Universidade Federal do Paraná

Jackson Antonio do Prado Lima

Uma abordagem baseada em hiper-heurística e otimização multi-objetivo para o teste de mutação de ordem superior

Curitiba PR
2017

Jackson Antonio do Prado Lima

Uma abordagem baseada em hiper-heurística e otimização multi-objetivo para o teste de mutação de ordem superior

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Informática no Programa de Pós-Graduação em Informática, setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Prof.^{a} Dr.^{a} Silvia Regina Vergilio.

Curitiba PR
2017

L732a Lima, Jackson Antonio do Prado
Uma abordagem baseada em hiper-heurística e otimização multi-objetivo para o teste de mutação de ordem superior / Jackson Antonio do Prado Lima. – Curitiba, 2017.
98 f ; il. color : 30 cm.

Dissertação - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática , 2017.

Orientador: Silvia Regina Vergilio .
Bibliografia: p. 90-98.

1. Software – Testes. 2. Algoritmos computacionais. 3. Programação heurística. I. Universidade Federal do Paraná. II.Vergilio, Silvia Regina. III. Título.

CDD: 005.14

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO PARANÁ
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
Setor CIÊNCIAS EXATAS
Programa de Pós-Graduação INFORMÁTICA

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em INFORMÁTICA da Universidade Federal do Paraná foram convocados para realizar a arguição da Dissertação de Mestrado de **JACKSON ANTONIO DO PRADO LIMA** intitulada: **Uma abordagem baseada em hiper-heurística e otimização multi-ojetivo para o teste de mutação de ordem superior**, após terem inquirido o aluno e realizado a avaliação do trabalho, são de parecer pela sua aprovação.

Curitiba, 23 de Fevereiro de 2017.

SILVIA REGINA VERGILIO
Presidente da Banca Examinadora (UFPR)

ANDREY RICARDO PIMENTEL
Avaliador Interno (UFPR)

MARIA CLAUDIA FIGUEIREDO PEREIRA EMER
Avaliador Externo (UTFPR)

Resumo

Determinar um conjunto de casos de teste que possua alta probabilidade de revelar defeitos em um *software* é um dos principais objetivos da área de teste de *software*. Dentre os vários critérios propostos na literatura destaca-se a Análise de Mutantes, uma abordagem promissora devido a sua capacidade em revelar defeitos, embora possua um custo computacional relativamente alto. Com o intuito de reduzir o custo da Análise de Mutantes, estudos empregam a utilização de mutação de ordem superior (*Higher Order Mutants*, HOMs). O uso de HOMs tem se destacado por reduzir o número de mutantes equivalentes, reduzir o esforço do teste e simular defeitos próximos dos defeitos reais. Entretanto, a geração dos melhores HOMs é uma tarefa complexa, devido ao grande número de mutantes que podem existir e a outros fatores que influenciam a geração, tais como a eficácia dos HOMs gerados. Trabalhos têm aplicado com sucesso técnicas da área da Engenharia de *Software* baseada em busca por meio da utilização de técnicas de otimização para solucionar esse problema. Entretanto, há ainda a necessidade de possuir um conhecimento sobre o comportamento do problema, de modo a determinar a melhor estratégia a ser utilizada, como projetar e configurar os algoritmos, escolhendo os diferentes operadores de busca e definindo seus parâmetros, para assim melhorar o direcionamento da busca. Neste sentido, o uso de hiper-heurística possibilita uma abordagem mais flexível para automatizar estas tarefas. Além disso, o uso de uma hiper-heurística de seleção de diferentes estratégias existentes para geração de HOMs pode ser útil para reduzir o esforço do testador. Diante disso, este trabalho propõe uma abordagem multi-objetivo que utiliza o conceito de hiper-heurística para gerar conjuntos de HOMs, denominada *Hyper-Heuristic for Generation of Higher Order Mutants* (HG4HOM). O objetivo é selecionar a menor quantidade de HOMs, os mais difíceis de serem mortos e assim melhorar a eficácia do teste, além de também possibilitar que ao matar um HOM seus FOMs (*First Order Mutants*) constituintes também possam ser mortos. Para isso, a abordagem é implementada e avaliada com dois algoritmos multi-objetivos: NSGA-II e SPEA2, e três métodos de seleção: *Choice-Function* (CF), *Fitness-Rate-Rank based Multi-Armed Bandit* (FRR-MAB) e a seleção aleatória (*Random*). O algoritmo SPEA2 utilizando o conceito de hiper-heurística juntamente com o método de seleção CF obteve os melhores resultados. Quando comparado com as estratégias tradicionais, a abordagem obteve resultados próximos em relação ao escore de mutação e valor equivalente ao melhor em relação ao tamanho do conjunto de casos de teste adequado. A abordagem obteve as soluções com melhores valores de *Euclidean Distance* considerando os objetivos relacionados a encontrar a menor quantidade de HOMs, os mais difíceis de serem mortos e capazes de substituírem seus FOMs constituintes.

