indicadores de qualidade de soluções utilizados na área de otimização.

4.1 Principais Conceitos

De maneira geral, um problema de otimização multiobjetivo pode ser definido como o problema de minimizar (analogamente para um problema de maximização) a Equação 4.1 [8].

$$F(x) = (f_1(x), f_2(x), ..., f_k(x)) \tag{4.1}$$

sujeito às Equações 4.2 e 4.3.

$$g_i(x) \leq 0, \ para \ i = \{1, 2, ..., m\} \tag{4.2}$$

$$h_j(x) = 0, \ para \ j = \{1, 2, ..., p\} \quad x \in \Omega \tag{4.3}$$

onde k corresponde ao número de funções objetivo sendo que $f_1(x), f_2(x), ..., f_k(x)$ correspondem às funções objetivo a serem minimizadas, x é um vetor de variáveis de decisão $x = (x_1, x_2, ..., x_n)$ e Ω corresponde ao conjunto finito de todas as possíveis soluções que satisfazem $F(x)$. Além disso, $g_i \leq 0$ e $h_j = 0$ são restrições que devem ser satisfeitas durante a minimização de $F(x)$ [8].

Considerando duas soluções $x \in \Omega$ e $y \in \Omega$, para um problema de minimização, a solução x domina y se e somente se as Equações 4.4 e 4.5 são satisfeitas, ou seja, x é dita uma solução não dominada se não existe qualquer solução y que domine x. Uma solução diz-se *não dominada* quando não há nenhuma outra solução admissível que melhore simultaneamente todos os objetivos, ou seja, a melhoria de um objetivo só é alcançada ao custo de prejudicar, pelo menos, um outro objetivo [6].

$$\forall f_i \in F, i = 1...K, f_i(x) \leqslant f_i(y) \tag{4.4}$$

$$\exists f_i \in F, f_i(x) \leq f_i(y) \tag{4.5}$$

O conceito de fronteira de Pareto, descrito na próxima seção, é utilizado para encontrar um conjunto de soluções candidatas a solucionar um problema multiobjetivo [7].

4.1.1 Fronteira de Pareto

Para solucionar problemas multiobjetivos procura-se encontrar soluções que melhor representem o compromisso entre os objetivos de um problema. Essas soluções formam um conjunto de soluções não dominadas no espaço de busca. O conjunto de soluções não dominadas para determinado problema chama-se fronteira de Pareto [7].

A Figura 4.1 mostra uma fronteira de Pareto com base em um esquema de custo/benefício para veículos automotores, nesse caso, os dois objetivos apresentados são Potência e Custo. Os pontos 1, A, B, C e 2 representam as soluções não dominadas para o problema e a linha formada pelos mesmos representa a fronteira de Pareto. Nos extremos da fronteira temos os pontos 1 e 2 que representam, respectivamente, o carro com menor custo e maior potência. Entre os extremos tem-se os pontos A, B e C que não apresentam o menor custo nem a maior potência, mas são soluções que equilibram tais objetivos.

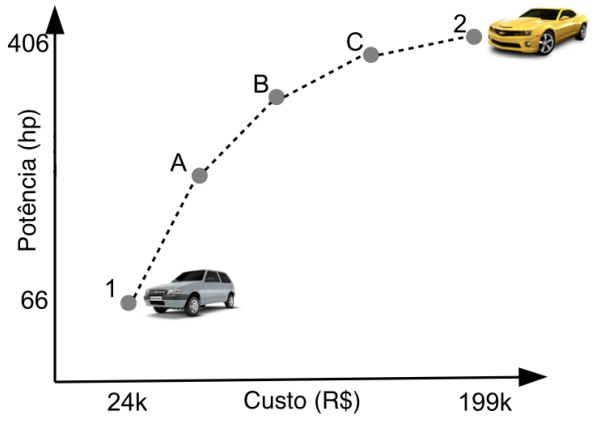

Figura 4.1: Exemplo de Fronteira de Pareto (Adaptada de [35])

4.2 Algoritmos Genéticos

Algoritmos Genéticos (AGs) são inspirados na teoria da evolução, em que as espécies que não conseguem se adaptar ao seu ambiente são extintas através da seleção natural. As espécies mais fortes possuem maior oportunidade de passar seus genes para a geração futura. A longo prazo, as espécies que levam a combinação correta em seus genes tornam-se dominantes em sua população. Por vezes, durante o processo de evolução, mudanças

aleatórias podem ocorrer nos genes. Se essas mudanças oferecem vantagens adicionais no desafio de sobrevivência, novas espécies evoluem a partir das antigas. Caso contrário, as espécies que foram mudadas são eliminadas pela seleção natural [30].

Algoritmos Genéticos operam com um conjunto de cromossomos, chamado de população. Os cromossomos são compostos por genes. Cada gene possui uma ou mais características. Normalmente, um cromossomo corresponde a uma única solução em um espaço de busca [30]. Além disso, é necessário estabelecer a função de avaliação (função de *fitness*) que define quais indivíduos da população (soluções) serão selecionados. Em termos práticos isso significa comparar os valores obtidos para a função de *fitness* de cada filho gerado e selecionar aqueles com melhores valores. Esses indivíduos com melhor *fitness*, ou adaptabilidade, possuem maior probabilidade de sobreviver e se reproduzir. É aí onde entra um mecanismo específico de seleção [5].

Para gerar novas soluções a partir das já existentes, dois tipos de operadores são utilizados: recombinação (ou cruzamento) e mutação. Na recombinação, dois cromossomos, chamados de pais, são combinados para formar novos cromossomos, chamados de filhos. Os pais são selecionados entre os cromossomos existentes na população, com preferência para os que tiverem melhor valor de *fitness*. Através da aplicação do operador de recombinação, genes de bons cromossomos são esperados para aparecer com mais frequência na população, o que eventualmente leva a uma convergência para uma solução. O operador de mutação introduz mudanças aleatórias nas características dos cromossomos, provendo a diversidade genética na população e auxiliando na busca de uma solução ótima [30].

Segundo Konak et al. [30] o procedimento geral de um algoritmo genético é dado pelos seguintes passos:

- **Passo 1:** Gerar soluções aleatoriamente para formar a primeira população e avaliar o *fitness* das soluções desta população.
- **Passo 2:** Gerar uma população de descendentes através da recombinação.
- **Passo 3:** Aplicar a mutação em cada solução com uma taxa pré-definida.

- **Passo 4:** Avaliar e atribuir um valor de *fitness* para cada solução com base no valor da função objetivo.
- **Passo 5:** Selecionar soluções com base no valor de *fitness* e copiá-las para a próxima população.
- **Passo 6:** Se o critério de parada for satisfeito, terminar a pesquisa, retornar para a população atual e voltar ao Passo 2.

4.3 Algoritmos Evolutivos Multiobjetivos

Algoritmos evolutivos multiobjetivos (*Multi-objective Evolutionary Algorithm* - MOEAs) são utilizados para a otimização de problemas multiobjetivos. Em geral, os objetivos utilizados são conflitantes, ou seja, a melhoria de um objetivo influi diretamente na piora de outro objetivo. Dessa maneira, não existe apenas uma solução ótima, mas sim um conjunto de soluções que melhor representam um compromisso entre os diferentes objetivos. Assim, MOEAs trabalham simultaneamente com um conjunto de soluções possíveis (também chamado população), o que permite encontrar várias soluções não dominadas, ditas ótimas, em apenas uma execução do algoritmo [8].

A primeira geração de MOEAs é caracterizada pela utilização do *niching* ou *fitness sharing* combinado com o *ranking* de Pareto. Os algoritmos mais representativos dessa geração são: *Non-dominated Sorting Genetic Algorithm* (NSGA), *Niched-Pareto Genetic Algorithm* (NPGA) e *Multi-objective Genetic Algorithm* (MOGA) [1].

A segunda geração de MOEAs surgiu com a inserção do conceito de elitismo. Em um contexto com único objetivo a utilização de algoritmos genéticos implica que a melhor solução encontrada sempre sobrevive para a próxima geração. Assim, as soluções encontradas por um algoritmo genético multiobjetivo são consideradas soluções elitistas. Entretanto, a implementação de elitismo em uma otimização multiobjetivo não é tão fácil quanto em uma otimização com único objetivo, devido ao grande número de possíveis soluções elitistas. Inicialmente os algoritmos genéticos não utilizavam o elitismo. Porém, a maioria dos algoritmos genéticos multiobjetivos e suas variações passaram a utilizá-

lo. Os MOEAs utilizam duas estratégias para a manutenção do elitismo: (i) mantendo soluções elitistas na população e (ii) armazenando soluções elitistas em arquivos externos, reintroduzindo posteriormente as mesmas na população [30]. Algoritmos da segunda geração serão utilizados no presente trabalho. Esses algoritmos são descritos mais detalhadamente a seguir.

4.3.1 *Nondominated Sorting Genetic Algorithm II* (NSGA-II)

Esse algoritmo [28] utiliza elitismo para manter a diversidade. O mecanismo de elitismo utilizado consiste em combinar os melhores pais com os melhores filhos [1]. O pseudocódigo do NSGA-II é apresentado no Algoritmo 4.1.

```
   Entrada: N', g, f_k(X)
 1 Inicializar população P'
 2 Gerar população aleatória - Tamanho N'
 3 Avaliar valores dos objetivos
 4 Atribuir rank baseado na dominância de Pareto - Ordenação
 5 Gerar população filho
 6     Seleção por torneio binário
 7     Cruzamento e Mutação
 8 para i=1 até g faça
 9 |   para cada Pai e Filho na População faça
10 |   |   Atribuir rank baseado na dominância de Pareto - Ordenação
11 |   |   Gerar fronteiras não dominadas
12 |   |   Ordenar cada solução das fronteiras considerando a distância de multidão e
   |   |   Percorrer todas as fronteiras adicionando para a próxima geração do
   |   |   primeiro ao N' indivíduo
13 |   fim
14 |   Selecionar indivíduos das melhores fronteiras e com maior distância de multidão
15 |   Gerar população filho
16 |       Seleção por torneio binário
17 |       Cruzamento e Mutação
18 fim
```

Algoritmo 4.1: Pseudocódigo do NSGA-II (Adaptado de [7])

O algoritmo recebe como entrada o tamanho da população "N'", número de gerações "g" e as funções que serão otimizadas $f_k(X)$, onde k representa o número de objetivos que serão otimizados.

A cada geração o algoritmo NSGA-II ordena os indivíduos das populações de pais e

filhos de acordo com a dominância entre as soluções, formando diversas fronteiras (linhas 10 e 11 do Algoritmo 4.1). A primeira fronteira corresponde às soluções não dominadas de toda a população, a segunda é composta por todas as soluções que passam a ser não dominadas, após retiradas as soluções da primeira fronteira, a terceira é composta por soluções que passam a ser não dominadas após a retirada da primeira e segunda fronteiras, e assim sucessivamente até todas as soluções estarem classificadas em alguma fronteira [3].

Em cada fronteira uma ordenação é feita usando uma outra medida, chamada distância de multidão (*crowding distance*), que tem como objetivo manter a diversidade das soluções. A distância de multidão calcula o quão distante está uma solução de seus vizinhos da mesma fronteira visando a estabelecer uma ordem decrescente que privilegia as soluções mais espalhadas no espaço de busca. Soluções que estão no limite do espaço de busca apresentam só um vizinho, mas são as mais diversificadas da fronteira, elas recebem altos valores para estarem no topo da ordenação [3].

O operador de seleção (linhas 6 e 16 do Algoritmo 4.1) utiliza ambas ordenações, de fronteiras e de distância de multidão. As ordenações são utilizadas também para determinar os indivíduos que sobrevivem para a próxima geração. O NSGA-II utiliza seleção por torneio, selecionando soluções de fronteiras com maior dominância. A criação de novos indivíduos é efetuada mediante a aplicação dos operadores de cruzamento e mutação (linhas 7 e 17 do Algoritmo 4.1) [3].

4.3.2 *Strength Pareto Evolutionary Algorithm 2* (SPEA2)

Esse algoritmo [14] apresenta como principal diferença em relação ao NSGA-II a forma de calcular a *fitness* e a utilização de elitismo através de um arquivo externo. O pseudocódigo do SPEA2 é apresentado no Algoritmo 4.2.

Entrada: $N', \overline{N}, g, f_k(X)$
1 Inicializar população $\mathbb{P}'$ - Tamanho N'
2 Criar um arquivo vazio $\mathbb{E}'$
3 **para** *i=1 até g* **faça**
4 | Calcular o *fitness* para cada indivíduo de $\mathbb{P}'$ e $\mathbb{E}'$
5 | Copiar todos os indivíduos não dominados de $\mathbb{P}'$ e $\mathbb{E}'$ para $\mathbb{E}'$
6 | **se** *Tamanho de* $\mathbb{E}'$ *maior que* $\overline{N}$ **então**
7 | | Usar operador de eliminação de soluções de $\mathbb{E}'$
8 | **senão se** $\mathbb{E}'$ *menor que* $\overline{N}$ **então**
9 | | Usar soluções dominadas de $\mathbb{P}'$ para completar $\mathbb{E}'$
10 | **fim**
11 | Executar seleção por torneio binário para preencher a *mating pool*
12 | Aplicar Cruzamento e Mutação para a *mating pool*
13 **fim**

Algoritmo 4.2: Pseudocódigo do SPEA2 (Adaptado de [7])

Como visto, as entradas do Algoritmo 4.2 são semelhantes as entradas do NSGA-II, com exceção do parâmetro "$\overline{N}$" que representa o tamanho do arquivo externo.

Para cada geração do SPEA2 todas as soluções têm um valor calculado chamado *strenght* que é utilizado para definir o *fitness* da solução. O valor de *strenght* de uma solução i corresponde ao número de indivíduos, que pertencem tanto ao arquivo externo como à população e que dominam a solução i [3].

A soma de todos os valores de *strenght* de soluções não dominadas por i corresponde a *fitness* de uma solução i (linha 4 do Algoritmo 4.2). Valor de *fitness* igual a zero (0) indica que um determinado indivíduo não é dominado por nenhuma outra solução. Por outro lado, valores altos de *strength* representam soluções dominadas por vários outros indivíduos [3].

Como o tamanho do arquivo externo é determinado por parâmetro, duas situações podem ocorrer: na primeira delas o arquivo pode estar com mais soluções não dominadas que seu limite, então uma operação para eliminar as soluções é efetuada (linha 7 do Algoritmo 4.2). Na segunda situação o número de soluções não dominadas é menor que o tamanho do arquivo, nesse caso, o arquivo é preenchido com as soluções dominadas (linha 9 do Algoritmo 4.2). Somente indivíduos que pertencem ao arquivo externo sobrevivem para uma próxima geração. A criação de novos indivíduos para compor uma

nova população é feita através da aplicação de operadores de mutação e cruzamento em indivíduos selecionados da população e do arquivo externo [3].

4.3.3 *Indicator-Based Evolutionary Algorithm* (IBEA)

No algoritmo IBEA [62], para cada solução encontrada um peso é atribuído de acordo com indicadores de qualidade (Seção 4.4), favorecendo assim os objetivos de otimização do usuário. O Algoritmo 4.3 apresenta o pseudocódigo do algoritmo IBEA.

```
   Entrada: α, N, k
 1 Gerar uma população P de tamanho α
 2 Contador de geração m recebe 0
 3 enquanto tamanho população P não exceda α faça
 4 |   para cada Indivíduo na População faça
 5 |   |   Calcular o fitness baseado no indicador de qualidade
 6 |   fim
 7 |   Escolher um indivíduo x pertencente a P com menor valor de fitness
 8 |   Remover x da população P
 9 |   Atualizar a fitness dos indivíduos que restaram na população
10 |   se m maior que N ou outro critério seja satisfeito então
11 |   |   Considerar os indivíduos não dominados em P como o vetor de decisão
12 |   senão
13 |   |   Realizar seleção por torneio binário em P com a finalidade de encher a base
   |   |   P'
14 |   |   Aplicar operadores de recombinação e mutação à base P' e adicionar os
   |   |   indivíduos à população P
15 |   |   Incrementar m e ir para a linha 5
16 |   fim
17 fim
```

Algoritmo 4.3: Pseudocódigo do IBEA (Adaptado de [50])

O algoritmo possui como entrada os valores: α que representa o tamanho da população, N o número máximo de gerações; e k o fator de escala de *fitness*. Para cada indivíduo da população um valor de *fitness* é calculado baseado no fator k (linha 5). Os indivíduos com menor valor de *fitness* são removidos da população (linhas 7 e 8). O *fitness* dos outros indivíduos são atualizados. Caso o contador de gerações seja maior que o número máximo de gerações, o vetor de decisão é retornado, composto pelos indíviduos não dominados da população P, e encerra-se a execução (linha 11). Senão, ocorre um torneio binário para compor uma base P' (linha 14). Os indivíduos adicionados a esta base sofrem mutações

e recombinações e são novamente adicionados à população P (linha 15). Após isso, o algoritmo volta para o cálculo de todas as *fitness* (linha 5) até que o vetor decisão seja obtido.

4.4 Indicadores de Qualidade

São utilizados para comparar o desempenho dos algoritmos que estão sendo avaliados. Baseiam-se, geralmente, nos conjuntos de soluções encontradas pelos algoritmos. Costumeiramente três conjuntos são utilizados [7]:

- PF_{approx}: formado por soluções não dominadas retornadas por um algoritmo executado;
- PF_{known}: resultado da união de todos os PF_{approx} obtidos por determinado algoritmo, removendo as soluções dominadas e repetidas. Representa as melhores soluções encontradas por um algoritmo para determinado problema;
- PF_{true}: representa a fronteira de Pareto ótima para o problema. Quando desconhecida, pode ser formada a partir da união de todas as soluções encontradas em todas as rodadas de todos os algoritmos que tiverem sido executados, removendo as soluções dominadas e repetidas [63].

Os principais indicadores utilizados são abordados a seguir

4.4.1 *Hypervolume*

Trata-se de uma medida do espaço coberto entre a Fronteira de Pareto e um ponto de referência. Se todos os objetivos devem ser maximizados então, preferencialmente, a fronteira de Pareto é aquela com maior *Hypervolume*. Considerando um problema de otimização com dois objetivos tendo que cada um dos pontos $(f_1(x^*); f_2(x^*))$ pertencente a fronteira de Pareto delimita um retângulo no espaço dos objetivos, o *hypervolume* corresponde a área formada pela soma de todos os retângulos [61].