Palavras-chave: Teste de *Software*, Análise de Mutantes, Mutação de Ordem Superior, Algoritmos Evolutivos Multi-Objetivos, Hiper-Heurística.

Abstract

One of the main testing goals is to determine test sets with a high probability of revealing faults. Mutant Analysis is a promising criterion due to its ability to reveal faults, although with a high computational cost. In order to decrease the mutation testing cost, studies employ the use of Higher Order Mutants (HOMs). The use of HOMs can contribute to decrease the number of equivalent mutants, decrease the test effort and simulate faults close to the real ones. However, the generation of the best HOMs is a complex task, due to the large number of mutants that may exist, and to other factors that influence the generation, such as the efficacy of the generated HOMs. To solve such a problem, some works have successfully applied Search-based Software Engineering techniques through the use of optimization techniques. However, it is still needed to have knowledge about the problem behavior, to determine the best strategy to be applied, and to know how to design and configure the algorithms by choosing the different search operators and defining their parameters in order to improve the search. In this sense, the use of hyper-heuristics allows a more flexible approach to automating these tasks. Also, the use of a hyper-heuristic for selection of different existing strategies to generate HOMs can be useful to reduce the tester's effort. Considering all these facts, this work proposes a multi-objective approach, called Hyper-Heuristic for Generation of Higher Order Mutants (HG4HOM), which uses the hyper-heuristic concept to generate sets of HOMs. The goal is to select a small number of HOMs which are difficult to kill, and that contribute to improve the test efficacy, that is, it is desired the test cases that kill the selects HOMs are also capable of killing their corresponding FOMs (First Order Mutants). The approach is implemented and evaluated with two multi-objective algorithms: NSGA-II and SPEA2, and three selection methods: Choice-Function (CF), Fitness-Rate-Rank based Multi-Armed Bandit (FRR-MAB), and random selection (Random). The SPEA2 algorithm using the hyper-heuristic concept together with the CF selection method obtained the best results. In comparison with respect to the traditional strategies, the approach achieved similar results related to the mutation score and statically equivalent values to the best strategy considering the size of the adequate test case sets. The approach obtained the best results when considering the Euclidean Distance values of the solutions with respect to the goals proposed.

Keywords: Software Testing, Mutation Analysis, Higher Order Mutation, Multi-objective Evolutionary Algorithms, Hyper-heuristic.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

AOS	*Adaptive Operator Selector*
CF	*Choice Function*
DF	*Decayed Factor*
EA	*Evolutionary Algorithm*
ED	*Euclidean Distance*
ER	*Error Ratio*
FIR	*Fitness Improvement Rate*
FOM	*First Order Mutant*
FRR-MAB	*Fitness-Rate-Rank based Multi-Armed Bandit*
HOM	*Higher Order Mutant*
FIFO	*First In, First Out*
HG4HOM	*Hyper-Heuristic for Generation of Higher Order Mutants*
irace	*Iterated Race for Automatic Algorithm Configuration*
HV	Hipervolume
LLH	*Low-Level Heuristic*
MAB	*Multi-Armed Badit*
MOEA	*Multi-Objective Evolutionary Algorithm*
MOP	*Multi-Objective Optimization Problem*
MS	*Mutation Score*
NSGA-II	*Non-dominated Sorting Genetic Algorithm*
PCH	Processo de Criação de HOMs
Q-Q	Quantil-Quantil
SBSE	*Search Based Software Engineering*
SOM	*Second Order Mutant*
SPEA2	*Strength Pareto Evolutionary Algorithm 2*
SSHOM	*Strongly Subsuming HOM*
SW	*Sliding Window*
TC	*Test Cases*
UCB	*Upper Confidence Bound*
WSD	*Weakly Subsuming and De-coupled*

Capítulo 1

Introdução

A Análise de Mutantes [26] é um critério de teste que utiliza os principais defeitos aos quais um programa está sujeito para geração e avaliação de um conjunto de casos de teste. Para isso, são gerados programas denominados mutantes a partir do programa em teste, com pequenas mudanças sintáticas (mutações). Então, é aplicado um dado conjunto de casos de teste no programa em teste e nos mutantes. Se o mutante produzir um resultado diferente do programa original ele é considerado um mutante morto. Neste caso, se o resultado do mutante estiver correto e o do programa não, então, um defeito foi revelado. Ao final, uma métrica chamada escore de mutação é gerada para fornecer uma medida objetiva que revela o nível de confiança dos casos de teste analisados. Essa métrica relaciona o número de mutantes mortos por um conjunto de casos de teste com o número de mutantes gerados. Portanto, quanto maior for o valor do escore de mutação maior será a capacidade dos casos de teste matarem os mutantes e consequentemente maior será a qualidade dos casos de teste.

Os mutantes gerados para o teste de mutação podem ser classificados quanto ao número de mudanças sintáticas que possuem. Os mutantes que possuem apenas uma mudança sintática são denominados mutantes de primeira ordem (*First Order Mutants*, FOMs) e os mutantes com mais de uma mudança sintática são denominados mutantes de ordem superior (*Higher Order Mutants*, HOMs) [50].

Embora o teste de mutação seja capaz de avaliar adequadamente um dado conjunto de casos de teste, esse critério possui problemas relacionados ao alto custo computacional e alto esforço humano envolvido [64]. De modo a reduzir o custo do teste de mutação, estratégias têm sido propostas visando a melhora da eficiência e a viabilidade do teste [25]. As estratégias são agrupadas de acordo como seu objetivo em estratégias para [64]: (i) redução de mutantes e; (ii) redução do custo de execução. Este trabalho tem como foco a redução do número de mutantes por meio da utilização de mutantes de ordem superior. No contexto de redução de mutantes, estudos empíricos comprovam que a utilização de mutantes de ordem superior pode reduzir o número de mutantes equivalentes [65, 105], reduzir o esforço do teste [61, 63, 89, 110] e simular defeitos próximos a defeitos reais. O defeito que se aproxima a um defeito real é o defeito dito complexo, geralmente composto de diferentes defeitos (modificações) simples que podem ser simulados apenas com o uso de HOMs [33, 50, 99, 113]. Entretanto, há o problema relacionado com o procedimento de geração de HOMs, uma vez que o número (espaço) de HOMs cresce exponencialmente em relação ao número de FOMs [61].

De modo a tratar o problema da geração de HOMs, sem diminuir a eficácia dos testes (dada pelo escore de mutação), estudos propuseram estratégias relacionadas à forma de geração dos HOMs. Dentre eles pode-se destacar o trabalho de Polo et *al.* [110] e o trabalho de Papadakis e Malevris [105].

Polo et *al.* [110] propuseram três estratégias para geração de mutantes de segunda ordem (*Second Order Mutants*, SOMs), considerando uma lista de FOMs, dada pela ordem na qual eles são gerados. São elas: *LastToFirst* - combina o primeiro mutante com o último da lista, o segundo com o penúltimo e assim por diante; *DifferentOperators* - combina os FOMs que foram gerados por diferentes operadores de mutações e; *RandomMix* - combina mutantes aleatoriamente.