4.4.2 Error Ratio

Corresponde a proporção entre os elementos do conjunto PF_{known} que estão e os que não estão presentes na fronteira PF_{true} [56]. A Equação 4.6 apresenta o cálculo desse indicador.

$$ER(A) = \frac{\sum_{a \in A^e}}{|A|} \tag{4.6}$$

Na equação acima, A corresponde ao conjunto de soluções da fronteira PF_{known} e e possui valor 1 quando a solução não está presente na fronteira PF_{true} ou 0 caso contrário. O valor obtido através desse indicador está entre 0 e 1. Quanto menor o número do *error ratio* maior é a quantidade de soluções de PF_{known} presentes em PF_{true}, em outras palavras, melhor é a fronteira no quesito quantidade de soluções ótimas encontradas.

4.4.3 Distância Euclidiana

Distância Euclidiana (ED) mede a distância entre uma solução e a solução ideal em um determinado espaço de objetivos. A solução ideal possui os melhores valores possíveis em todos os objetivos do problema. Por exemplo, se os valores encontram-se normalizados (entre 0 e 1), para um problema de minimização de dois objetivos, então a solução ideal seria o conjunto (0,0). Em outras palavras, o indicador mede quão próxima uma solução está de ser ideal. Quanto menor o valor de ED, melhor é o *trade-off* de objetivos da solução. A Equação 4.7 apresenta o cálculo da distância Euclidiana entre dois pontos $P = (p_1, p_2, \ldots, p_n)$ e $Q = (q_1, q_2, \ldots, q_n)$ num espaço euclidiano n-dimensional.

$$\sqrt{\sum_{i=1}^{n} (p_i - q_i)^2} \tag{4.7}$$

4.5 *Framework* jMetal

jMetal em inglês significa *Metaheuristic Algorithms in Java* e consiste em uma ferramenta baseada em Java, para otimização multiobjetivo com metaheurísticas [13].

A arquitetura baseada em objetos do *framework* e suas características permitem: realizar experimentos com o atual estado da arte das técnicas, desenvolver algoritmos, resolver problemas de otimização, integrar o jMetal com outras ferramentas; dentre outras funcionalidades [13].

Além disso, essa ferramenta apresenta diversas implementações de algoritmos multiobjetivos como NSGA-II, SPEA2, IBEA. Inclui ainda várias instâncias de problemas e indicadores de qualidade como *hypervolume* (HV).

A principal motivação dos criadores do jMetal é a disponibilização de programas utilizados em seus próprios trabalhos para a comunidade de pesquisa de otimização multiobjetivo [13]. Baseado nisso, e na importância dessa ferramenta, esse *framework* será utilizado na implementação realizada neste trabalho.

4.6 Considerações Finais

A otimização multiobjetivo tem ganhado destaque nos últimos anos frente à característica multiobjetivo dos problemas do mundo real. O entendimento de alguns conceitos como Ótimo de Pareto, Fronteira de Pareto e elitismo são fundamentais para o bom desenvolvimento de um trabalho de otimização multiobjetivo. Nesse contexto, destaca-se a utilização de algoritmos baseados na natureza (evolutivos, por exemplo) para a resolução de tais problemas. Entretanto não basta somente a aplicação desses algoritmos, são precisos indicadores de qualidade para avaliar quão bons, ou não, são os algoritmos em questão na resolução de determinado problema multiobjetivo.

Algoritmos de otimização têm sido utilizados no problema de configuração de Linha de Produto de Software com os mais variados propósitos como evolução, customização e teste. Trabalhos sobre teste de LPS são os mais relacionados ao presente trabalho e são apresentados no próximo capítulo.

CAPÍTULO 5

TESTE BASEADO EM BUSCA DE LPS

Como já abordado, uma Linha de Produto de *Software* possui uma grande quantidade de características. Muitas delas são comuns a vários produtos, enquanto outras, são específicas a determinados produtos. Produtos são formados pela junção de uma ou mais características. A seleção ou não de determinada característica para formação de um produto é dita um problema de configuração de LPS e envolve restrições de inclusão, exclusão e outras associadas a recursos de desenvolvimento, que devem ser respeitadas.

A configuração de LPS é um problema complexo da Engenharia de Software pois o número de combinações possíveis (produtos) cresce exponencialmente com relação ao número de características. Assim, uma alternativa para sua resolução é a modelagem como um problema de otimização e a resolução através de técnicas de busca. Por esse motivo o problema de configuração de LPS tem sido recentemente tema de pesquisa no campo de SBSE (Search Based Software Engineering) [21]. Entretanto, esse problema é influenciado por diversos fatores e existem diferentes critérios a serem satisfeitos pelos produtos. Portanto, foi realizada uma busca por trabalhos sobre o tema de configuração e teste baseados em busca de LPS a fim de se levantar o que tem sido feito/proposto pelos trabalhos existentes e apontar novas direções de pesquisa para melhor lidar com este problema. Os resultados dessa busca estão detalhados em [18].

Além disso, na literatura, o recente trabalho de Harman et al. [20] mostra um crescente número de publicações com abordagens baseadas em busca na área de LPS nos últimos três anos, o que enfatiza a importância do tema. Observa-se analisando ambos trabalhos [18, 20] que os principais objetivos da configuração baseada em busca de LPS são: evolução, customização e teste de LPS.

Neste capítulo um resumo desses principais trabalhos é apresentado. Maior ênfase é

dada em trabalhos baseados em busca para o teste de LPS, de especial interesse para essa dissertação. Eles são detalhados na Seção 5.3.

5.1 Evolução de LPS

O trabalho de Karimpour e Ruhe [29] apresenta soluções para a evolução de LPS através da adição de novas características. Isso ocorre através de métricas que consideram valores atribuídos às características em equilíbrio com a integridade dos produtos. A representação utilizada para o problema utiliza uma string binária. Nesse caso, entretanto, para cada característica uma string do tamanho do número de produtos é adicionada. Assim, se a característica F1 possui uma string (1,0,0), por exemplo, significa que tem-se três produtos e que apenas o primeiro deles possui a característica. O algoritmo utilizado para encontrar ótimas configurações de produtos é o NSGA-II. O contexto do trabalho de Sanchez et al. [48] consiste na geração de produtos em tempo real. Para isso os autores propõem um algoritmo para a seleção de uma configuração que otimize métricas de qualidade. Inicialmente, os sistemas de configuração são representados por diagramas de características definidos em duas tuplas (S,D), sendo que S representa o conjunto de características selecionadas e D o conjunto de características não-selecionadas.

5.2 Customização de LPS

Os trabalhos apresentados por [49, 50, 51] propõem a otimização da seleção de características para customização de produtos. Nesses trabalhos os autores visam a obter produtos obedecendo restrições entre as características e minimizando violações de regras no FM, custos e defeitos associados a uma característica. Os trabalhos utilizam a mesma função de fitness que consiste na maior corretude (menor quantidade de violações de regras), maior quantidade de características, menor quantidade de características que não foram utilizadas anteriormente, menor número de defeitos e menor custo. A representação dos FMs é feita através de strings binárias em que o número de bits equivale ao número de características. Se o valor do bit for verdadeiro a característica é selecionada,

caso contrário a característica é removida. São utilizados vários algoritmos genéticos destacando-se principalmente a superioridade do algoritmo IBEA sobre algoritmos mais conhecidos como NSGA-II e SPEA2.

Guo et al. [19] propõem em seu trabalho uma abordagem chamada GAFES que utiliza algoritmos genéticos para otimizar a seleção de características de uma LPS. A representação do Diagrama de Características segue o mesmo padrão adotado por [49, 50, 51]. Os autores propõem um algoritmo baseado em algoritmo genético chamado FesTransform que repara uma seleção de característica que violou as restrições de um diagrama. A abordagem GAFES utiliza a combinação de uma seleção aleatória de características com o algoritmo FesTransform para obter uma população inicial. Ainda no contexto de customização, White et al. [60] apresentam uma abordagem que provê três contribuições no estudo de configuração multi-passo para LPS. Configuração Multi-Passo consiste em um problema que envolve a transição de uma configuração inicial, através de de uma série de configurações intermediárias, para uma configuração final que atende a um conjunto de requisitos desejados. A representação do problema engloba entre os principais objetivos o custo de troca de configuração, o número máximo de passos no problema de configuração, os conjuntos de configuração no início e no fim da configuração e é representada no formato CSP (Constraint Satisfaction-Problem). A resolução do problema é feita através de algoritmos resolvedores CSPs. Outros trabalhos relacionados a configuração de LPS [11, 36, 42] abordam diferentes objetivos como custo, preferências de usuário e de tomadores de decisões assim como violações das regras dos FMs. Abordagens de seleção multiobjetivo são comparadas no trabalho de Olaechea et al. [39] que utiliza o algoritmo IBEA em comparação com o Guided Improvement Algorithm (GIA) para a satisfação de custo e consumo de memória.

Os trabalhos mais relacionados a esta dissertação são os que envolvem teste de linha de produto de software. Esses trabalhos são descritos na próxima seção.

5.3 Teste de LPS

O objetivo do trabalho de Henard et al. [22] é a priorização de casos de teste que consiste na organização dos casos de teste de tal maneira que defeitos sejam descobertos o quanto antes. Esse trabalho possui como objetivo principal uma abordagem baseada em similaridade para a geração e seleção de produtos para o teste de grandes LPS através do critério *t-wise*. Similaridade, nesse contexto, é uma heurística usada para comparar dois produtos. Os resultados apresentados pelo trabalho sugerem que dois produtos dissimilares são mais propensos a cobrir um maior número de t-conjuntos válidos (conjuntos criados pelo critério *t-wise*) do que dois produtos similares. O processo inicia com a utilização do Algoritmo Evolutivo (1+1) para gerar produtos válidos em que a similaridade é utilizada como fitness para a seleção dos melhores indivíduos (os mais dissimilares entre si). O cálculo do fitness é realizado de maneira a priorizar um conjunto de produtos com maior capacidade de cobrir t-conjuntos. Dois algoritmos são utilizados para gerar essa priorização de conjuntos no cálculo, são eles *Greedy Priorization* e *Near Optimal Priorization*. De maneira geral, o processo é dado pela geração de produtos válidos e pela seleção dos conjuntos de produtos para teste (priorização dos casos de teste).

Em outro trabalho, Henard et al. [24] mantém o tema de teste de LPS. Entretanto, nesse trabalho os autores formulam um problema multiobjetivo que visa um conjunto final de produtos com a maior cobertura *pairwise* e o mínimo número de produtos e custo. Para esse fim os autores estabelecem o uso de um algoritmo genético, propondo a representação para os indivíduos bem como operadores genéticos de mutação e recombinação, utilizados nesse tipo de algoritmo. Apesar da possibilidade de existirem produtos inválidos para o problema em questão, nesse trabalho os autores utilizam resolvedores SAT para gerar apenas produtos válidos que são utilizados como espaço de busca. Ou seja, a busca por um conjunto final ocorre em um espaço de busca que contém apenas produtos válidos para um diagrama em questão.

Em seu mais recente trabalho Henard et al. [25] abordam o teste baseado em mutação de LPS. Essa abordagem inicia-se criando FM mutantes a partir de FM válido inserido. Então, um processo de busca baseado no Algoritmo Evolutivo (1+1) faz uso dos dia-

gramas FM original e FM mutante para gerar um conjunto de configurações para teste. Seguindo os passos da abordagem proposta, a primeira etapa cria versões mutantes do FM original que contém defeitos inseridos nos FM. A próxima etapa consiste em, a partir dos mutantes criados, e através de um processo baseado em busca, gerar e minimizar conjuntos de configurações para teste. Os autores propõem os elementos utilizados no algoritmo genético como individuo, representado por um conjunto de configurações de teste; população, representada por um único indivíduo e avaliação de fitness, baseada no escore de mutação de cada configuração para teste do indivíduo.

A minimização de casos de teste é tema do trabalho de Wang et al. [58] que utilizam algoritmos genéticos baseados em uma função de agregação (*Weight-Based GAs*) dos seguintes fatores: número de testes, habilidade em revelar defeitos, cobertura *pairwise*. Diferentes configurações do AG são avaliadas considerando diferentes pesos na função de agregação. São utilizados três diferentes algoritmos que tratam esses pesos de diferentes formas: WBGA (pesos pré-definidos), WBGA-MO (possui um conjunto de pesos para seleção) e RWGA (pesos aleatórios normalizados). O objetivo principal é encontrar um subconjunto mínimo de casos de teste, do conjunto total de casos de teste de uma LPS, para testar um determinado produto. Ao mesmo tempo procura-se conseguir um alto nível de cobertura *pairwise* e um alto nível na detecção de defeitos.

Em outro trabalho mais recente, Wang et al. [59] abordam a priorização multiobjetivo de casos de teste no contexto de LPS. Os autores abordam um caso de estudo real. Os autores formularam o problema de otimização com quatro objetivos: custo total para alocação de recursos de teste, número de casos de teste, cobertura *pairwise* e capacidade de detecção de defeitos dos casos de teste. Nesse trabalho, são comparados diferentes tipos de algoritmos na resolução do problema em questão. O primeiro deles, AVM (Alternating Variable Method) representa os algoritmos de busca local. O Algoritmo Genético (AG) foi selecionado para ser o algoritmo de busca global. O último algoritmo utilizado nesse trabalho, Algoritmo Evolutivo (1+1) é mais simples que o GA mas pode ser mais eficiente em alguns casos.

Ensan et al. [15] também apresentam o problema de minimização de casos de teste

para LPS. Segundo [15] o esboço da abordagem consiste em primeiro gerar aleatoriamente um conjunto de configurações (aplicações) do diagrama de características, sem considerar restrições extras, considerando somente o fato de que as configurações devem ser válidas. Essas configurações são utilizadas como população inicial do algoritmo genético. Depois de gerada a população inicial, cada configuração é avaliada de acordo com a função de *fitness*. A função de fitness utilizada leva em consideração duas métricas: cobertura de variabilidades e complexidade ciclomática. A primeira refere-se ao total de pontos de variabilidade delimitados para um determinado produto. A segunda refere-se ao número de restrições de integridade que são aplicadas no processo de derivação de produtos de um determinado produto. Posteriormente as configurações mais "fracas" são descartadas enquanto as demais configurações são preservadas para serem usadas como sementes para a mutação e a recombinação. Os autores ressaltam que, como as configurações geradas são baseadas em configurações anteriores elas não são necessariamente válidas; por isso, a próxima etapa do processo consiste na validação das configurações geradas, restando apenas as válidas para o processo. Essas etapas são repetidas até que seja satisfeita a condição final. No final do processo a população final do algoritmo genético é considerada o conjunto de testes e cada uma das configurações consiste em um teste que pode ser usado como uma aplicação de exemplo, a ser testada para o teste de LPS.

Lopez-Herrejon et al. [34] abordam a otimização multiobjetivo de conjuntos de teste para o teste *pairwise*. Segundo os autores o objetivo do trabalho consiste em minimizar o número de produtos de teste e maximizar a cobertura *pairwise*. Para isso, estabelecem um valor fixo para o número de produtos de teste e uma relação com o número de características do FM. Os autores propõem um algoritmo para encontrar a fronteira de Pareto, esse algoritmo tem como entrada um FM válido e como saída a fronteira.

5.4 Discussão

Nessa seção são abordados os aspectos relevantes encontrados nos trabalhos e identificadas algumas tendências e oportunidades de pesquisa. Essa análise baseia-se na Tabela 5.4.

Observa-se que todos os trabalhos relacionados utilizam algoritmos evolutivos. Isto

Tabela 5.1: Trabalhos Relacionados - Teste de LPS

Referência	Autores	Coluna	Algoritmos	Fitness
[22]	C. Henard, M. Papadakis, G. Perrouin, J. Klein, P. Heymans e Y. Le Traon	Bypassing the Combinatorial Explosion: Using Similarity,to Generate and Prioritize T-wise Test Suites for Large,Software Product Lines	Adaptação do Algoritmo Evolutivo (1+1)	Similaridade entre soluções
[24]	C. Henard, M. Papadakis, G. Perrouin, J. Klein e Y. Le Traon	Multi-objective Test Generation for Software Product Lines	MOGA	Cobertura Pairwise, Número de produtos, Custo de Teste
[25]	C. Henard, M. Papadakis and Y. Le Traon	Mutation-Based Generation of Software Product Line Test Configurations	Baseado no Algoritmo Evolutivo (1+1)	Escore de mutação
[58]	S. Wang, S. Ali e A. Gotlieb	Minimizing Test Suites in Software Product Lines Using Weight-based Genetic Algorithms	WBGA, WBGA-MO e RWGA	Número de testes, habilidade em revelar defeitos e cobertura pairwise.
[59]	S. Wang, D. Buchmann, S. Ali, A. Gotlieb, D. Pradhan e M. Liaaen	Multi-objective Test Prioritization in Software Product Line Testing: An Industrial Case Study	Algoritmo Evolutivo (1+1), Algoritmo Genético e AVM	Custo total de recursos para teste, número de casos de teste, cobertura pairwise e capacidade de detectção de defeitos de casos de teste
[15]	F. Ensan, E. Bagheri e D. Ga{s}evi{c}	Evolutionary Search-based Test Generation for Software Product Line Feature Models	Algoritmo Genético	Cobertura de variabilidades e Complexidade Ciclomática
[34]	R. E. Lopez-Herrejon, J. F. Chicano, J. Ferrer, A. Egyed e E. Alba	Multi-objective Optimal Test Suite Computation for Software Product Line Pairwise Testing.	Algoritmo multiobjetivo baseado no resolvedor SAT	Número de Produtos e Cobertura pairwise

demonstra a popularidade destes algoritmos. Com relação a otimização, dois trabalhos utilizam apenas um objetivo e são portanto mono-objetivo. A grande maioria engloba mais de um objetivo, entretanto, esses objetivos não são otimizados individualmente, eles são agregados em um único objetivo que é otimizado. Dessa maneira, apenas o trabalho de Lopez-Herrejon et al. [34] utiliza a otimização multiobjetivo visando a otimização simultânea desses diferentes objetivos interdependentes e muitas vezes conflitantes.

As funções de *fitness* utilizadas estão associadas a diferentes métricas. São elas similaridade entre soluções, custo de teste, escore de mutação, número de testes, habilidade de revelar defeitos, recursos para teste, custo total de recursos, cobertura de variabilidades, complexidade ciclomática. Além dessas, destacam-se em diferentes trabalhos métricas como cobertura *pairwise* e número de produtos.