Papadakis e Malevris [105] estenderam o estudo de Polo et *al.* [110] e introduziram cinco novas estratégias: *First2Last*, *SameNode*, *SameUnit*, *SU_F2Last* e *SU_DiffOp*. *First2Last* ordena os FOMs em relação à ordem em que o trecho de código que foi alterado aparece no código fonte e combina os mutantes com o procedimento semelhante ao da estratégia *LastToFirst*. *SameNode* combina os mutantes a partir do mesmo bloco base. *SameUnit* combina os mutantes a partir da mesma unidade do programa. *SU_F2Last* e *SU_DiffOp* combinam os mutantes baseadas na aplicação das estratégias *First2ToLast* e *DifferentOperators*, aplicando-as individualmente para cada unidade do programa.

Diferentes fatores precisam ser considerados na construção de HOMs. Há grande complexidade em encontrar HOMs que não sejam equivalentes, que sejam iguais ou mais difíceis de serem mortos que seus FOMs constituintes, além da grande quantidade de HOMs que pode existir. Desse modo, o problema de geração de HOMs pode ser tratado como um problema de otimização. Os problemas de otimização da Engenharia de *Software* vêm sendo eficientemente resolvidos no campo de pesquisa denominado Engenharia de *Software* Baseada em Busca (*Search Based Software Engineering*, SBSE) [53] que aplica técnicas baseadas em busca para a construção de soluções para diversos problemas de Engenharia de *Software*, tais como problemas das áreas de teste [91] e projeto de *software* [115], dentre outros.

Algoritmos de busca também vêm sendo utilizados com sucesso para a geração de HOMs [50,51,61,71,96,101,102,104]. Destacam-se os trabalhos de Jia e Harman [61], Landgon et *al.* [71], Harman et *al.* [52], Nguyen e Madeyski [96], e Omar et *al.* [104]. Jia e Harman [61] utilizam otimização mono-objetivo com os algoritmos de Busca Gulosa (*Greedy*), Genético e Subida de Encosta (*Hill Climbing*), para encontrar HOMs que sejam capazes de substituir seus FOMs constituintes sem perder a eficácia dos testes (*Strongly Subsuming* HOM, SSHOM), sendo que esses HOMs foram encontrados em todos os experimentos.

Landgon et *al.* [71] aplicam uma abordagem diferente ao utilizar Programação Genética (*Genetic Programming*, GP) junto com o NSGA-II (Non-dominated Sorting Genetic Algorithm - II). O NSGA-II é utilizado apenas para a seleção dos HOMs e o algoritmo de GP é encarregado da construção e avaliação dos HOMs. O objetivo desse trabalho é encontrar HOMs mais realísticos e mais difíceis de serem mortos do que os FOMs constituintes.

O trabalho de Harman et *al.* [52] utiliza Algoritmo Genético e Busca Gulosa, sendo que o Algoritmo Genético busca encontrar SSHOMs e a Busca Gulosa visa a remover mutações de um HOM sem prejudicar a sua qualidade. Nesse trabalho os algoritmos obtiveram sucesso e conseguiram substituir os FOMs pelos HOMs em aproximadamente 45%.

Nguyen e Madeyski [96] utilizaram os algoritmos NSGA-II, εMOEA [23,66], εNSGA-II [1,66] e NSGA-III para gerar e buscar HOMs, avaliando o desempenho em encontrar SSHOMs. Dentre os algoritmos analisados o εNSGA-II apresentou os melhores resultados.

Omar et *al.* [104] buscaram encontrar HOMs com alta probabilidade de revelar defeitos não revelados por seus FOMs constituintes (*subtle* HOMs). Para isso, utilizaram seis técnicas: Algoritmo Genético, Busca Local, Busca Local Guiada por Interação de Dados (BLGD), Busca Local Guiada por Casos de Teste (BLGC), Busca Aleatória Restrita e Busca Enumerada Restrita. As técnicas de Busca Local, BLGD e BLGC foram mais eficientes do que as outras técnicas em encontrar *subtle* HOMs.