Com a finalidade de contribuir com novas investigações e com o campo de pesquisa, algumas oportunidades/tendências foram identificadas e resumidas a seguir.

- Apenas um trabalho apresenta a otimização simultânea de objetivos conflitantes no teste de LPS, o que mostra ser necessário o estudo de abordagens multiobjetivo;

- Apenas um trabalho utiliza o critério de teste de mutação. Entretanto, essa abordagem não utiliza operadores voltados para as classes de defeitos que podem ocorrer em

uma LPS. Dessa maneira, torna-se possível a utilização dos operadores de mutação e da ferramenta proposta por Ferreira [17] em uma abordagem multiobjetivo;

- A maioria dos trabalhos utiliza algoritmos evolutivos mono-objetivos. Portanto, a utilização de algoritmos conhecidos de outras áreas do teste e configuração de LPS como NSGA-II, SPEA2 e IBEA em um problema multiobjetivo torna-se um importante tópico de pesquisa;

- Apesar de vários trabalhos utilizarem a métrica *pairwise* e apenas um trabalho utilizar a métrica baseada em mutação de FMs, nenhum trabalho utiliza essas duas métricas em conjunto como objetivos de uma mesma otimização. O único trabalho que considera o teste de mutação e o número de mutantes cobertos não considera outros fatores tais como tamanho do conjunto de teste ou cobertura de outros critérios como *pairwise*.

5.5 Considerações Finais

Como exposto, diferentes trabalhos na literatura remetem ao tema de teste baseado em busca de LPS. Existem diferentes abordagens, com diferentes propósitos, utilizando tipos de técnicas diferentes. Entretanto, em nenhum desses trabalhos, métricas como a minimização de mutantes vivos, minimização de produtos para teste e cobertura *pairwise* foram considerados em conjunto, o que proporciona espaço para pesquisa nessa área. Dentre os mais diversos resultados percebe-se a popularidade de algoritmos evolutivos. Entretanto, algoritmos multiobjetivos tais como NSGA-II, SPEA2 e IBEA, não foram utilizados para geração de dados para o teste de mutação.

Considerando este fato e as discussões apresentadas na Seção 5.4, esse trabalho apresenta uma abordagem que visa a obter a maior cobertura do teste de mutação, com um tamanho mínimo de produtos e ao mesmo tempo atender outros critérios como *pairwise*. Os próximos capítulos detalham a abordagem proposta, a implementação e os resultados obtidos através da aplicação dessa abordagem.

CAPÍTULO 6

ABORDAGEM PROPOSTA

Esse capítulo aborda aspectos gerais da abordagem proposta e detalhes específicos da implementação desse trabalho. Essa abordagem possui como objetivo a geração de conjuntos de produtos, que são conjuntos de dados de teste para satisfazer o teste de mutação de FMs. O objetivo é conseguir altos valores de cobertura com custo mínimo, dado pelo número de dados de teste. Essa tarefa é complexa e pode ser resolvida eficientemente por algoritmos de otimização multiobjetivo.

De acordo com Harman et al. [21] para implementar uma solução baseada em busca para um problema, são necessários os seguintes ingredientes: i)uma representação adequada da solução, que precisa ser representada de uma maneira que possa ser manipulada pelo algoritmo; ii) operadores de busca para melhorar as soluções e explorar o espaço de busca; e iii) uma maneira adequada de avaliar a qualidade das soluções, ou seja, a função de *fitness*. Os ingredientes da abordagem proposta são descritos neste capítulo.

6.1 Representação da População

Definiu-se indivíduo como um conjunto de tamanho aleatório de produtos gerados. Isto porque cada indivíduo representa uma solução (conjunto de produtos) para um problema em questão. A representação do produto consiste em um vetor booleano de tamanho igual ao número de características da LPS em teste, sendo que "1" representa a característica selecionada e "0" a não selecionada. Com base na LPS AGM apresentada na Seção 2 a Figura 6.1 apresenta uma síntese da representação de um possível indivíduo gerado.

Através da Figura 6.1 é possível visualizar o indivíduo X com três produtos (1, 5, 11), também é possível verificar o vetor de características criado a partir do FM, representado através das características AGM, services, rules, configuration, action, play, pause, save, brickles, bowling, pong, moviment, collision. Cada produto do individuo é descrito através

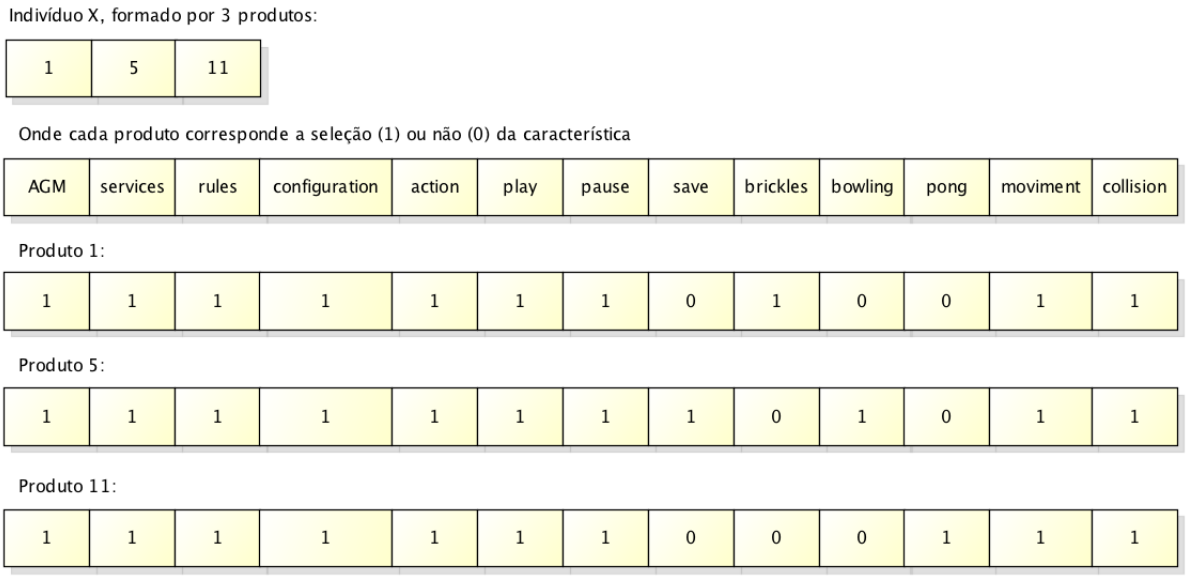

Figura 6.1: Representação adotada para o indivíduo

de um vetor booleano que indica a presença (1) ou ausência (0) da característica de tal posição. Por exemplo, o produto 1 contém todas as características exceto save, bowling e pong.

6.2 Operadores Genéticos

Com a finalidade de inserir diversidade na população utilizada foram propostos operadores de recombinação e mutação.

6.2.1 Operador de Recombinação

O operador de recombinação proposto chama-se ProductCrossover e trabalha com a recombinação entre dois indivíduos. Para realizar a recombinação são selecionados dois indivíduos através do operador de seleção BinaryTournament2 (Seção 6.2.3) implementado pelo jMetal. A partir disso, caso os indivíduos possuam tamanho maior que 1 e tenham número par de elementos, ambos são divididos utilizando o tamanho do indivíduo dividido por 2. Assim, se um individuo tem tamanho 8, a primeira parte possui os 4 primeiros elementos e a segunda possui os 4 últimos. Caso o tamanho seja um número impar, na divisão por 2 é considera a parte inteira para a primeira parte e para segunda parte

considera-se o total menos a primeira parte. Por exemplo, se um individuo tem tamanho 7 a primeira parte possui os 3 primeiros elementos e a segunda parte os 4 últimos. Após a divisão, a primeira parte do individuo 1 se junta com a segunda parte do individuo 2. De maneira análoga, a segunda parte do indivíduo 1 se junta com a primeira parte do indivíduo 1. Lembrando-se que essa junção é uma união, ou seja, os elementos iguais são descartados. Ao final do processo formam-se dois filhos. A Figura 6.2 apresenta um exemplo de aplicação desse operador.

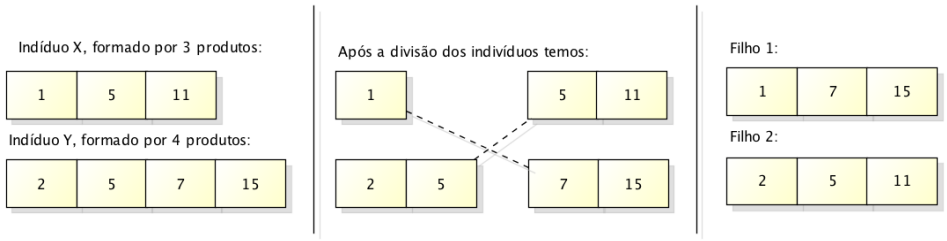

Figura 6.2: Esquema de cruzamento de pais com tamanho maior que 1

Caso um dos pais possua tamanho igual 1 a divisão em partes é feita somente no pai com tamanho maior que 1. A recombinação é feita unindo a única parte do indivíduo a cada uma das duas partes do outro indivíduo. Formam-se dessa maneira dois filhos distintos. Caso os dois indivíduos possuam tamanho igual a 1, optou-se por não aplicar o operador, devido à impossibilidade de se formarem dois filhos distintos. Um exemplo de aplicação do ProductCrossover nesse caso é apresentado na Figura 6.3.

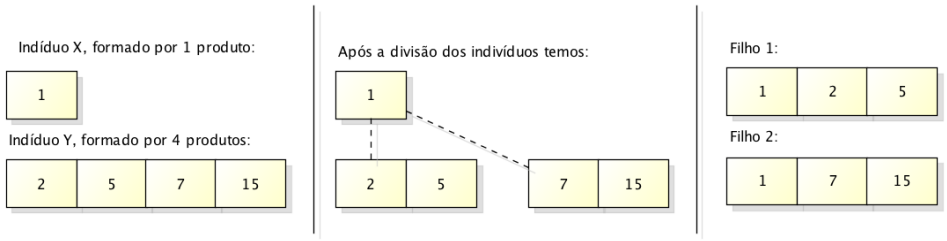

Figura 6.3: Esquema de recombinação para pai com tamanho igual a 1

6.2.2 Operador de Mutação

O operador de mutação proposto chama-se ProductMutation e trabalha com três diferentes tipos de mutação: adição, remoção e troca. Na adição um produto aleatório é

selecionado do conjunto de produtos gerados e se não existir no indivíduo é adicionado. Na remoção um produto aleatório do indivíduo é selecionado e removido. Já na troca seleciona-se um produto aleatório, se o produto não existir no individuo então seleciona-se aleatoriamente um produto do indivíduo e efetua-se a troca. A Figura 6.2 apresenta exemplos de aplicação desse operador.

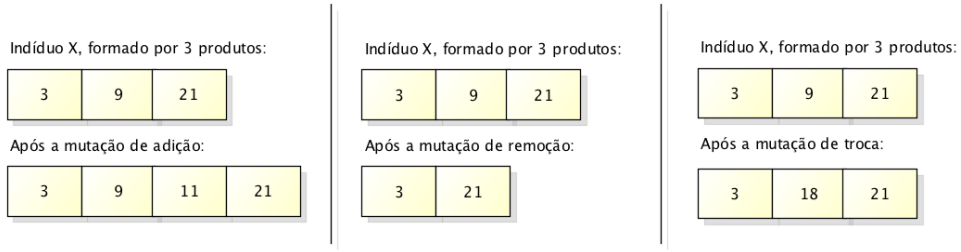

Figura 6.4: Esquema de mutação do tipo adição, remoção e troca

6.2.3 Operador de Seleção

A seleção dos melhores indivíduos se dá através do operador BinaryTournament2, um tipo de torneio binário, implementado pelo jMetal [13]. Nesse tipo de seleção os dois melhores indivíduos de uma população são selecionados para a próxima geração [7]. Caso ocorra a recombinação, os dois indivíduos são recombinados entre si gerando dois novos filhos. Caso haja mutação, a mesma ocorre nos indivíduos selecionados.

6.3 Função de *Fitness*

Foram definidos três objetivos para o cálculo de *fitness* do indivíduo (Δ). Esses objetivos foram baseados nos trabalhos relacionados. Todos os objetivos foram modelados como minimização, de acordo com o funcionamento do framework jMetal 6.5. O primeiro deles corresponde ao número de produtos presentes em um indivíduo (S_Δ). Esse número é representado através da razão entre o número de produtos presentes no indivíduo (n_Δ) e o número total de produtos válidos gerados sendo considerados (nt), conforme a Equação 6.1. Nesse objetivo, deseja-se obter o menor número de produtos possível para o indivíduo e que atenda concomitantemente os demais objetivos de maneira satisfatória.

$$S_{\Delta} = \frac{n_{\Delta}}{nt} \tag{6.1}$$

O segundo objetivo corresponde à cobertura dos diagramas mutantes gerados. Um diagrama mutante é dito morto (coberto) por um determinado produto quando o mesmo é considerado válido para o diagrama original e inválido para o diagrama mutante, ou válido para o diagrama mutante e inválido para o diagrama original. Nesse objetivo deseja-se minimizar a quantidade de mutantes deixados vivos. Como o escore de mutação corresponde ao percentual de cobertura, o valor a ser minimizado corresponde a um menos o escore de mutação.

O número de mutantes vivos (A_{Δ}) corresponde à relação entre o número de mutantes mortos (DM_{Δ}) e o número total de mutantes ativos AM. Esse total é obtido descartando os mutantes não válidos e aqueles que produzem um mesmo conjunto de produtos quando comparados com o FM original.

$$A_{\Delta} = 1 - \frac{DM_{\Delta}}{AM} \tag{6.2}$$

O terceiro objetivo para o cálculo do *fitness* representa a cobertura dos pares gerados pela técnica *pairwise*. Um par é dito coberto quando determinado produto possui as duas características presentes em um par. Nesse objetivo deseja-se minimizar a quantidade de pares não cobertos. Para isso, reduz-se de um o valor encontrado para os pares em questão.

A cobertura de *pairwise* P_{Δ} corresponde à relação entre o número de pares cobertos pelo individuo PC_{Δ} e o número total de pares válidos P. O cálculo da cobertura é apresentado na Equação 6.3.

$$P_{\Delta} = 1 - \frac{PC_{\Delta}}{P} \tag{6.3}$$

6.4 Aspectos de Implementação

6.4.1 Aspectos Gerais

De maneira geral a abordagem proposta funciona da seguinte maneira: primeiramente um FM é dado como entrada juntamente com os parâmetros da execução como: tamanho da população, número máximo de avaliações, tamanho da população auxiliar, taxa de recombinação, taxa de mutação, número máximo de execuções, algoritmo utilizado e número de produtos que serão gerados (quando necessário), relativo a nt. Devido a uma limitação do framework FaMa, para FMs com um grande número de características não é possível obter todas as combinações de produtos válidos. Nesses casos é necessário delimitar o número máximo de produtos que se deseja utilizar através do parâmetro descrito. Caso o parâmetro inserido seja "0"(zero), todo o conjunto de produtos válidos para o FM é gerado. Nos experimentos descritos neste trabalho (Capítulo 7), em particular, utilizou-se a geração de todos os produtos válidos, sem delimitação de número de produtos através do parâmetro, pois as LPSs utilizadas possuem pequeno número de características.

Independente da quantidade de produtos utilizada, se total ou delimitada por parâmetro, todos os produtos gerados formam um conjunto de produto válidos que são utilizados como total (nt). Após essa etapa são gerados os diagramas mutantes e os pares para o diagrama de entrada. O próximo passo na abordagem proposta é verificar quais produtos matam quais mutantes e quais produtos cobrem quais pares. Após essa etapa uma tabela temporária é criada vinculando cada produto aos seus respectivos mutantes mortos e pares cobertos.

Com base no conjunto de produtos válidos gerado o processo de otimização multiobjetivo é iniciado. Os indivíduos da população inicial são gerados com base na seleção de um número aleatório de produtos do conjunto em questão. A partir de então o processo evolutivo se inicia, passando por n gerações (baseado no número máximo de avaliações) com a aplicação dos operadores genéticos propostos. Para cada geração um valor de *fitness* é atribuído aos indivíduos, valor esse calculado com base na matriz temporária criada anteriormente. Ao final da execução obtêm-se conjuntos de produtos que atendem

ao problema em questão com diferentes valores de *fitness*. Esses e outros aspectos da abordagem proposta, assim como aspectos específicos da implementação são abordados a seguir.

6.4.2 Integração com jMetal

Para a implementação da abordagem utilizou-se a linguagem de programação Java pela facilidade de integração com as ferramentas e frameworks utilizados. Utilizou-se como ambiente de desenvolvimento a ferramenta Eclipse IDE versão Luna 4.4.1. Codificou-se a integração do framework JMetal, responsável por toda a estrutura dos algoritmos e da formulação do problema com a ferramenta *Feature Mutation Test Suite* (FMTS) proposta por Ferreira [17].

Inicialmente utilizou-se o framework JMetal para a formulação de um problema multiobjetivo. A Figura 6.5 apresenta o diagrama de classes simplificado do JMetal. A partir da análise do diagrama verifica-se que a estrutura para formulação de um problema determina um tipo de solução (SolutionType) que por sua vez define um tipo de variável (Variable). Baseado na estrutura apresentada, desenvolveu-se a classe DataGenerationProblem, correspondente ao tipo de problema, responsável por armazenar os principais parâmetros referentes a execução dos experimentos como número de objetivos, número de variáveis e tipo de solução. Além disso, a classe é responsável pelos cálculos de fitness (Seção 6.3) das soluções encontradas através da função *evaluate()*. Desenvolveu-se também, ainda na estrutura do JMetal, a classe ProductArraySolutionType, que representa um array de variáveis. Sua principal função *createVariables()* gera, com o auxílio de outra classe implementada chamada ProductVariable, os indivíduos utilizados no problema. Essa última classe é a responsável pela representação dos indivíduos utilizados (cromossomos) e sua população com produtos (genes).

A representação do indivíduo seria feita inicialmente através de um vetor de vetores booleanos onde cada vetor booleano representaria um produto. Entretanto, esse tipo de estrutura prejudicaria a manipulação dos indivíduos no código implementado além de exigir mais recursos para processamento. Assim, como mencionado, optou-se por

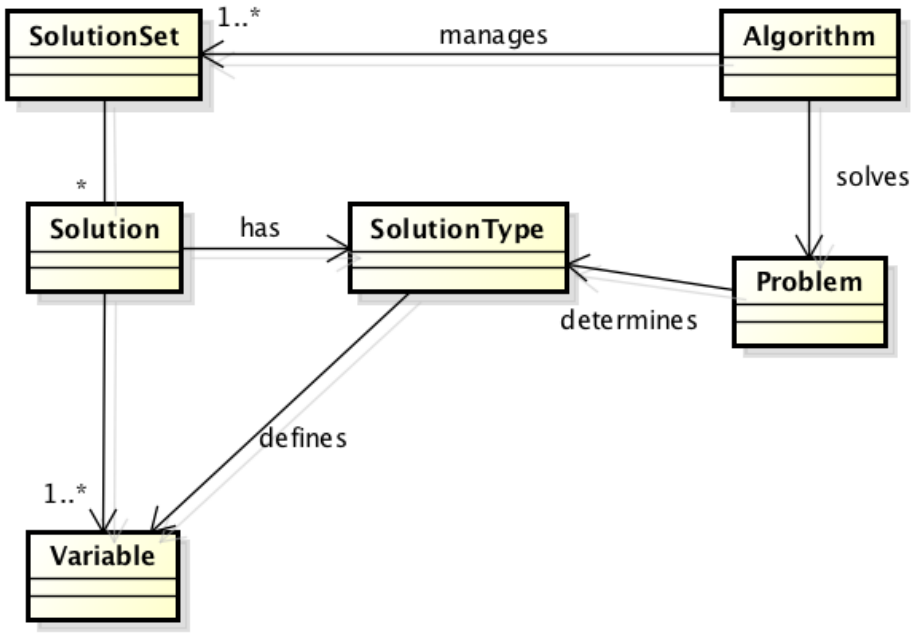

Figura 6.5: Diagrama de Classes simplificado do JMetal (Adaptado de [13])

uma representação do indivíduo que consiste em um vetor de inteiros em que cada inteiro representa um produto. O armazenamento (inclusão, retorno) desses produtos é realizado através de outra classe desenvolvida chamada ProductSet. Nessa classe implementou-se um Hash em que cada inteiro associa-se a um produto. Dessa maneira, quando necessário realizar operações entre os indivíduos (operação de recombinação, por exemplo) necessita-se apenas trabalhar com vetores de inteiros, o que simplifica a estrutura.

Para a execução selecionaram-se os algoritmos NSGA-II, SPEA2 e IBEA (Seção 4.3), todos implementados no framework JMetal. Apesar dos algoritmos estarem corretamente implementados foi necessária uma pequena alteração para que pudessem utilizar os operadores de crossover e mutação propostos. Precisou-se alterar também as classes CrossoverFactory e MutationFactory, utilizadas pelos algoritmos na estrutura do JMetal, para que englobassem os operadores. A partir de então pode-se referenciar ProductCrossover e ProductMutation como parâmetros para qualquer algoritmo implementado.

6.4.3 Integração com FMTS

Ferreira [17] define FTMS como uma ferramenta de auxílio para automatizar o teste de mutação de LPS. Segundo o autor a ferramenta possibilita que para um FM seja possível

realizar o processo de teste de mutação através da aplicação dos operadores propostos e geração de diagramas mutantes, geração de conjunto de produtos e avaliação do conjunto gerado. Além disso, possibilita a avaliação de conjuntos de casos de teste, a otimização desses conjuntos e a identificação de diagramas equivalentes. Para mais informações sobre o processo e a ferramenta proposta por Ferreira consulte a Seção 3.3.2.

Aspectos técnicos da ferramenta englobam a divisão em pacotes de sua implementação. A Figura 6.6 apresenta o diagrama dos pacotes utilizados na FTMS. O pacote *FeatureModel* contém classes utilizadas para a modelagem do diagrama de características utilizado. O pacote *inOut* é responsável pela leitura e escrita dos diagramas de características no disco em formato XML. O *testFramework* é responsável pela geração dos mutantes, geração dos produtos e execução dos testes. Nesse pacote também são realizadas as validações de diagramas, geração de produtos iniciais, otimização de resultado e o cálculo do escore de mutação. O último pacote, *userInterface*, implementa funções de interação entre a aplicação e o usuário.

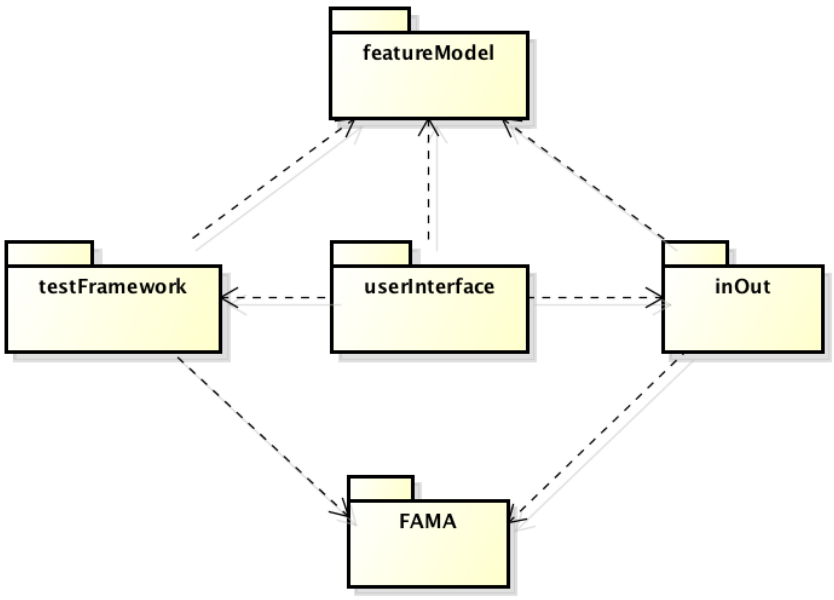

Figura 6.6: Diagrama de pacotes da ferramenta FMTS (Adaptado de [17])

Com a finalidade de atualizar a ferramenta em questão algumas alterações foram feitas. Para que a funcionalidade de geração de dados para o teste de mutação de LPS fosse adicionada, incluíram-se dois pacotes adicionais a estrutura. O primeiro deles corresponde ao *combTool* responsável por toda a estrutura necessária para a execução do

algoritmo AETG, utilizado na geração dos pares da métrica pairwise. O segundo pacote *jMetal* corresponde a toda estrutura do framework, e engloba as novas classes implementadas: DataGenerationProblem, ProductArraySolutionType, ProductVariable bem como os operadores de mutação (ProductMutation) e crossover (ProductCrossover). A Figura 6.7 apresenta o diagrama dos pacotes atualizados da FMTS.

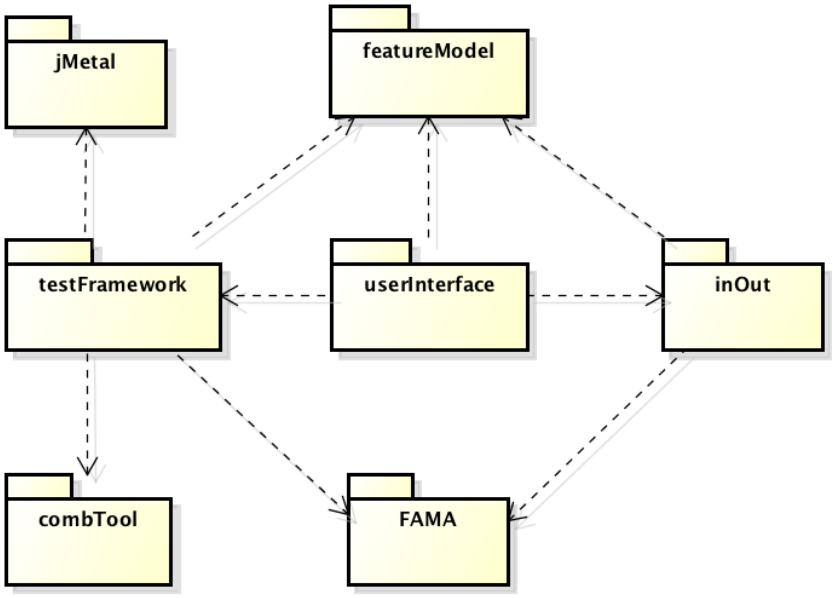

Figura 6.7: Diagrama de pacotes atualizado da ferramenta FMTS

Além disso, implementaram-se algumas classes dentro dos pacotes já existentes na FMTS. A classe PairsGenerator foi adicionada ao pacote *testFramework*. Sua principal função, CreatePair(), é utilizada na criação dos pares de características. Para a criação, o pacote *combTool* é acionado através da utilização do algoritmo AETG que efetivamente cria os pares através de combinações matemáticas. Como a definição de *pairwise* estabelece que cada par de característica deve ser testado apenas uma vez, após a criação são descartados os pares repetidos.

Outra classe implementada nesse pacote chama-se ValidPair. Através de sua função isValid() é possível verificar se um par criado através do AETG é válido para o diagrama de características em questão. Em outras palavras, verifica se o par criado não viola nenhuma restrição de exclusão do diagrama. A estrutura do par foi adicionada na classe Pair, incluída no pacote *featureModel*. Sua principal função, add(), adiona duas características de um diagrama de características ao objeto par criado.

A classe ProductSet, implementada no pacote *testFramework*, concentra algumas funcionalidades. A primeira delas é armazenar o conjunto de mutantes mortos e pares cobertos através das funções addDeadMutants() e addCoveredPairs(). A segunda funcionalidade é o cálculo dos valores de A_Δ e P_Δ. A função mutationValue() calcula o A_Δ obtido por um indivíduo. De maneira análoga a função pairValue() calcula o P_Δ obtido por um indivíduo.

Durante a execução dos experimentos iniciais verificou-se a geração de um grande número de arquivos temporários que não eram excluídos. Dessa maneira, a memória da máquina que executava os experimentos era inundada por arquivos desnecessários. Para resolver esse problema implementou-se a classe Dustman. A principal função dessa classe, clean(), é responsável por acessar o arquivo temporário utilizado pelo sistema operacional em execução e excluir os arquivos temporários gerados no processo de resolução do problema. Dessa maneira, evita-se que a memória temporária do computador em uso fique cheia.

Segundo Ferreira [17], devido a simplicidade da aplicação a forma de interação com o usuário escolhida foi o console. Através de comandos pré-determinados é possível ao usuário: realizar a leitura de um diagrama para a aplicação, gerar diagramas mutantes, calcular os valores A_Δ e P_Δ, visualizar produtos e mutantes gerados, verificar diagramas equivalentes e refinar os resultados dos testes. Alterou-se o console da ferramenta para que incluísse também a funcionalidade de geração de dados de teste através do comando "gproblem". Ao digitar esse comando seguido dos parâmetros descritos na Seção 6.4.1 o usuário executa uma instância do problema.

De maneira geral, como visto na Seção 3.3.2, a ferramenta FMTS é dividida em três módulos: geração de mutantes, geração de casos de teste (produtos) e a execução dos casos de teste. Com a finalidade de englobar as funcionalidades desse trabalho, adicionou-se o módulo de geração de pares. O funcionamento desse novo módulo é descrito a seguir.

6.4.4 Geração de Pares

O módulo de geração de pares é responsável pela geração dos pares utilizados pelo critério *pairwise*. Um diagrama em teste é inserido no formato XML fornecido pelo FaMa. A partir de então um array de características é criado, com as características pertencentes ao diagrama. A entrada desse módulo consiste nesse array de características. O algoritmo AETG, pertencente ao módulo, gera então todas as possíveis combinações com as características inseridas. A Figura 6.8 apresenta um exemplo de geração de pares. Nesse exemplo a entrada de um *array* com as características {relógio, mostrador, pulseira, analógico, digital} geraria uma combinação de 25 pares distintos apresentados na Tabela 6.1.

Tabela 6.1: Resultado da Geração de Pares

Par	**Características**
1	Relógio - Relógio
2	Relógio - Mostrador
3	Relógio - Pulseira
4	Relógio - Analógico
5	Relógio - Digital
6	Mostrador - Relógio
7	Mostrador - Mostrador
8	Mostrador - Pulseira
9	Mostrador - Analógico
10	Mostrador - Digital
11	Pulseira - Relógio
12	Pulseira - Mostrador
13	Pulseira - Pulseira
14	Pulseira - Analógico
15	Pulseira - Digital
16	Analógico - Relógio
17	Analógico - Mostrador
18	Analógico - Pulseira
19	Analógico - Analógico
20	Analógico - Digital
21	Digital - Relógio
22	Digital - Mostrador
23	Digital - Pulseira
24	Digital - Analógico
25	Digital - Digital

A saída do módulo corresponde ao conjunto de todos os pares de características que não são necessariamente válidos para o diagrama em questão. Exemplos de pares inválidos são:

{relógio, relógio}, {mostrador, mostrador}, {pulseira, pulseira}, {analógico, analógico}, {digital, digital}, pois possuem características duplicadas. Na representação utilizada um produto jamais pode ter características repetidas, dessa maneira um par assim jamais poderia ser coberto. Esses pares inválidos são descartados antes da avaliação. Existem também diferentes pares gerados que cobrem as mesmas características. Um exemplo seriam os pares {relógio, analógico} e {analógico, relógio}. Como a definição de pairwise [40] diz que um par de características deve ser testado uma única vez os pares duplicados são descartados. Além desses casos, existem pares que violam restrições do diagrama (includes e excludes) e por isso são considerados inválidos.

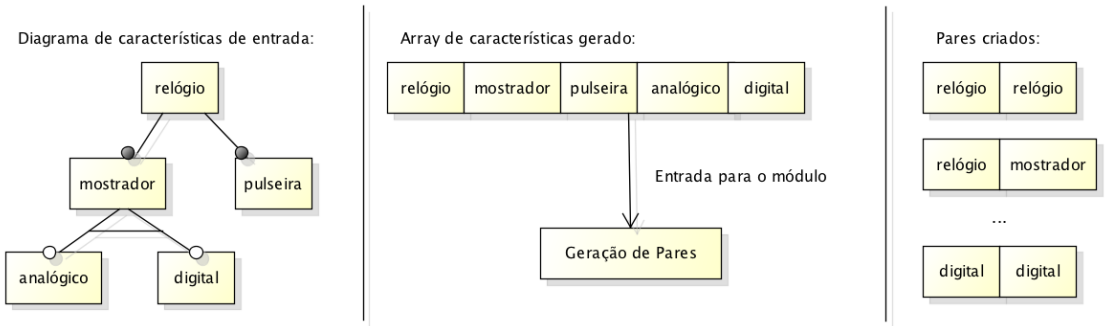

Figura 6.8: Exemplo de geração de pares através do módulo Geração de Pares

O próximo passo no módulo dos pares é o teste de cada par em relação ao diagrama original e aos produtos que devem ser testados. O processo de teste consiste nas seguintes etapas: validação do par em relação ao diagrama original; verificação se um determinado produto possui as características do par. Caso um determinado par seja considerado válido em relação ao diagrama original e o produto em questão possuir suas características, esse par é dito "coberto". A Figura 6.9 apresenta um exemplo do processo de geração de pares e do posterior teste relacionado aos mesmos.

6.5 Considerações Finais

Este capítulo apresentou uma abordagem que utiliza algoritmos evolutivos multiobjetivos para a geração de dados para o teste de mutação de Linha de Produto de Software, bem como seus principais aspectos de implementação.

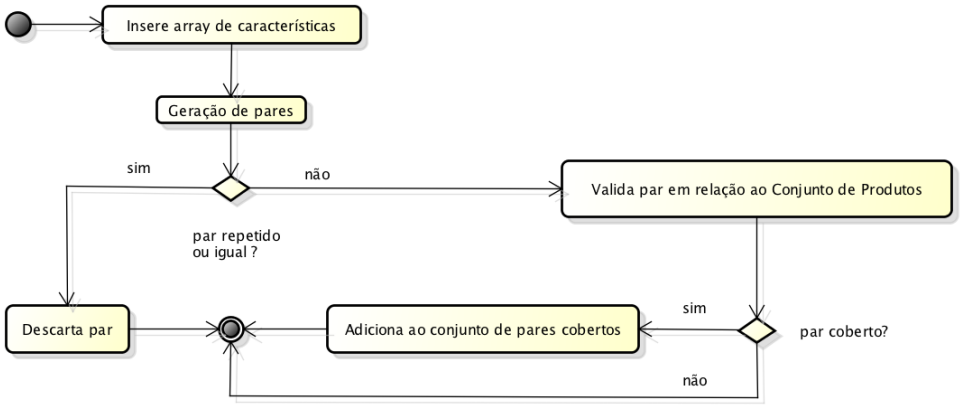

Figura 6.9: Processo de geração e validação de pares

Na abordagem, diferentemente das presentes na literatura, o indivíduo representa um conjunto de produtos (casos de teste). Para esta implementação operadores de busca específicos foram propostos. Três algoritmos foram implementados: NSGA-II, SPEA2 e IBEA, através do *framework* jMetal.

Três funções de *fitness* (objetivos) podem ser usadas: cobertura de *pairwise*, mutantes mortos/vivos e número de casos de teste. Resultados de avaliação dessa abordagem são apresentados no próximo capítulo.

CAPÍTULO 7

AVALIAÇÃO EXPERIMENTAL

Neste capítulo são apresentados os experimentos realizados para a avaliação da abordagem de geração de dados para o teste de mutação de LPS proposta neste trabalho. Primeiramente é descrito como a avaliação foi realizada e depois os resultados são apresentados e analisados.

7.1 LPS Utilizadas

Para a execução dos experimentos foram selecionadas quatro LPS utilizadas anteriormente por Ferreira [17]. A CAS (*Car Audio System*) [47] é uma linha de produto de software para sistemas de som automotivo. JAMES [4] uma linha de produto de software para sistemas web colaborativos. Weather Station [17] representa uma linha de produto de software para sistemas meteorológicos. EShop [52] é uma linha de produto de software para E-commerce.

Linhas de Produto de Software caracterizam-se por propriedades como possuir diferentes características e possuir diferentes tipos de restrições (inclusão e exclusão). A Tabela 7.1 apresenta algumas propriedades como número de características, número de restrições de inclusão e número de restrições de exclusão das LPS utilizadas.

Tabela 7.1: Propriedades das LPS utilizadas

LPS	**# Características**	**# Inclusão**	**# Exclusão**
JAMES	14	1	1
CAS	21	2	1
WeatherStation	22	0	0
EShop	22	1	1

Como visto, o número de características das LPS utilizadas não ultrapassa 22. Por isso, puderam ser utilizados os conjuntos totais de produtos válidos gerados para cada

LPS, não necessitando da especificação do parâmetro referente ao de número limite de produtos, como descrito na Seção 6.4.1.