Considerando a utilização das técnicas de otimização no contexto de HOMs e os trabalhos descritos, observa-se que, embora alguns trabalhos apliquem uma abordagem de otimização mono-objetivo, o problema é de fato multi-objetivo, pois é impactado por muitos fatores, tais como: número de casos de teste, dificuldade de um mutante ser morto e alto escore de mutação.

Implementar uma solução com algoritmos de otimização baseados em busca, seja com um objetivo (mono-objetivo) ou com mais de um objetivo (multi-objetivo), pode ser uma tarefa difícil, principalmente para não especialistas do campo de otimização que podem ter dificuldades para realizar adaptações na implementação para o problema a ser tratado, escolher os algoritmos mais adequados, e escolher uma configuração de parâmetros adequada ao algoritmo selecionado. Essas escolhas não possuem um guia de como proceder [14] e para isso, alguns pesquisadores utilizam hiper-heurísticas.

Hiper-heurística é uma técnica capaz de gerenciar heurísticas de baixo nível (*Low-Level Heuristics*, LLH) aplicando operadores durante a otimização do problema de maneira automática e sem a necessidade de supervisão [14]. Segundo Burke et *al.* [14], o uso de hiper-heurística possibilita uma abordagem mais flexível e com uma metodologia que automatiza o projeto (*design*) e a configuração (*tunning*) dos algoritmos.

A hiper-heurística é apontada por alguns autores como uma tendência e tema de pesquisa futura na área de SBSE [3, 49, 98], assim, na literatura existem poucos trabalhos que aplicam hiper-heurísticas [8, 17, 35, 36, 48, 60, 69, 84, 121] para resolver problemas da Engenharia de *Software*. Dentre eles é relevante destacar os trabalhos de: Basgalupp et *al.* [8] que visam à geração de algoritmos que criam árvores de decisão utilizadas na predição de esforço de *software*; Jia e Harman [60] que avaliam o aprendizado e a aplicação das estratégias de teste combinatorial; Kumari et *al.* [69] que visam à resolução do problema de clusterização de módulos; Carvalho et *al.* [17], Guizzo et *al.* [48] e Mariani et *al.* [84] que buscam determinar uma sequência de módulos para o teste de integração; e Strickler et *al.* [121], Ferreira et *al.* [35] e Ferreira et *al.* [36] que utilizam hiper-heurística no teste de mutação com Linha de Produto de *Software*. Entretanto, tais trabalhos não abordam a aplicação de hiper-heurística no contexto de HOMs.

1.1 Justificativa

Diante do contexto apresentado, o presente trabalho é justificado por meio das seguintes motivações:

1. A análise de mutantes é um critério eficaz em revelar defeitos e o uso de HOMs auxilia a reduzir o número de mutantes equivalentes, reduzir o esforço do teste, simular defeitos próximos a defeitos reais e possibilita aumentar a qualidade do conjunto de casos de teste;

2. Foi observado que a geração de HOMs pode ser tratada como um problema de otimização multi-objetivo. Entretanto, na literatura há poucos trabalhos que utilizam algoritmos multi-objetivos para a construção, avaliação e seleção de HOMs;

3. Existem na literatura várias estratégias para a geração de HOMs. Isso traz a dificuldade para o testador em escolher apenas uma. Além disso, pode ser difícil para o testador que possui pouco conhecimento na área de otimização implementar uma solução baseada em busca;

4. Na literatura não existem trabalhos que utilizem o conceito de hiper-heurística no contexto de teste de mutação de ordem superior. A praticidade e a abstração em relação ao domínio

do problema, proporcionadas por esse conceito, são vantagens a serem consideradas no auxílio ao testador no teste de mutação com HOMs, afim de que a escolha das estratégias para geração de HOMs seja feita automaticamente.