A Tabela 7.2 apresenta os números de: produtos, mutantes, mutantes ativos e pares válidos.

Tabela 7.2: Números das LPS utilizadas

LPS	**# Produtos** nt	**# Mutantes Gerados** M	**# Mutantes Ativos** AM	**# Pares Válidos** P
JAMES	68	129	106	182
CAS	450	268	227	420
WeatherStation	504	387	357	462
EShop	1152	453	394	462

Do número de mutantes gerados M foram descartados os mutantes equivalentes que derivam o mesmo conjunto de produtos que o FM original. Além disso, para gerar ao final um conjunto de produtos válidos de acordo com o FM original foram descontados mutantes que não podem ser mortos por nenhum produto válido de acordo com o FM. Assim foi gerado AM. Vale a pena ressaltar que a FMTS descarta os mutantes anômalos que não são diagramas bem formados, eles não são contados em M.

Neste trabalho todos os operadores de mutação da FMTS foram aplicados com porcentagem de 100%.

7.2 Questões de Pesquisa

Com o intuito de guiar a avaliação conduzida, as seguintes questões de pesquisa foram propostas:

- **Q1:** *Como são as soluções produzidas pela abordagem proposta em relação aos valores de fitness?* Para avaliar essa questão o conjunto de soluções produzido por cada algoritmo foi considerado. São analisadas as soluções com o melhor *trade-off* (melhores valores de ED) entre os objetivos e associadas aos melhores valores de *fitness*, relativos ao teste de mutação.

- **Q2:** *Como são os resultados produzidos pelos três algoritmos utilizados?* Esta questão avalia qual algoritmo produziu as melhores soluções. Para isso, são uti-

lizados indicadores de qualidade como hypervolume, error ratio (ER) e distância euclidiana (ED), descritos no Capítulo 4.

Para responder as questões de pesquisa foram criados dois experimentos: i) um experimento com dois objetivos (2M), considerando o teste de mutação e número de produtos; e ii) um experimento com três objetivos (3MP), considerando o teste de mutação, o teste *pairwise* e número de produtos. Para ambos os experimentos foram utilizados os operadores de mutação e recombinação propostos. A população inicial foi gerada a partir da seleção de um subconjunto, de tamanho aleatório, do conjunto total de produtos nt. Os algoritmos utilizados foram NSGA-II, SPEA2 e IBEA. Além disso, uma comparação foi feita entre os desempenhos apresentados pelos mesmos. Cada experimento foi executado dez vezes para cada LPS na fase de ajuste de parâmetros, descrita a seguir, e trinta vezes na fase de avaliação usando as melhores configurações.

7.3 Ajuste de Parâmetros

Antes da execução de cada experimento realizou-se um ajuste de parâmetros. Utilizou-se o *tuning* ao invés de parâmetros fixos pois dessa maneira é possível assegurar a comparação entre os melhores desempenhos dos experimentos realizados.

Arcuri e Fraser [2] apresentam uma análise empírica para o *tuning* de parâmetros em SBSE. Entretanto, neste trabalho o número de problemas não é suficiente para seguir as diretrizes propostas pelos autores. Dessa maneira, optou-se por conduzir os experimentos da mesma maneira que os autores conduziram na busca por seus resultados e na formulação de suas diretrizes. Assim, baseado na metodologia utilizada pelos autores, os seguintes parâmetros foram utilizados no *tuning*: i) tamanho da população, ii) número máximo de avaliações, iii) tamanho da população auxiliar (quando necessário), iv) taxa de recombinação, e v) taxa de mutação. Além disso, os parâmetros foram ajustados para cada algoritmo utilizado em cada LPS.

Os valores para o tamanho de população e tamanho de população auxiliar foram definidos também com base nos trabalhos relacionados. Sayyad et. al. [50] e Karimpour

e Ruhe [29] utilizam o valor fixo de população 100. Wang et. al. [58] variam o valor da população ente 100 e 200. Além disso, Arcuri e Fraser [2] afirmam que os valores de tamanho de população encontrados na literatura para SBSE tendem a ser 50, 100 e 200. Dessa maneira, optou-se por utilizar os valores apresentados por Arcuri e Fraser pois englobam os valores utilizados nos demais trabalhos.

A maioria dos trabalhos [29] [50] [58] utiliza uma taxa de mutação de 90%. Dessa maneira, estabeleceu-se o valor em questão como valor máximo e selecionaram-se dois outros valores inferiores de maneira arbitrária sendo eles 10% e 50%. Quanto à taxa de mutação já não há um valor comum. Os autores utilizam diferentes valores e por isso optou-se por variar a mutação na mesma escala da recombinação.

Para o número máximo de avaliações não há consenso na literatura. Sayyad et. al. [50] utilizam valores como 50 mil com 10 execuções e 50 milhões com apenas 1 execução. Karimpour e Ruhe [29], por sua vez, utilizam um valor fixo de 20 mil avaliações. Neste trabalho, como o número de experimentos é grande, optou-se por estabelecer o número máximo de avaliações em 50 mil. De maneira arbitrária definiram-se dois outros valores sendo 10 mil e 30 mil avaliações.

O critério de parada de execução do algoritmo utilizado é o número máximo de avaliações descrito anteriormente. Assim, o número máximo de avaliações influi diretamente no tempo de execução do algoritmo, pois quando o mesmo é atingido a execução é encerrada.

Utilizando os valores estabelecidos para o ajuste de parâmetros criaram-se 81 combinações para o algoritmo NSGA-II que não utiliza população auxiliar e 162 combinações para os algoritmos SPEA2 e IBEA que utilizam população auxiliar. Cada conjunto de combinações foi executado para as 4 LPS selecionadas e para os 3 algoritmos utilizados. Dessa maneira, totalizaram-se 29.160 execuções.

Com o término do *tuning* as melhores combinações de parâmetros foram selecionadas com base nas melhores médias de *hypervolume* [61]. Para aquelas médias de *hypervolume* que não apresentaram diferença estatística pelo teste de Kruskal-Wallis [31] com nível de significância de 5%, a configuração com menor tempo de execução foi selecionada.

A Tabela 7.3 apresenta o conjunto das melhores configurações encontradas para cada LPS/Algoritmo. Essas configurações foram utilizadas para a realização dos experimentos utilizados neste trabalho.

Tabela 7.3: Melhores configurações obtidas com *tuning* de parâmetros

Experimento	LPS	Algoritmo	População	Máx. de Aval.	Pop. Aux.	% Crossover	% Mutação
2M	CAS	NSGAII	50	10.000	Não Possui	10%	50%
		SPEA2	50	10.000	50	90%	10%
		IBEA	50	10.000	50	10%	50%
	JAMES	NSGAII	100	10.000	Não Possui	50%	10%
		SPEA2	50	10.000	50	50%	50%
		IBEA	50	10.000	50	10%	50%
	WeatherStation	NSGAII	50	10.000	Não Possui	10%	10%
		SPEA2	50	10.000	50	10%	90%
		IBEA	100	10.000	100	10%	50%
	EShop	NSGAII	200	10.000	Não Possui	50%	50%
		SPEA2	100	10.000	50	90%	50%
		IBEA	100	10.000	50	10%	10%
3MP	CAS	NSGAII	50	10.000	Não Possui	10%	90%
		SPEA2	200	10.000	50	90%	50%
		IBEA	50	10.000	50	10%	50%
	JAMES	NSGAII	50	10.000	Não Possui	90%	90%
		SPEA2	50	10.000	50	90%	90%
		IBEA	50	10.000	50	50%	50%
	WeatherStation	NSGAII	100	10.000	Não Possui	50%	10%
		SPEA2	200	10.000	50	10%	10%
		IBEA	100	10.000	50	10%	90%
	EShop	NSGAII	200	10.000	Não Possui	10%	50%
		SPEA2	100	10.000	50	90%	90%
		IBEA	50	10.000	50	90%	50%

7.4 Resultados e Discussões

Nesta seção são apresentados os resultados experimentais obtidos. As Subseções 7.4.1 e 7.4.2 apresentam os resultados quantitativos considerando os valores de *fitness* encontrados em cada experimento e comparando-os com seus valores de ER, suas médias de *hypervolume*, seus tempos de execução e os valores de maior e menor ED. Ao final, a Subseção 7.5 apresenta algumas discussões com o intuito de responder às questões de pesquisa apresentadas na Seção 7.2.

7.4.1 Resultados Experimento 2M

A Tabela 7.4 apresenta informações para as fronteiras de Pareto obtidas no experimento 2M. A primeira coluna exibe a LPS utilizada enquanto a segunda apresenta a quantidade de soluções na fronteira real (PF_{true}). No caso desses experimentos, como as PF_{true} são desconhecidas para cada LPS, as mesmas foram formadas a partir da união de todas as soluções não-dominadas (PF_{known}) obtidas nos experimentos de todos algoritmos em relação a cada LPS. As colunas três, quatro e cinco apresentam as quantidades de

soluções nas PF_{known} obtidas para os algoritmos NSGAII, SPEA2 e IBEA em relação ao experimento 2M. Apresentam ainda, separado por uma "/", a quantidade dessas soluções que também estão presentes nos respectivos conjuntos PF_{true}. O ER é apresentado em parênteses. Valores em negrito correspondem aos melhores valores de ER, ou seja, dos elementos do conjunto PF_{known} os que mais apresentaram soluções no conjunto PF_{true}.

Tabela 7.4: Fronteiras de Pareto - Resultados experimento 2M

LPS	**PFtrue**	**PFknown**		
		NSGAII	**SPEA2**	**IBEA**
CAS	17	13/6 (0.54)	**16/9 (0.44)**	15/2 (0.87)
JAMES	8	10/4 (0.60)	10/3 (0.70)	**8/4 (0.50)**
WeatherStation	18	20/8 (0.60)	17/6 (0.65)	**19/8 (0.58)**
EShop	21	20/7 (0.65)	**18/9 (0.50)**	15/6 (0.60)

Ainda em relação a Tabela 7.4, considerando-se o experimento 2M, observa-se que o algoritmo NSGAII obteve em dois dos quatro casos o maior número de soluções no conjunto PF_{known}, enquanto em outro caso empatou no número com o algoritmo SPEA2. Entretanto, para todos esses casos, o valor de ER foi maior em relação aos demais algoritmos. Isso indica que, apesar do maior número de soluções encontradas, poucas delas pertencem a fronteira real. Em outras palavras, as soluções encontradas por esse algoritmo, em sua maioria, são dominadas por melhores soluções.

No caso dos algoritmos SPEA2 e IBEA ambos apresentaram bons desempenhos. Para a LPS CAS o algoritmo SPEA2 teve a maior diversidade de soluções não-dominadas encontradas totalizando 16 soluções. Dessas, 9 estão presentes no conjunto PF_{true}, o que corresponde a um ER de 0.44. Esse valor é menor em comparação ao obtido pelos algoritmos NSGAII (0.54) e IBEA (0.87), o que não indica necessariamente as melhores soluções. Para a LPS EShop o algoritmo não apresentou o maior número de soluções encontradas, 18 contra 20 do algoritmo NSGAII, entretanto, apresentou novamente o menor valor de ER. No caso das LPS JAMES e WeatherStation o algoritmo IBEA apresentou os menores valores de ER mesmo apresentando menor diversidade de soluções encontradas. A Figura 7.1 apresenta os conjuntos PF_{true} encontradas para as LPS CAS, JAMES, WeatherStation e EShop, respectivamente no experimento 2M.

Analisando a Figura 7.1 observa-se que para a LPS JAMES a fronteira PF_{true} ob-

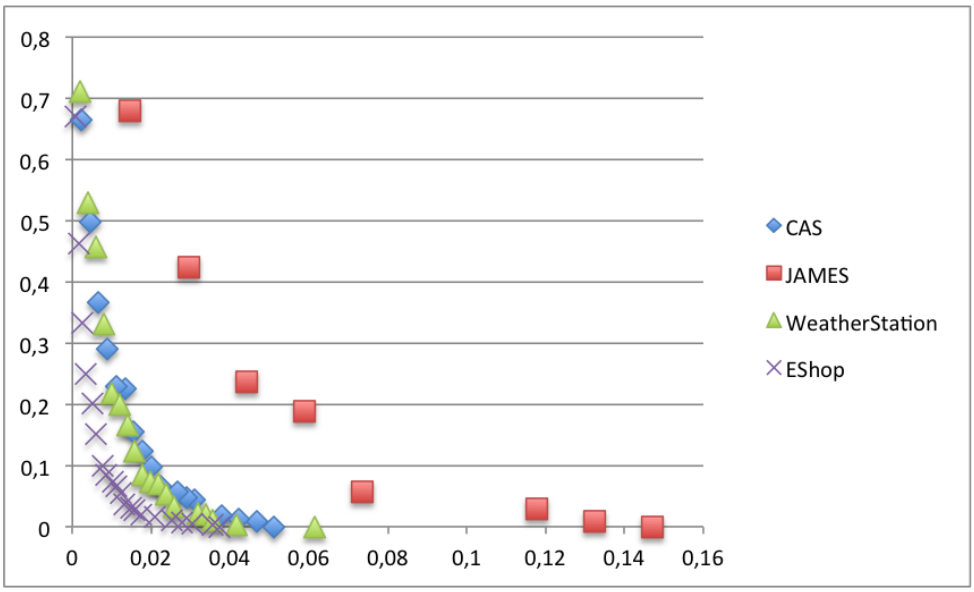

Figura 7.1: PF_{true} para 2 Objetivos

tida apresenta poucas soluções. Isso implica diretamente na sua distribuição no espaço de busca. Em comparação com as fronteiras das LPS CAS, WeatherStation e EShop visualiza-se a diferença entre o desenho apresentado pelas soluções encontradas. O pouco número de soluções da LPS JAMES faz com que sua fronteira não fique bem delineada como as demais. Um provável motivo para o menor número de soluções encontradas é que a LPS em questão possui o menor número de produtos das quatro utilizadas. Dessa maneira, o espaço de busca é muito menor se comparado, por exemplo, com as demais LPS (ver Tabela 7.2).

Para disponibilizar outro meio de análise quantitativa em relação aos resultados obtidos pelos experimentos, a Tabela 7.5 apresenta a média dos valores de *hypervolume* que o experimento 2M obteve em suas respectivas execuções.

Tabela 7.5: Médias de *hypervolume* experimento 2M

LPS	***Hypervolume***		
	NSGAII	**SPEA2**	**IBEA**
CAS	= 0,9661 (0,0176)	**= 0,9736 (0,0129)**	0,9641 (0,0178)
JAMES	**0,9414 (0,0061)**	0,9334 (0,0071)	0,9322 (0,0059)
WeatherStation	= 0,9753 (0,0102)	= 0,9718 (0,0135)	**= 0,9799 (0,0045)**
EShop	**= 0,9888 (0,0049)**	= 0,9843 (0,0098)	0,9821 (0,0083)

Na Tabela 7.5 o sinal "=" representa médias de *hypervolume* que não apresentaram diferença estatística, segundo o teste de Kruskal-Wallis com 5% de significância, quando

comparadas com outras médias do mesmo experimento e da mesma LPS. O teste foi realizado não apenas com o valor das médias mas com os valores das trinta execuções que as geraram. Em negrito destacam-se os melhores valores de *hypervolume* seguidos, em parênteses, pelos desvios padrão.

A primeira coluna da tabela apresenta as LPS utilizadas. As colunas dois, três e quatro apresentam os valores de média obtidos através dos algoritmos NSGAII, SPEA2 e IBEA, respectivamente, para o experimento 2M.

Observa-se que somente no caso da LPS JAMES a melhor média de *hypervolume* não obteve igualdade estatística. Nas demais LPS o melhor valor de média apresenta sempre uma igualdade estatística como no caso das LPS CAS e EShop, ou duas igualdades como no caso da LPS WeatherStation. Considerando-se essas igualdades observa-se que o algoritmo NSGAII apresenta os melhores valores em todos os quatro casos estudados. O algoritmo SPEA2 apresenta os melhores resultados em três dos quatro casos. Nesse experimento o algoritmo IBEA não obteve bons valores de *hypervolume*, apresentando somente um melhor valor de média com duas igualdades estatísticas.

A Tabela 7.6 apresenta os tempos de execução, em segundos, para cada algoritmo em relação a cada LPS no experimento 2M. Em negrito destacam-se os menores tempos de execução para cada LPS.

Tabela 7.6: Tempos de execução do experimento 2M

LPS	**Tempo de execução**		
	NSGAII	**SPEA2**	**IBEA**
CAS	73	105	**60**
JAMES	**5**	39	12
WeatherStation	98	139	**85**
EShop	245	273	**189**

Sobre os tempos obtidos, para o experimento 2M o algoritmo IBEA obteve os melhores valores em três dos quatro casos. O único caso em que outro algoritmo foi mais rápido foi a LPS JAMES, em que o algoritmo NSGAII foi melhor.

Outro fator de compração foi considerar os menores valores de ED. A Tabela 7.7 apresenta os valores de ED obtidos para o experimento 2M. A primeira coluna mostra a LPS utilizada. A segunda coluna apresenta a melhor solução encontrada, considerando

o número de mutantes mortos, pelo algoritmo NSGA-II para cada LPS. De maneira análoga as colunas quatro e seis apresentam as melhores soluções para os algoritmos SPEA2 e IBEA. As colunas três, cinco e sete apresentam os valores de menor ED para os algoritmos NSGAII, SPEA2 e IBEA, respectivamente. Esses valores são calculados com base em uma solução ideal, nesse caso, um único produto que possua 100% de cobertura no critério avaliado. Cada célula da tabela apresenta os valores de ED e o valor transformado de *fitness* (Tamanho Conjunto; Cobertura de mutantes). Ou seja, o valor apresentado não é o mesmo utilizado para o cálculo, está apenas transformado para melhor visualização. Em negrito destacam-se os melhores valores.

Tabela 7.7: Valores de ED para o experimento 2M

LPS	**Algoritmos**					
	NSGAII		**SPEA2**		**IBEA**	
	Melhor	**-ED**	**Melhor**	**-ED**	**Melhor**	**-ED**
CAS	(44; 100)	0,044 (19; 98.678)	(47; 100)	**0,041 (17; 98.238)**	(23; 100)	0,048 (17; 96.916)
JAMES	(10; 100)	0,117 (7; 94.340)	(13; 100)	0,121 (8; 97.170)	(11; 100)	**0,092 (5; 94.340)**
WeatherStation	(31; 100)	**0,037 (18; 98.880)**	(35; 100)	0,039 (17; 98.039)	(34; 100)	0,038 (16; 97.759)
EShop	(43; 100)	0,028 (28; 98.477)	(51; 100)	0,026 (20; 97.970)	(44; 100)	**0,025 (24; 98.477)**

Observa-se que, em relação aos melhores resultados (menor ED), o algoritmo IBEA obteve melhor desempenho em duas LPS. Para as duas outras LPS os algoritmos SPEA2 e NSGAII obtiveram os melhores resultados. Comparando os melhores resultados considerando o ED e a cobertura de mutantes (colunas dois, quatro e seis) percebe-se que o número de produtos sobe muito para elevar um pequeno percentual de cobertura, isso ocorre em todos os casos apresentados. Esse fato pode ser melhor observado no algoritmo SPEA2 quando executado para a LPS CAS. Nesse caso, o número de produtos sobe de 17 para 47, um acréscimo de 30 produtos, para elevar apenas 1,762%. Considerando as melhores soluções com relação a cobertura de mutantes observa-se que o algoritmo NSGA-II obteve os melhores resultados, em relação aos demais algoritmos, em três das quatro LPS. O outro melhor resultado foi obtido pelo algoritmo IBEA para a LPS CAS.

Nesse contexto, a solução que será utilizada para o teste de LPS é selecionada pelo

testador. Ele pode optar por valorizar o critério de cobertura, e assim assegurar que 100% dos mutantes vivos (AM) sejam cobertos ou optar por valorizar a solução com melhor valor de ED, e obter um conjunto minimizado de produtos que não possui total cobertura, mas que representa o melhor compromisso entre os objetivos. Vale ressaltar que essa é uma decisão diretamente relacionada aos interesses e recursos disponíveis ao testador.

7.4.1.1 Exemplo de Solução - Experimento 2M

A Tabela 7.8 apresenta em detalhes soluções obtidas no experimento 2M. A primeira coluna apresenta o ID da solução, colunas 2 e 3 apresentam seus valores de *fitness* correspondentes e transformados para melhor visualização, coluna 4 apresenta os produtos pertencentes a solução. Esse exemplo foi obtido na execução do algoritmo NSGA-II para a LPS CAS com os seguintes parâmetros: Tamanho de população 50, número máximo de avaliações 10.000, tamanho da população auxiliar 0 (NSGA-II não utiliza), 0,1 para taxa de recombinação (correspondente a 10%) e 0,5 para taxa de mutação (correspondente a 50%). Esse conjunto de soluções possui tamanho 13 e varia o número de produtos de 1 até 44, variando a cobertura do critério de mutação entre 33,48% e 100%.

Tabela 7.8: Soluções NSGA-II para LPS CAS - Experimento 2M

ID	Valores de Fitness		Produtos
	(S; A)	(#número de produtos; %cobertura mutantes)	
1	(0,002; 0,665)	(1; 33,48)	449
2	(0,004; 0,529)	(2; 47,137)	109, 402
3	(0,007; 0,366)	(3; 63,436)	282, 351, 438
4	(0,009; 0,348)	(4; 65,198)	77, 124, 171, 433
5	(0,011; 0,229)	(5; 77,093)	39, 208, 287, 361, 421
6	(0,016; 0,154)	(7; 84,581)	87, 132, 190, 319, 325, 332, 376
7	(0,018; 0,141)	(8; 85,903)	16, 25, 202, 262, 285, 312, 326, 370
8	(0,024; 0,084)	(11; 91,630)	8, 29, 33, 47, 82, 139, 165, 236, 297, 385, 432
9	(0,026; 0.079)	(12; 92,070)	6, 37, 44, 60, 140, 216, 251, 325, 349, 411, 421, 442
10	(0,029; 0,048)	(13; 95,154)	11, 17, 36, 67, 130, 171, 200, 216, 300, 313, 359, 395, 412
11	(0,042; 0,013)	(19; 98,678)	15, 18, 34, 87, 107, 113, 134, 143, 154, 255, 257, 264, 279, 287, 312, 364, 388, 414, 416
12	(0,053; 0,004)	(24; 99,559)	13, 29, 57, 110, 132, 138, 146, 153, 154, 156, 157, 177, 187, 215, 230, 235, 238, 245, 269, 287, 338, 394, 418, 436
13	(0,098; 0,0)	(44; 100,0)	9, 15, 18, 26, 30, 34, 43, 68, 77, 83, 84, 90, 108, 121, 125, 126, 137, 144, 165, 198, 209, 212, 235, 243, 244, 246, 277, 289, 297, 299, 304, 310, 312, 329, 332, 341, 349, 351, 362, 383, 412, 415, 438, 444

Através da Tabela 7.9 são apresentados alguns dos produtos pertencentes a solução com o melhor compromisso entre os objetivos, no caso a solução 10 com 19 produtos que possui 98,678% de cobertura de mutação.

Tabela 7.9: Produtos inclusos na solução 10

Produto	**Características Inclusas**
15	TRAFIC-MESSAGE-CHANEL, WHEEL-CONTROL, NAVIGATION-SYSTEM, MAP-DATA-VIA-CD, MAP-DATA-VIA-USB, PLAYBACK, USB, CD, CONTROL, TITLE-CHANEL-SELECTION, FORWARD-BACKWARD, VOLUME, SWITCH
18	TRAFIC-MESSAGE-CHANEL, NAVIGATION-SYSTEM, PLAYBACK, CASSETE, CONTROL, TITLE-CHANEL-SELECTION, FORWARD-BACKWARD, VOLUME, SWITCH
34	TRAFIC-MESSAGE-CHANEL, NAVIGATION-SYSTEM, PLAYBACK, CD, CONTROL, TITLE-CHANEL-SELECTION, FORWARD-BACKWARD, VOLUME, SWITCH, MEDIA-FORMAT, AAC
87	TRAFIC-MESSAGE-CHANEL, WHEEL-CONTROL, NAVIGATION-SYSTEM, PLAYBACK, USB, DVD, CONTROL, TITLE-CHANEL-SELECTION, FORWARD-BACKWARD, VOLUME, SWITCH, MEDIA-FORMAT, MP3, AAC
107	TRAFIC-MESSAGE-CHANEL, WHEEL-CONTROL, PLAYBACK, CASSETE, CONTROL, TITLE-CHANEL-SELECTION, FORWARD-BACKWARD, VOLUME, SWITCH, MEDIA-FORMAT, WMA

7.4.2 Resultados Experimento 3MP

A Tabela 7.10 apresenta informações para as fronteiras de Pareto obtidas no experimento 3MP. A primeira coluna exibe a LPS utilizada enquanto a segunda apresenta a quantidade de soluções na fronteira real (PF_{true}).

Tabela 7.10: Fronteiras de Pareto - Resultados experimento 3MP

LPS	**PFtrue**	**PFknown**		
		NSGAII	**SPEA2**	**IBEA**
CAS	20	23/3 (0.87)	**21/13 (0.38)**	17/4 (0.76)
JAMES	13	12/2 (0.83)	**13/7 (0.42)**	13/5 (0.62)
WeatherStation	28	23/5 (0.78)	**24/18 (0.25)**	26/6 (0.77)
EShop	29	37/15 (0.59)	**20/10 (0.50)**	18/4 (0.78)

Para a tabela em questão observa-se uma unanimidade nos menores valores de ER. Em todos os casos o algoritmo SPEA2 obteve melhores resultados. Destaca-se o desempenho do algoritmo na LPS Weather Station onde de um total de 24 soluções encontradas 18 delas pertencem a PF_{true}, o que corresponde a um ER de apenas 0.25. Apesar disso,

em nenhum desses casos, observou-se uma maior diversidade de soluções encontradas por esse algoritmo. Para dois dos quatro casos (CAS e EShop) o NSGAII obteve um maior número de soluções, em outro caso (JAMES) o SPEA2 empatou com o IBEA e no último caso (WeatherStation) o IBEA obteve um maior número de soluções.

A Tabela 7.11 apresenta a média dos valores de *hypervolume* que cada experimento obteve em suas respectivas execuções.

Tabela 7.11: Médias de *hypervolume* experimento 3MP

LPS	***Hypervolume***		
	NSGAII	**SPEA2**	**IBEA**
CAS	0,9672 (0,0125)	**0,9819 (0,0022)**	0,965 (0,013)
JAMES	**= 0,9213 (0,0093)**	= 0,9211 (0,0139)	0,9166 (0,0122)
WeatherStation	0,977 (0,0059)	**0,9819 (0,003)**	0,9768 (0,0063)
EShop	**0,9885 (0,0025)**	0,9842 (0,0078)	0,976 (0,0096)

No experimento 3MP apenas o caso da LPS JAMES apresentou igualdade estatística entre os valores dos algoritmos NSGAII e SPEA2. Considerando a igualdade estatística, o algoritmo NSGAII apresentou melhores valores de *hypervolume* para as LPS JAMES e EShop. Já o algoritmo SPEA2 apresentou os melhores valores de *hypervolume* em três LPS: CAS, JAMES e WeatherStation.

A Tabela 7.12 apresenta os tempos de execução, em segundos, para cada algoritmo em relação a cada LPS no experimento 3MP. Em negrito destacam-se os menores tempos de execução para cada LPS.

Tabela 7.12: Tempos de execução do experimento 3MP

LPS	**Tempo de Execução**		
	NSGAII	**SPEA2**	**IBEA**
CAS	135	219	**102**
JAMES	58	54	**17**
WeatherStation	429	246	**122**
EShop	**325**	348	399

Para o experimento 3MP o algoritmo IBEA obteve os melhores tempos para as LPS CAS, JAMES e Weather Station. No caso da LPS EShop o melhor tempo obtido foi através do algoritmo NSGAII.

Com relação ao ED a Tabela 7.13 apresenta os menores valores de ED obtidos para

o experimento 3MP. Esses valores são obtidos através da comparação com uma solução ideal, nesse caso, um único produto que mate 100% dos mutantes e cubra 100% dos pares. Em negrito destacam-se os melhores valores.

Tabela 7.13: Valores de ED para o experimento 3MP

LPS	Algoritmo					
	NSGA-II		SPEA2		IBEA	
	Melhor	-ED	Melhor	-ED	Melhor	-ED
CAS	(45;100;100)	0,049 (21;98,678;98,907)	(41;100;100)	**0,039 (15;98,238;98,907)**	(44;100;100)	0,044 (14;96,916;99,453)
JAMES	(13;100;100)	0,117 (7;94,340;100)	(11;100;100)	**0,106 (7;97,170;100)**	(13;100;100)	0,118 (8;99,057;98,667)
WeatherStation	(28;100;100)	0,042 (21;99,160;100)	(23;100;100)	**0,038 (15;98,039;98,462)**	(25;100;100)	0,043 (20;98,599;98,974)
EShop	(40;100;100)	**0,022 (24;99,492;99,505)**	(31;100;100)	0,026 (31;100;100)	(45;100;100)	0,031 (29;98,477;99,010)

Observando-se a Tabela 7.13 observa-se que, das quatro LPS utilizadas o algoritmo SPEA2 obteve os melhores resultados (menor ED) em três delas: CAS, JAMES e WeatherStation. Para a LPS EShop o algoritmo NSGAII apresentou o melhor valor de ED. O algoritmo IBEA, nesse experimento, não obteve nenhuma melhor solução. Comparando os melhores resultados considerando o ED e a cobertura de mutantes (colunas dois, quatro e seis) percebe-se o número de produtos necessários para aumentar um pequeno percentual de cobertura de mutantes. Isso pode ser visto através do algoritmo IBEA quando executado para LPS CAS, onde foi necessário o acréscimo de 30 produtos para o incremento de 3,084% na cobertura de mutantes e de apenas 0,547% na cobertura de pares. Considerando-se as melhores soluções com relação as coberturas de mutantes e pares, observa-se que o algoritmo SPEA2 foi melhor em todos os casos, com destaque para a LPS EShop onde a solução de melhor cobertura foi também a melhor solução com relação ao ED.

Assim como no caso do experimento 2M, nesse caso o testador precisará optar, de acordo com seus interesses e recursos disponíveis, pela solução que julgar melhor. A única diferença, nesse caso, é que existe um objetivo a mais, a cobertura de pares.

7.4.2.1 Exemplo de Solução - Experimento 3MP

A Tabela 7.14 apresenta em detalhes as soluções obtidas no experimento 3MP. A primeira coluna apresenta o ID da solução, colunas 2 e 3 apresentam os valores de *fitness* correspondentes e transformados para melhor visualização, coluna 4 apresenta os produtos

pertencentes a solução. Esse exemplo foi obtido na execução do algoritmo NSGA-II para a LPS CAS com os seguintes parâmetros: Tamanho de população 50, número máximo de avaliações 10.000, tamanho da população auxiliar 0 (NSGA-II não utiliza), 0,1 para taxa de recombinação (correspondente a 10%) e 0,9 para taxa de mutação (correspondente a 90%). Esse conjunto de soluções possui tamanho 23 e varia o número de produtos de 1 até 45, variando a cobertura do critério de mutação entre 31,278% e 100% e a cobertura do critério *pairwise* entre 57,377% e 100%.

Tabela 7.14: Soluções NSGA-II para LPS CAS - Experimento 3MP

ID	Fitness Values		Produtos
	(S; A; P)	(#número de produtos; %cobertura mutantes; %cobertura pares)	
1	(0,002; 0,687; 0,426)	(1; 31,278; 57,377)	425
2	(0,004; 0,533; 0,279)	(2; 46,696; 72,131)	328, 387
3	(0,007; 0,396; 0,246)	(3; 60,352; 75,410)	60, 84, 352
4	(0,007; 0,427; 0,142)	(3; 57,269; 85,792)	349, 371, 413
5	(0,009; 0,317; 0,191)	(4; 68,282; 80,874)	13, 81, 289, 442
6	(0,011; 0,189; 0,027)	(5; 81,057; 97,268)	53, 226, 247, 438, 449
7	(0,018; 0,141; 0,082)	(8; 85,903; 91,803)	3, 18, 33, 86, 118, 133, 190, 274
8	(0,018; 0,145; 0,071)	(8; 85,463; 92,896)	11, 65, 70, 129, 309, 338, 373, 442
9	(0,02; 0,110; 0,049)	(9; 88,987; 95,082)	46, 88, 104, 174, 181, 256, 281, 301, 323
10	(0,02; 0,145; 0,038)	(9; 85,463; 96,721)	5, 35, 67, 151, 166, 406, 410, 411, 446
11	(0,024; 0,079; 0,005)	(11; 92,070; 99,454)	89, 105, 107, 149, 232, 328, 378, 399, 415, 429, 433
12	(0,029; 0,062; 0,027)	(13; 93,833; 97,268)	14, 89, 229, 232, 239, 310, 324, 336, 343, 353, 383, 395, 421
13	(0,031; 0,044; 0,005)	(14; 95,595; 99,454)	22, 40, 86, 104, 107, 109, 180, 232, 283, 329, 336, 337, 370, 439
14	(0,036; 0,062; 0)	(16; 93,833; 100)	0, 1, 57, 59, 81, 83, 165, 197, 205, 206, 283, 350, 375, 401, 416, 439
15	(0,038; 0,040; 0,011)	(17; 96,035; 98,907)	18, 28, 49, 72, 105, 108, 132, 148, 183, 271, 272, 306, 378, 411, 431, 436, 444
16	(0,04; 0,035; 0,011)	(18; 96,476; 98,907)	23, 25, 28, 66, 72, 106, 141, 208, 252, 258, 260, 314, 344, 360, 361, 367, 428, 445
17	(0,047; 0,013; 0,011)	(21; 98,678; 98,907)	12, 16, 29, 61, 101, 134, 149, 180, 201, 204, 252, 262, 268, 277, 278, 298, 329, 361, 368, 420, 445
18	(0,049; 0,031; 0)	(22; 96,916; 100)	10, 32, 59, 95, 97, 98, 138, 142, 170, 199, 211, 247, 253, 254, 343, 362, 364, 374, 419, 433, 437, 447
19	(0,049; 0,026; 0,005)	(22; 97,357; 99,454)	7, 19, 22, 55, 70, 88, 95, 110, 126, 132, 161, 270, 274, 275, 296, 297, 344, 346, 355, 407, 409, 447
20	(0,058; 0,013; 0)	(26; 98,678; 100)	19, 29, 41, 43, 44, 59, 62, 73, 75, 101, 129, 130, 138, 170, 194, 270, 304, 318, 321, 326, 332, 356, 366, 375, 395, 425
21	(0,073; 0,009; 0)	(33; 99,119; 100)	7, 12, 19, 61, 62, 70, 87, 88, 103, 104, 132, 136, 148, 150, 164, 174, 200, 208, 216, 222, 223, 243, 244, 255, 266, 296, 317, 320, 327, 337, 346, 408, 409
22	(0,084; 0,004; 0)	(38; 99,560; 100)	14, 18, 24, 37, 47, 64, 67, 74, 79, 106, 130, 138, 148, 168, 198, 203, 204, 211, 244, 248, 258, 265, 283, 284, 290, 300, 302, 308, 324, 345, 349, 355, 361, 404, 412, 423, 448, 449
23	(0,1; 0; 0)	(45; 100; 100)	3, 11, 13, 16, 21, 29, 40, 65, 79, 102, 111, 115, 117, 143, 144, 156, 159, 188, 194, 199, 200, 209, 215, 218, 225, 229, 233, 245, 270, 274, 278, 279, 287, 309, 319, 324, 325, 326, 369, 372, 375, 377, 391, 438, 448

Através da Tabela 7.15 são apresentados alguns dos produtos pertencentes a solução com o melhor compromisso entre os objetivos, no caso a solução 17 com 21 produtos que possui 98,678% de cobertura de mutação e 98,907% de cobertura *pairwise*.