1.2 Objetivos

No teste de *software* o testador necessita avaliar um *software* de forma prática, rápida e eficaz, principalmente quando se faz uso do critério de análise de mutantes que possui um alto custo computacional associado. Para isso deve-se buscar reduzir esses custos, por exemplo, através da redução do número de mutantes, redução do número de casos de teste necessários e da determinação de mutantes que revelem mais defeitos. Um alternativa é o uso de uma abordagem baseada em busca, por exemplo, na determinação de um conjunto adequado de mutantes e casos de teste. Entretanto, para aplicar tal abordagem é necessário que o testador possua um conhecimento sobre o comportamento do problema, de modo a determinar a melhor estratégia a ser utilizada, como projetar e configurar os algoritmos, escolhendo os diferentes operadores de busca e definindo seus parâmetros, para assim melhorar o direcionamento da busca.

De modo a auxiliar a reduzir o esforço do testador para realizar as tarefas mencionadas acima, o presente trabalho tem como objetivo geral a utilização de algoritmos multi-objetivos empregando o conceito de hiper-heurística para possibilitar uma abordagem mais flexível para automatizar a geração de conjuntos de HOMs mais adequados a um domínio de um problema a ser investigado sem diminuir a eficácia dos testes.

Para esse objetivo o trabalho propõe a abordagem *Hyper-Heuristic for Generation of Higher Order Mutants* (HG4HOM) que possibilita reduzir os esforços do testador para implementar uma solução baseada em busca, configurar seus algoritmos e utilizar as principais vantagens das diferentes estratégias tradicionais de geração de HOMs (não baseadas em busca) existentes.

A HG4HOM visa a determinar os melhores conjuntos de HOMs, sendo que esses mutantes possam revelar defeitos ainda não revelados, reduzir o número de mutantes ou substituir seus FOMs constituintes sem perder a eficácia do teste com relação ao escore de mutação. Além da redução do esforço do testador, a hipótese desse trabalho é que a HG4HOM apresenta resultados similares aos das estratégias tradicionais.

A configuração da abordagem foi realizada utilizando *irace* [74] e sua implementação é realizada com: (i) dois algoritmos multi-objetivos comumente utilizados no teste de mutação: NSGA-II [24] e SPEA2 [131]; (ii) três métodos de seleção: *Choice Function*, adaptado por Maashi et *al.* [79]; *Fitness-Rate-Rank based Multi-Armed Bandit* (FRR-MAB); e a seleção aleatória (*Random*); (iii) utilização do conceito de dominância [48] na avaliação da qualidade da aplicação das heurísticas de baixo nível; (iv) utilização de quatro estratégias para a geração de HOMs [89, 110]: *FirstToLast*, *RandomMix*, *Different-Operators* e *Each-Choice*; (v) utiliza como operadores de mutação as estratégias de geração de HOMs combinadas com a manipulação de uma solução por meio da adição e troca de HOMs. Além desses operadores é utilizado o operador de remoção de HOMs, o qual não utiliza nenhuma estratégia; e (vi) as heurísticas de baixo nível são compostas de um operador de mutação e um operador de recombinação.

O desempenho dos algoritmos é avaliado por meio de indicadores de qualidade utilizados na área de otimização, além de serem utilizados os valores de *fitness* obtidos, escore de mutação e número de casos de teste para comparação com as estratégias tradicionais.

1.3 Organização do Trabalho

O presente trabalho está organizado de acordo com a seguinte maneira:

- **Capítulo 2 - Teste de *Software***: neste capítulo são apresentados conceitos sobre teste de *software* e a importância de sua aplicação, além de técnicas e critérios de teste, enfatizando o teste de mutação e suas ferramentas de auxílio.
- **Capítulo 3 - Otimização Multi-Objetivo e Hiper-Heurísticas**: neste capítulo são abordados os principais conceitos sobre otimização multi-objetivo, os algoritmos utilizados para o propósito da otimização e conceito de hiper-heurística.
- **Capítulo 4 - Trabalhos Relacionados**: neste capítulo são apresentados os trabalhos relacionados, sendo descritos alguns trabalhos encontrados que propõem estratégias de geração de mutantes de ordem superior e trabalhos que aplicam hiper-heurística na área de SBSE.
- **Capítulo 5 - Abordagem baseada em Hiper-heurística para Geração de Mutantes de Ordem Superior**: neste capítulo é apresentada a abordagem proposta e descritos os seus aspectos de implementação.
- **Capítulo 6 - Avaliação Experimental**: neste capítulo é apresentada a metodologia do estudo empírico conduzido para avaliar a abordagem, bem como os resultados obtidos e ameaças à validade.
- **Capítulo 7 - Conclusão**: neste capítulo são apresentadas as conclusões do trabalho e também são descritos alguns trabalhos futuros.