7.4.3 Resultados - Comparação Experimentos

A Tabela 7.16 apresenta informações sobre as fronteiras de Pareto obtidas nos experimentos. A primeira coluna exibe a LPS utilizada enquanto a segunda e terceira colunas

Tabela 7.15: Produtos incluídos na solução 17

Produto	**Características Inclusas**
12	TRAFIC-MESSAGE-CHANEL , NAVIGATION-SYSTEM , MAP-DATA-VIA-CD , PLAYBACK , USB , CD , CONTROL , TITLE-CHANEL-SELECTION , FORWARD-BACKWARD , VOLUME , SWITCH
101	TRAFIC-MESSAGE-CHANEL , WHEEL-CONTROL , NAVIGATION-SYSTEM , MAP-DATA-VIA-USB , PLAY-BACK , USB , CD , CONTROL , TITLE-CHANEL-SELECTION , FORWARD-BACKWARD , VOLUME , SWITCH , MEDIA-FORMAT , WMA
204	TRAFIC-MESSAGE-CHANEL , PLAYBACK , USB , DVD , CONTROL , TITLE-CHANEL-SELECTION , FORWARD-BACKWARD , VOLUME , SWITCH , MEDIA-FORMAT , MP3 , AAC , WMA , Product 205 : CAS , TRAFIC-MESSAGE-CHANEL , WHEEL-CONTROL , PLAYBACK , USB , DVD , CONTROL , TITLE-CHANEL-SELECTION , FORWARD-BACKWARD , VO-LUME , SWITCH , MEDIA-FORMAT , MP3 , AAC , WMA
329	TRAFIC-MESSAGE-CHANEL , WHEEL-CONTROL , NAVIGATION-SYSTEM , MAP-DATA-VIA-USB , PLAYBACK , USB , DVD , CONTROL , TITLE-CHANEL-SELECTION , FORWARD-BACKWARD , VOLUME , SWITCH , MEDIA-FORMAT , MP3 , AAC , WAV
445	TRAFIC-MESSAGE-CHANEL , WHEEL-CONTROL , PLAY-BACK , USB , DVD , CONTROL , TITLE-CHANEL-SELECTION , FORWARD-BACKWARD , VOLUME , SWITCH , MEDIA-FORMAT , MP3 , AAC , WMA , WAV

apresentam a quantidade de soluções em suas fronteiras reais (PF_{true}) para os experimentos 2M e 3MP, respectivamente. As colunas quatro, cinco e seis apresentam os resultados com relação ao ER para os algoritmos NSGA-II, SPEA2 e IBEA no experimento 2M. De maneira análoga, as colunas sete, oito e nove apresentam os mesmos resultados em relação ao experimento 3MP.

Tabela 7.16: Fronteiras de Pareto - Resultados

LPS	PF_{true}		PF_{known}					
			2M			**3MP**		
	2M	3MP	**NSGAII**	**SPEA2**	**IBEA**	**NSGAII**	**SPEA2**	**IBEA**
CAS	17	20	13/6 (0.54)	**16/9 (0.44)**	15/2 (0.87)	23/3 (0.87)	**21/13 (0.38)**	17/4 (0.76)
JAMES	8	13	10/4 (0.60)	10/3 (0.70)	**8/4 (0.50)**	12/2 (0.83)	**13/7 (0.42)**	13/5 (0.62)
WeatherStation	18	28	20/8 (0.60)	17/6 (0.65)	**19/8 (0.58)**	23/5 (0.78)	**24/18 (0.25)**	26/6 (0.77)
EShop	21	29	20/7 (0.65)	**18/9 (0.50)**	15/6 (0.60)	37/15 (0.59)	**20/10 (0.50)**	18/4 (0.78)

Através da Tabela 7.16 pode-se observar o aumento, em todas as LPS, do número de soluções não-dominadas do experimento 2M para 3MP. Isso demonstra que, para as LPS

analisadas, como era esperado, o acréscimo de um objetivo contribuiu diretamente para o aumento do conjunto PF_{true}.

Para disponibilizar outro meio de análise quantitativa em relação aos resultados obtidos pelos experimentos, as Tabelas 7.5 e 7.11 apresentam as média dos valores de *hypervolume* que cada experimento obteve em suas respectivas execuções. O algoritmo IBEA não obteve bons resultados não apresentando melhor *hypervolume* para nenhum dos casos estudados. Observa-se que no caso da LPS CAS os valores de *hypervolume* aumentaram em todos os algoritmos do experimento 2M para o experimento 3MP. Isso, entretanto, não é regra geral. Nos casos das LPS JAMES e EShop houve uma pequena queda nas médias de valores quando comparados os experimentos. Para a LPS WeatherStation houve queda nas médias para os algoritmos NSGAII e IBEA e ligeiro acréscimo no algoritmo SPEA2.

A Tabela 7.17 apresenta os tempos de execução, em segundos, para cada algoritmo em relação a cada LPS nos experimentos 2M e 3MP. Em negrito destacam-se os menores tempos de execução para cada LPS.

Tabela 7.17: Tempo execução dos experimentos

LPS	**Tempo de execução**					
	2M			**3MP**		
	NSGAII	**SPEA2**	**IBEA**	**NSGAII**	**SPEA2**	**IBEA**
CAS	73	105	**60**	135	219	**102**
JAMES	**5**	39	12	58	54	**17**
WeatherStation	98	139	**85**	429	246	**122**
EShop	245	273	**189**	**325**	348	399

Observa-se através da tabela que o algoritmo IBEA obteve os menores tempos de execução em três das quatro LPS nos dois experimentos executados. As exceções são nos experimento 2M o algoritmo NSGAII na LPS JAMES e no experimento 3MP o mesmo algoritmo na LPS EShop. Apesar do menor tempo de execução, o algoritmo IBEA, para os experimentos estudados, não obteve bons resultados como visto na Tabela 7.16. Em comparação com os algoritmos NSGAII e SPEA2, o algoritmo obteve maiores valores de ER e menores médias de *hypervolume*.

Na Tabela 7.18 são comparados os valores de ED obtidos nos dois experimentos realizados. A primeira coluna apresenta o nome da LPS avaliada, as colunas dois, quatro e

seis apresentam os menores valores de ED para os algoritmos NSGAII, SPEA2 e IBEA, respectivamente, quando utilizados no experimento 2M. As colunas três, cinco e sete apresentam os maiores valores de ED. Analogamente, as colunas de oito a treze apresentam os valores de menor e maior ED para o experimento 3MP. Nessa tabela não são apresentados as soluções ideais nem os valores de *fitness* obtidos. Esses valores estão disponíveis nas Tabelas 7.7 e 7.13.

Tabela 7.18: Valores de menor ED

LPS	**Menores valores ED**					
	2M			**3MP**		
	NSGAII	**SPEA2**	**IBEA**	**NSGAII**	**SPEA2**	**IBEA**
CAS	0,044	**0,041**	0,048	0,049	**0,039**	0,044
JAMES	0,117	0,121	**0,092**	0,117	**0,106**	0,118
WeatherStation	**0,037**	0,039	0,038	0,042	**0,038**	0,043
EShop	0,028	0,026	**0,025**	**0,022**	0,026	0,031

Analisando-se a Tabela 7.18 observa-se que em dois casos, o acréscimo de um objetivo auxiliou na obtenção de melhores valores de ED. Esses casos são vistos nas LPS CAS onde o valor de menor ED foi de 0,039 para 0,037 e na LPS EShop onde o valor variou de 0,024 para 0,021. Em outros dois casos o acréscimo de um objetivo prejudicou o valor obtido para o menor ED. Esses casos são as LPS JAMES, onde o valor variou de 0,081 para 0,092, e a LPS WeatherStation onde o valor variou de 0,035 para 0,037. Portanto, não se pode afirmar que o acréscimo de um objetivo prejudica ou auxilia na obtenção de um melhor valor de ED. Pode-se, entretanto, afirmar que existem casos em que a adição de um objetivo pode resultar em benefício e casos em que acontece o contrário.

A Tabela 7.19 apresenta o número de produtos necessários para conseguir 100% de cobertura, em todos os critérios avaliados, para os experimentos 2M e 3MP.

Tabela 7.19: Número de produtos para 100% de cobertura

LPS	**Número de Produtos**					
	2M			**3MP**		
	NSGAII	**SPEA2**	**IBEA**	**NSGAII**	**SPEA2**	**IBEA**
CAS	44	47	**23**	45	**41**	44
JAMES	**10**	13	11	13	**11**	13
WeatherStation	**31**	35	34	28	**23**	25
EShop	**43**	51	44	40	**31**	45

Através dessa tabela nota-se que para o experimento 2M o algoritmo NSGA-II obteve três melhores soluções, nesse caso aquelas com menor número de produtos e 100% de cobertura. Para a LPS CAS o algoritmo IBEA obteve o melhor resultado. Nesse experimento SPEA2 não obteve nenhum melhor resultado. Para o experimento 3MP observa-se uma unanimidade nas melhores soluções por parte do algoritmo SPEA2. Comparado o tamanho dos conjuntos entre os experimentos percebe-se que na maioria dos casos o número de produtos diminuiu quando adicionado mais um objetivo aos experimentos. Quando comparadas somente as melhores soluções esse número é equilibrado, existem casos em que há diminuição do conjunto e casos onde há aumento.

7.5 Respondendo às Questões de Pesquisa

A seguir são apresentadas as respostas às questões de pesquisas propostas na Seção 7.2.

7.5.1 Questão 1

Diferentemente das abordagens com um único objetivo, na abordagem proposta os algoritmos produzem diferentes soluções ótimas. Esse aspecto é uma vantagem para o testador que pode priorizar um objetivo escolhendo soluções nos extremos da fronteira ou de acordo com os objetivos do teste e recursos disponíveis.

Soluções com melhores valores de ED representam os melhores *trade-offs* entre os objetivos. Considerando os melhores valores para o indicador ED foi possível selecionar os resultados com melhores *trade-off* entre objetivos para cada algoritmo executado (NSGAII, SPEA2 e IBEA) em cada LPS analisada (CAS, JAMES, WeatherStation e EShop). A seguir são apresentados os valores de *fitness* obtidos nessa avaliação.

Considerando-se primeiramente o experimento 2M os melhores resultados obtidos são: para CAS um conjunto de 17 produtos e uma cobertura de 98,238%; JAMES um conjunto de 5 produtos e 94,34% de cobertura; WeatherStation um conjunto de 18 produtos e 98,88% de cobertura; EShop 24 produtos e 98,477%. Com exceção da JAMES, todas as

outras LPS obtiveram cobertura superior a 98%, o que representa um bom percentual de cobertura.

Além do alto percentual de cobertura, destaca-se a redução no número de produtos que necessitam ser testados em relação ao conjunto total. Para a LPS CAS que possui no total 450 produtos, são necessários apenas 17 produtos para uma cobertura maior que 98%, o que representa uma redução de 96,22% no tamanho do conjunto produtos para teste. Para JAMES essa redução corresponde a 92,65%, apenas 5 produtos de 68. No caso da WeatherStation esse percentual corresponde a 96,43% reduzindo o número de produtos de 504 para 18. Finalmente, a LPS EShop obteve o maior percentual de redução 97,92%, reduzindo o conjunto de produtos de 1152 para 24.

Caso o testador deseje valorizar as melhores soluções com 100% de cobertura de mutantes a redução no tamanho do conjunto de produtos é menor, entretanto ainda expressiva. No caso da LPS CAS o número iria de 450 para 23, o que corresponde a 94,89% de redução. Para a LPS JAMES esse valor seria de 85,29%. No caso da LPS WeatherStation essa redução representa 93,85% e para a LPS Eshop o valor de redução seria de 96,27%.

No caso do experimento 3MP os melhores resultados obtidos são: para a LPS CAS um conjunto de 15 produtos, 98,238% de cobertura no critério mutantes e 98,907% de cobertura no critério *pairwise*; LPS JAMES 7 produtos, 97,17% de cobertura mutantes e 100% de cobertura *pairwise*; para WeatherStation 15 produtos, 98,039% de cobertura mutantes e 98,492% de cobertura *pairwise*; EShop um conjunto de 24 produtos, 99,942% de cobertura mutantes e 99,505% de cobertura *pairwise*.

De maneira análoga a análise do experimento 2M, no caso do experimento 3MP também observaram-se reduções significativas no tamanho dos conjuntos de produtos para teste. No caso da LPS CAS a redução foi maior que no experimento 2M, com o decréscimo de 450 produtos para 15, o que representa uma redução de 96,67%. Para a LPS JAMES esse percentual foi de 89,71%, o menor obtido. WeatherStation obteve uma redução de 97,02%. A LPS EShop manteve o percentual de redução do experimento 2M, 97,92%.

Também nesse experimento o testador pode priorizar os conjuntos com 100% de cober-

tura nos critérios análise de mutantes e cobertura *pairwise*. Assim como no experimento 2M, há um decréscimo no percentual de redução, entretanto esse valor permanece expressivo quando comparado ao conjunto total de produtos válidos. No caso da LPS CAS a redução é de 90,89% no número de produtos. Para JAMES esse valor corresponde a 83,82%. No caso da LPS WeatherStation a redução fica em 95,44% e para a LPS EShop o valor fica em 97,31%.

Através dos valores apresentados o testador pode ponderar as opções disponíveis e optar por aquela solução que melhor satisfaça suas necessidades. Isso inclui optar por priorizar uma cobertura total dos critérios avaliados ou priorizar os melhores valores de ED, que possuem os menores conjuntos mas não possuem cobertura total.

7.5.2 Questão 2

Com relação à questão de pesquisa 2, os seguintes pontos principais foram observados comparando os algoritmos.

No experimento 2M, NSGA-II apresentou grande diversidade de soluções e um maior número de soluções no conjunto PF_{known}, na maioria dos casos. Esse algoritmo também obteve os melhores valores de *hypervolume*. Isso significa que o algoritmo oferece para o testador um maior número de soluções com diferentes valores de *fitness*, o que proporciona uma escolha de acordo com suas preferências ou de acordo com as restrições da atividade de teste.

IBEA e SPEA2 apresentaram maior número de soluções no conjunto PF_{true}. O algoritmo IBEA parece ser uma melhor alternativa em relação ao SPEA2 pois obteve soluções com menores valores de ED em um tempo de execução mais baixo. Essas soluções representam os melhores *trade-offs* entre os algoritmos. Além disso, IBEA parece ser uma boa opção para a utilização em LPS de grande porte. Entretanto, se o testador estiver interessado apenas em valores associados ao *fitness* relacionado ao teste de mutação, NSGA-II parece ser a melhor escolha.

No caso do experimento 3MP, o algoritmo que apresentou grande diversidade e maior número de soluções presentes no conjunto PF_{known} foi o SPEA2. Considerando as igual-

dades estatísticas, esse algoritmo obteve três dos quatro melhores *hypervolumes*. Isso significa que o algoritmo oferece um maior número de soluções com diferentes valores de *fitness*, o que proporciona uma escolha, por parte do testador, de acordo com suas necessidades.

NSGA-II e IBEA não obtiveram bons resultados com relação ao número de soluções no conjunto PF_{known}. Com relação ao tempo de execução o algoritmo IBEA se sobressai aos demais, sendo o mais rápido em três das quatro LPSs. Com relação aos valores de ED novamente o SPEA2 se mostrou a melhor opção obtendo três dos melhores resultados. Assim, se o testador desejar melhores opções com relação ao tempo de execução o algoritmo IBEA é o mais indicado, já se optar por valorizar os melhores conjuntos de teste então SPEA2 é a melhor opção.

7.6 Ameaças à Validade

Nesta seção são apresentadas algumas ameaças que podem invalidar os resultados obtidos. Elas são descritas a seguir.

- As LPSs utilizadas são de pequeno porte. Entretanto, os resultados obtidos através delas oferecem um indicativo inicial de que a utilização de algoritmos evolutivos multiobjetivos é capaz de fornecer bons resultados para o teste de mutação em conjunto com outros critérios como *pairwise*;

- Apesar da incapacidade do FaMa em gerar todos os produtos válidos para uma LPS de grande porte, na implementação realizada é possível definir um número limite de produtos que se deseja gerar. Entretanto, optou-se por utilizar apenas LPS pequenas que gerem todos os produtos para avaliação a fim de obter-se resultados iniciais e posteriormente avaliar a escalabilidade da abordagem em LPS maiores;

- Os algoritmos de otimização são não determinísticos por isso foram adotados procedimentos e avaliações citados na literatura para avaliar estes algoritmos. Eles foram executados 30 vezes e 1 *tuning* de parâmetros foi realizado.

7.7 Considerações Finais

Este capítulo apresentou os resultados da avaliação experimental conduzida em quatro LPS diferentes com três diferentes algoritmos. Foram criados dois experimentos distintos: i) 2M e ii) 3MP. Esses experimentos tiveram seus parâmetros ajustados para as quatro LPS e para os três algoritmos utilizados e posteriormente foram executados 30 vezes com as melhores configurações encontradas. Os resultados foram comparados com base nos indicadores ER, *hypervolume*, e nos tempos de execução. Após as execuções realizou-se uma análise quantitativa dos resultados obtidos.

Observou-se que os algoritmos NSGA-II, SPEA2 e IBEA obtêm um número reduzido de conjuntos de casos de teste quando comparados aos conjuntos totais de produtos válidos gerados. Entretanto, entre essas soluções é possível distinguir as melhores de acordo com a necessidade do testador.

Com os resultados obtidos é possível responder as questões de pesquisa colocadas na Seção 7.2. Dessa maneira, foi possível verificar a performance dos algoritmos NSGAII, SPEA2 e IBEA na resolução do problema de geração de dados para o teste de LPS. No caso do experimento 2M, em geral a cobertura em relação a mutação foi superior a 98%. Considerando o número de produtos, a redução foi em geral superior a 96%. No caso do experimento 3MP, em geral a cobertura foi superior a 98% nos critérios mutantes e *pairwise*. Além disso houve uma redução superior a 96% na maioria dos dados para o teste de LPS.

Foi possível, ainda, através da utilização do indicador ED selecionar as soluções com melhor *trade-off* entre os objetivos. No caso do experimento 2M, o algoritmo IBEA parece ser uma melhor alternativa em relação ao SPEA2 pois obteve soluções com menores valores de ED em um tempo de execução mais baixo. Caso o testador opte por uma LPS de grande porte, o algoritmo IBEA aparenta ser a melhor opção. Caso esteja interessado em valores relacionados ao *fitness* associado ao número de mutantes, NSGA-II parece a melhor solução. Para o experimento 3MP, o algoritmo IBEA se sobressai aos demais, sendo o mais rápido em três das quatro LPSs. Se o testador optar por valorizar os melhores conjuntos de teste então SPEA2 parece ser a melhor opção.

CAPÍTULO 8

CONCLUSÕES

Este trabalho introduziu uma abordagem baseada em busca que possui como objetivo a geração de conjuntos de produtos, que são conjuntos de dados de teste, para satisfazer o teste de mutação de FMs. A ideia é matar o maior número possível de mutantes, com o menor número de dados de teste, e ainda se desejado satisfazer o critério de teste *pairwise*. Para alcançar este objetivo a abordagem inclui uma representação para o problema que permite manipular uma população de conjuntos de dados de teste e operadores de busca adequados a esta representação, a serem utilizados pelos algoritmos evolutivos multiobjetivos.

Para implementar a abordagem proposta utilizou-se o framework jMetal e os algoritmos NSGA-II, SPEA2 e IBEA. Além disso, utilizou-se a ferramenta FMTS proposta por Ferreira [17] para a geração dos produtos e dos mutantes utilizados. Para a resolução do problema definiram-se três funções objetivo que visavam a minimizar o número de produtos nos conjuntos de teste, e maximizar a quantidade de mutantes mortos pela abordagem e a quantidade de pares cobertos.

Os resultados da avaliação mostram que boas soluções são geradas, representando os melhores *trade-offs* entre os objetivos: tamanho de conjuntos de produtos, cobertura de mutantes e cobertura *pairwise*. Essas soluções podem ser utilizadas pelo testador de acordo com suas necessidades. No experimento 2M a maioria dos conjuntos de produtos gerados possuía um tamanho reduzido 96% em relação ao conjunto total de produtos válidos com escores de mutação superiores a 98% de cobertura. O experimento 3MP registrou conjuntos de produtos, em geral, 96% menores com escores de mutação e *pairwise* acima dos 98%. Assim, o testador pode priorizar o objetivo que desejar selecionado os conjuntos de produtos. Por exemplo, se quiser priorizar a cobertura de mutantes pode selecionar o conjunto com maior número de produtos e maior cobertura. Se desejar

priorizar o tamanho do conjunto pode selecionar o menor de acordo com o percentual de redução. A grande vantagem dessa abordagem é oferecer diferentes alternativas para o teste, diferentemente de uma abordagem mono-objetivo.

As limitações encontradas durante a realização deste trabalho estão relacionadas principalmente ao *framework* FaMa. Verificou-se que durante alguns processos executados através desse *framework* são criados arquivos temporários que não são excluídos. Dependendo da quantidade de execuções que se está trabalhando esse problema pode acarretar no mal funcionamento do computador, com sintomas como lentidão e travamento. Além disso, o *framework* trabalha bem com LPS com pequeno número de características. Entretanto, ao tentar executar algum processo para uma LPS com um número maior de características (50, por exemplo), o FaMa não é capaz de executar o processo completamente, pois possui um *timeout* interno que interrompe a execução após determinado tempo. Por esse motivo, neste trabalho foram utilizados apenas LPS com número pequeno de características. Apesar dessas limitações, o FaMa foi utilizado pois já possuía integração com a ferramenta FMTS, utilizada neste trabalho.

As principais lições aprendidas na execução desta dissertação estão relacionadas ao método de desenvolvimento de trabalho e análise de resultados. Durante o desenvolvimento percebeu-se a importância de um bom embasamento teórico através dos trabalhos relacionados. Além disso, verificou-se a importância da busca por métodos e indicadores que permitam a avaliação quantitativa dos dados obtidos.

Este trabalho deixa algumas contribuições. A primeira delas corresponde a um incremento na ferramenta já proposta por Ferreira [17], FMTS. Além disso, pela primeira vez, foram utilizados diferentes critérios de teste de software em conjunto em um problema de otimização multiobjetivo para o teste de LPS. Esses critérios incluem a análise de mutantes e o teste *pairwise* citado em diferentes trabalhos. Outra contribuição é a comparação de diferentes algoritmos em diferentes experimentos. Como visto, na maioria dos casos em SBSE, são utilizados os algoritmos NSGAII, SPEA2 e IBEA. Entretanto, não há registro de trabalhos que utilizassem os três algoritmos em um comparativo para o problema em questão.

8.1 Trabalhos Futuros

A seguir são descritos alguns possíveis trabalhos futuros gerados a partir desse trabalho.

- **Adição de novos objetivos:** Um possível trabalho futuro é a avaliação do problema em questão com a adição de mais objetivos. Um exemplo de objetivo pode estar associado a um conjunto de preferências do usuário, o que traria mais interação para o problema, visto que o usuário indicaria quais características seriam mais interessantes do seu ponto de vista. Dessa maneira, o problema deveria levar em consideração a maximização da satisfação do usuário bem como o atendimento aos demais objetivos já estudados. Outros objetivos podem incluir restrições da atividade de teste tal como custo associado às características;

- **Utilização de novas ferramentas:** A busca por diferentes frameworks para geração e validação de produtos de um FM pode agilizar a resolução do problema em questão. Isto porque através de um novo framework talvez seja possível contornar problemas encontrados no decorrer desse trabalho como a geração de arquivos temporários e o *timeout* estabelecido pelo framework FaMa na geração dos produtos;

- **Utilização de outras LPS:** A utilização de outras LPS maiores e industriais pode ser objeto de estudo. Para LPS muito grandes o problema da explosão do número de produtos permaneça. Por isso a importância do limite no número de produtos implementado, que deverá ser avaliado em experimentos futuros. Através da análise desses casos pode-se obter a confirmação dos resultados já encontrados ou ainda encontrar novos campos de pesquisa como a geração de produtos para LPS com grande número de características e avaliação de escalabilidade;

- **Utilização de outros algoritmos de busca:** A abordagem inclui operadores evolutivos tais como de recombinação e mutação, e é independente de algoritmo. Por isso, outros algoritmos podem ser explorados e avaliados tais como o PAES e o MOGA.

BIBLIOGRAFIA

[1] A. Abraham, L. Jain C, e R. Goldberg. *Evolutionary Multiobjective Optimization: Theoretical Advances and Applications (Advanced Information and Knowledge Processing)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[2] A. Arcuri e G. Fraser. On parameter tuning in search based software engineering. *Proceedings of the Third International Conference on Search Based Software Engineering*, SSBSE'11, páginas 33–47, Berlin, Heidelberg, 2011. Springer-Verlag.

[3] W. K. G. Assunção. Uma abordagem para integração e teste de módulos baseada em agrupamento e algoritmos de otimização multiobjetivos. Dissertação de Mestrado, Universidade Federal do Paraná, 4 de 2012.

[4] D. Benavides, S. Trujillo, e P. Trinidad. On the modularization of feature models. *In First European Workshop on Model Transformation*, 2005.

[5] L. N. Castro. *Computação Natural: Uma Jornada Ilustrada*. Livraria da Física, 2010.

[6] C. A. Coello. An updated survey of GA-based multiobjective optimization techniques. ACM *Computing Surveys*, 32(2):109–143, junho de 2000.

[7] C.A Coello, G. B. Lamont, e D. A. Van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[8] C.A. Coello, G. T. Pulido, e E. M. Montes. Current and future research trends in evolutionary multiobjective optimization. *Information Processing with Evolutionary Algorithms*, Advanced Information and Knowledge Processing, páginas 213–231. Springer London, 2005.

[9] D. M. Cohen, S. R. Dalal, M. L. Fredman, e G. C. Patton. The AETG system: An approach to testing based on combinatorial design. IEEE *Transactions on Software Engineering*, 23(7):437–444, julho de 1997.

[10] M. B. Cohen, M. B. Dwyer, e J. Shi. Coverage and adequacy in software product line testing. *Proceedings of the* ISSTA *2006 Workshop on Role of Software Architecture for Testing and Analysis*, ROSATEA '06, páginas 53–63, New York, NY, USA, 2006. ACM.

[11] J. Cruz, P.S. Neto, R. Britto, R. Rabelo, W. Ayala, T. Soares, e M. Mota. Toward a hybrid approach to generate software product line portfolios. *Evolutionary Computation (CEC), 2013 IEEE Congress on*, páginas 2229–2236, June de 2013.

[12] M. E. Delamaro, J. C. Maldonado, e M. Jino. *Introdução ao Teste de Software*. Elsevier, Rio de Janeiro, 2007.

[13] J. J. Durillo e A. J. Nebro. jmetal: A java framework for multi-objective optimization. *Advances in Engineering Software*, 42:760–771, 2011.

[14] M. Laumanns E. Zitzler e L. Thiele. SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. *Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems*, páginas 95–100, Athens, Greece, 2001. International Center for Numerical Methods in Engineering.

[15] F. Ensan, E. Bagheri, e D. Gašević. Evolutionary search-based test generation for software product line feature models. *Proceedings of the 24th International Conference on Advanced Information Systems Engineering*, CAiSE'12, páginas 613–628, Berlin, Heidelberg, 2012. Springer-Verlag.

[16] *FaMa FW*. http://www.isa.us.es/fama/Documentation, 2014.

[17] J. M. Ferreira. Teste de linha de produto de software baseado em mutação do diagrama de características. Dissertação de Mestrado, Universidade Federal do Paraná, 2012. Dissertação de Mestrado.

[18] R.A. Matnei Filho e S.R. Vergilio. Configuração baseada em busca de linha de produto de software: Resultados de um mapeamento sistemático. V Workshop de Engenharia de Software Baseada em Busca (WESB), 2014.

[19] J. Guo, J. White, G. Wang, J. Li, e Y. Wang. A genetic algorithm for optimized feature selection with resource constraints in software product lines. *Journal of Systems and Software*, 84(12):2208 – 2221, 2011.

[20] M. Harman, Y. Jia, J. Krinke, W. B. Langdon, J. Petke, e Y. Zhang. Search based software engineering for software product line engineering: A survey and directions for future work. *Proceedings of the 18th International Software Product Line Conference - Volume 1*, SPLC '14, páginas 5–18, New York, NY, USA, 2014. ACM.

[21] M. Harman, S. A. Mansouri, e Y. Zhang. Search-based software engineering: Trends, techniques and applications. *ACM Computing Surveys*, 45(1):11:1–11:61, dezembro de 2012.

[22] C. Henard, M. Papadakis, G. Perrouin, J. Klein, P. Heymans, e Y. Le Traon. Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test suites for large software product lines. *Computing Research Repository - CoRR*, abs/1211.5451, 2012.

[23] C. Henard, M. Papadakis, G. Perrouin, J. Klein, e Y. le Traon. Assessing software product line testing via model-based mutation: An application to similarity testing. *Software Testing, Verification and Validation Workshops (ICSTW), 2013 IEEE Sixth International Conference on*, páginas 188–197, March de 2013.

[24] C. Henard, M. Papadakis, G. Perrouin, J. Klein, e Y. Le Traon. Multi-objective test generation for software product lines. *Proceedings of the 17th International Software Product Line Conference*, SPLC '13, páginas 62–71, New York, NY, USA, 2013. ACM.

[25] C. Henard, M. Papadakis, e Y. Le Traon. Mutation-based generation of software product line test configurations. Claire Le Goues e Shin Yoo, editors, *Search-Based*

Software Engineering, volume 8636 of *Lecture Notes in Computer Science*, páginas 92–106. Springer International Publishing, 2014.

[26] IEEE standard glossary of software engineering terminology. IEEE *Std 610.12-1990*, páginas 1–84, 1990.

[27] T. Yano e E. Martins I.R.S Lima. Uso de análise de mutantes para avaliação de uma abordagem de testes baseados em modelos: um estudo exploratório. 2012. [Online; acessado em 18/03/2015, disponível em : `http://icomp.ufam.edu.br/sast2012/papers/104713.pdfm`].

[28] S. Agarwal K. Deb, A. Pratap e T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, Abr de 2002.

[29] R. Karimpour e G. Ruhe. Bi-criteria genetic search for adding new features into an existing product line. *Combining Modelling and Search-Based Software Engineering* (CMSBSE)*, 2013 1st International Workshop on*, páginas 34–38, 2013.

[30] A. Konak, D. W. Coit, e A. E. Smith. Multi-objective optimization using genetic algorithms: A tutorial. *Reliability Engineering & System Safety*, 91(9):992–1007, setembro de 2006.

[31] W. H. Kruskal e W. A. Wallis. Use of ranks in one-criterion variance analysis. *Journal of the American Statistical Association*, 47(260):pp. 583–621, 1952.

[32] R. Kuhn, R. Kacker, Y. Lei, e J. Hunter. Combinatorial software testing. Number 8, páginas 94–96.

[33] B. P. Lamancha e M. P. Usaola. Testing product generation in software product lines using pairwise for features coverage. *Proceedings of the 22Nd IFIP WG 6.1 International Conference on Testing Software and Systems*, ICTSS'10, páginas 111–125, Berlin, Heidelberg, 2010. Springer-Verlag.

[34] R. E. Lopez-Herrejon, J. F. Chicano, J. Ferrer, A. Egyed, e E. Alba. Multi-objective optimal test suite computation for software product line pairwise testing. *ICSM*, páginas 404–407. IEEE, 2013.

[35] M. C. A. García e T. S. Mendonza. Implementation of an evolutionary algorithm in planning investment in a power distribution system. *Revista Ingeniería e Investigación*, 31(2):118–124, 2011.

[36] J. Muller. Value-based portfolio optimization for software product lines. *Proceedings of the 2011 15th International Software Product Line Conference*, SPLC '11, páginas 15–24, Washington, DC, USA, 2011. IEEE Computer Society.

[37] G. J. Myers e C. Sandler. *The Art of Software Testing*. John Wiley & Sons, 2004.

[38] C. Nie e H. Leung. A survey of combinatorial testing. ACM *Computing Surveys*, 43(2):11:1–11:29, fevereiro de 2011.

[39] R. Olaechea, D. Rayside, J. Guo, e K. Czarnecki. Comparison of exact and approximate multi-objective optimization for software product lines. *Proceedings of the 18th International Software Product Line Conference - Volume 1*, SPLC '14, páginas 92–101, New York, NY, USA, 2014. ACM.

[40] S. Oster, M. Zink, M. Lochau, e M. Grechanik. Pairwise feature-interaction testing for SPLs: Potentials and limitations. *Proceedings of the 15th International Software Product Line Conference, Volume 2*, SPLC '11, páginas 6:1–6:8, New York, NY, USA, 2011. ACM.

[41] V. Pareto. Manuel d'economie politique. Ams Press, Paris, 1927.

[42] J. A. Pereira, E. Figueiredo, e T. Noronha. Modelo computacional para apoiar a configuração de produtos em linha de produtos de software. *Anais do IV Workshop de Engenharia de Software Baseada em Busca*, WESB'13, páginas 80–89, 2013.

[43] K. Pohl, G. Böckle, e F. J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[44] M. Polo e B. Pérez. A framework and a web implementation for combinatorial testing. 2010. http://161.67.140.42/CombTestWeb/.

[45] R. S. Pressman. Rio de Janeiro, 5 edition.

[46] A. R. C. Rocha, J. C. Maldonado, K. C. Maldonado, e *et. al. Qualidade de Software - Teoria e Prática*. Prentice Hall, 2001.

[47] D. Sokenou S. Weißleder e H. Schlingloff. Reusing state machines for automatic test generation in product lines. Thomas Bauer, Hajo Eichler, Axel Rennoch, editor, *MoTiP ?08: Model-Based Testing in Practice*. Fraunhofer IRB Verlag, 2008.

[48] L.E. Sanchez, S. Moisan, e J. P. Rigault. Metrics on feature models to optimize configuration adaptation at run time. *Combining Modelling and Search-Based Software Engineering* (CMSBSE)*, 2013 1st International Workshop on*, páginas 39–44, 2013.

[49] A. S. Sayyad, J. Ingram, T. Menzies, e H. Ammar. Scalable product line configuration: A straw to break the camel's back. *Proc. ASE*, páginas 465–474. IEEE, 2013.

[50] A. S. Sayyad, T. Menzies, e H. Ammar. On the value of user preferences in search-based software engineering: A case study in software product lines. *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, páginas 492–501, Piscataway, NJ, USA, 2013. IEEE Press.

[51] A.S. Sayyad, J. Ingram, T. Menzies, e H. Ammar. Optimum feature selection in software product lines: Let your model and values guide your search. *Combining Modelling and Search-Based Software Engineering* (CMSBSE)*, 2013 1st International Workshop on*, páginas 22–27, 2013.

[52] S. Segura, R.M. Hierons, D. Benavides, e A. Ruiz-Cortes. Automated test data generation on the analyses of feature models: A metamorphic testing approach. *Software Testing, Verification and Validation (ICST), 2010 Third International Conference on*, páginas 35–44, April de 2010.

[53] SEI. A Framework for Software Product Line Practice, Version 4.2. SEI - Software Engineering Institute / Carnegie Mellon, 2007. [Online; acessado em 26/12/2013, disponível em : `http://www.sei.cmu.edu/productlines/frame_report/PL.essential.act.htm`].

[54] K. Tai e Y. Lei. A test generation strategy for pairwise testing. IEEE *Transactions on Software Engineering*, 28(1):109–111, 2002.

[55] F. J. van der Linden, K. Schmid, e E. Rommes. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[56] D. A. Van Veldhuizen. Multiobjective evolutionary algorithms: Classifications, analyses, and new innovations. Relatório técnico, Evolutionary Computation, 1999.

[57] W3Schools. XML tutorial, 2015. http://www.w3schools.com/xml/default.asp.

[58] S. Wang, S. Ali, e A. Gotlieb. Minimizing test suites in software product lines using weight-based genetic algorithms. *Proceeding of the Fifteenth Annual Conference on Genetic and Evolutionary Computation Conference*, GECCO '13, páginas 1493–1500, New York, NY, USA, 2013. ACM.

[59] S. Wang, D. Buchmann, S. Ali, A. Gotlieb, D. Pradhan, e M. Liaaen. Multi-objective test prioritization in software product line testing: An industrial case study. *Proceedings of the 18th International Software Product Line Conference - Volume 1*, SPLC '14, páginas 32–41, New York, NY, USA, 2014. ACM.

[60] J. White, J. A. Galindo, T. Saxena, B. Dougherty, D. Benavides, e D. C. Schmidt. Evolving feature model configurations in software product. *Journal of Systems and Software*, 87(0):119 – 136, 2014.

[61] E. Zitzler. *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*. Tese de Doutorado, ETH Zurich, Switzerland, 1999.

[62] E. Zitzler e S. Künzli. Indicator-based selection in multiobjective search. *Parallel Problem Solving from Nature - PPSN VIII*, volume 3242 of *Lecture Notes in Computer Science*, páginas 832–842. Springer Berlin Heidelberg, 2004.

[63] E. Zitzler, L. Thiele, M. Laumanns, C.M. Fonseca, e V.G. da Fonseca. Performance assessment of multiobjective optimizers: an analysis and review. *Evolutionary Computation, IEEE Transactions on*, 7(2):117–132, April de 2003.