Capítulo 2

Teste de Software

O desenvolvimento de um *software* não é uma atividade simples, podendo se tornar complexa dependendo do tamanho e domínio da aplicação. Dessa maneira, a atividade está sujeita a diversos tipos de problemas que resultam em produtos finais diferentes do esperado [25]. Durante o desenvolvimento, dentro do processo de garantia da qualidade, existem práticas agrupadas com o nome de Validação, Verificação e Testes (VV&T). Essa série de atividades possui como objetivo garantir que tanto o *software* construído quanto o produto em si estejam de acordo com as especificações. Atividades de Validação e Verificação (V&V) podem consumir cerca de 50% a 60% do custo total no ciclo de vida de um *software* [106].

Dentre essas atividades o teste é uma das mais aplicadas, constituindo-se um dos elementos para fornecer evidências da confiabilidade do *software* em complemento a outras atividades como, por exemplo, o uso de revisões e de técnicas formais e rigorosas de especificação e de verificação [82]. Assim, tal atividade é considerada um elemento importante na garantia da qualidade do *software* [25]. Entretanto, mesmo identificando a importância do teste, sabe-se muito menos sobre essa atividade do que em relação a outros aspectos e/ou atividades do desenvolvimento de *software* [92].

No campo de Engenharia de *Software* há muitas terminologias e para melhor entendê-las alguns termos relacionados ao teste de *software* foram definidos pela IEEE [114]. São eles:

- Engano (*Mistake*): Ação humana que produz um defeito;
- Defeito (*Fault*): Passo, processo ou definição de dados incorretos em um programa de computador;
- Erro (*Error*): Estado intermediário incorreto ou resultado inesperado na execução do programa.
- Falha (*Failure*): A incapacidade de um sistema ou componente para desempenhar as suas funções requeridas dentro dos requisitos de desempenho especificados.

Deve-se considerar que todo programa contém defeitos, o teste irá apenas revelá-los [92]. Dessa forma, o teste de *software* pode ser definido como o processo de execução de um programa com o intuito de revelar defeitos, além de demonstrar que um programa trabalha aparentemente como o desejado, fornecendo indicações de confiabilidade e qualidade do *software*.

Na atividade de teste são comumente utilizados casos de teste com o intuito de avaliar um programa. O caso de teste possui um conjunto de dados de entrada (dados de teste), informações sobre as condições de execução, e informação sobre a saída esperada ao executar o teste [114]. De modo a evitar uma identificação equivocada do resultado de um caso de teste esse é baseado

em dois componentes [92]: (i) uma descrição dos dados de entrada para o programa; (ii) uma descrição precisa da saída correta do programa para esse conjunto de dados de entrada. Um caso de teste, para ser considerado adequado, deve ter alta probabilidade de revelar um defeito ainda não descoberto. O caso de teste é dito bem sucedido se conseguir revelar um defeito.

2.1 Técnicas e Critérios

Um fator a ser considerado na atividade de teste é o consumo do tempo e o custo. Com o intuito de reduzir custos e de modo a auxiliar na condução e avaliação do teste de *software*, técnicas e critérios têm sido propostos. As técnicas diferem-se quanto a origem das informações utilizadas para definir ou avaliar casos de teste, bem como os requisitos das atividades de teste [92], sendo que cada técnica possui uma variedade de critérios para esse fim. As técnicas de teste são geralmente classificadas em três tipos: teste funcional (teste caixa-preta/*black-box testing*); teste estrutural (teste caixa-branca/*white-box testing*); e teste baseada em erros (*fault-based testing*).

A técnica funcional visa a testar os requisitos funcionais do *software* analisando o seu comportamento, demonstrando que as suas funções são operacionais, ou seja, a entrada de um dado de teste é adequadamente aceita e a saída é corretamente produzida. Essa técnica possui como característica a não consideração de detalhes da implementação, tratando seu conteúdo como uma caixa, e avalia o *software* através da perspectiva do usuário [92], o qual está interessado em utilizar o programa. Devido a tais características essa técnica recebeu a denominação de Teste Caixa-Preta [112]. Nessa abordagem, os dados de teste são derivados das especificações [92], porém há o problema de que, geralmente, a especificação existente é informal. Exemplos de critérios utilizados por essa técnica são [112]: Análise do Valor Limite, Grafo de Causa-Efeito e Particionamento em Classe de Equivalência.

A técnica estrutural, ao contrário da técnica funcional, avalia o comportamento interno do *software*. Dessa maneira, requer a análise do código fonte e a seleção de casos de teste que possibilitem exercitar partes do código e não de sua especificação [112]. Assim, essa estratégia deriva dados de teste através da análise lógica do programa [92]. Essa técnica possibilita isolar partes de um programa com o intuito de avaliar o comportamento da função escolhida e aspectos como teste de condição, teste de fluxo de dados, teste de ciclos e teste de caminhos lógicos [112], permitindo que os testes projetados sejam mais precisos. Os critérios utilizados por essa técnica baseiam-se em diferentes tipos de estruturas para determinar as partes do programa que têm a sua execução requerida, podendo ser classificados em [112]: baseados em fluxo de dados, baseados no fluxo de controle e baseados na complexidade. Os critérios baseados no fluxo de controle possuem os seus requisitos de teste derivados apenas das característica de controle da execução do programa, como comandos ou desvios. Os critérios mais conhecidos são: todos-nós, todos-arcos, todos-caminhos. Os critérios baseados no fluxo de dados possuem os seus requisitos de teste derivados do fluxo de dados do programa, explorando as interações que envolvem definições de variáveis e referências a tais definições [116]. Exemplos desta classe de critérios são os critérios de Rapps e Weyuker [116] e os critérios Potenciais-Usos [82]. Os critérios baseados na complexidade possuem os seus requisitos de teste derivados da complexidade de um programa, um exemplo é a complexidade ciclomática (ou de McCabe) [112].

A técnica baseada em defeitos utiliza informações sobre os defeitos mais frequentes para derivar casos de teste, bem como sobre os tipos de defeitos que se deseja revelar. Assim, baseia-se na inclusão de defeitos propositais (artificiais) como forma de revelar defeitos existentes previamente (naturais), fornecendo indicadores para gerenciar o processo de teste (porcentagem de defeitos remanescentes, qualidade dos casos de teste). O destaque da técnica está nos defeitos que podem ser cometidos pelo programador ou projetista durante o desenvolvimento e nas

abordagens empregadas que possibilitam detectar a ocorrência destes defeitos. Exemplos dos critérios dessa técnica são: Semeadura de Erros (*Error Seeding*) [13] e Análise de Mutantes (*Mutation Analysis*) [26]. Destaca-se a Análise de Mutantes que vem se mostrando promissora devido a sua capacidade de relevar defeitos, apesar do seu custo computacional relativamente alto [64].

Através dos critérios de teste é possível selecionar e avaliar casos de teste de modo a aumentar a capacidade de provocar falhas ou, quando isso não for possível, estabelecer um elevado nível de confiança na correção do *software* [82]. Os critérios podem ser classificados de três formas, de acordo com a maneira como são utilizados: cobertura dos testes, adequação dos casos de teste e geração de casos de teste. Os critérios de cobertura dos testes provêm a identificação de componentes do programa que devem ser executados de modo a garantir a qualidade do *software* e informar quando o mesmo foi suficientemente testado [116]. Os critérios de adequação de casos teste são utilizados para verificar se um dado conjunto de casos de teste é suficiente ou não para avaliar um *software* ou uma função [82]. Os critérios de geração de casos de teste são utilizados para gerar, por meio de regras e diretrizes, um conjunto de casos de teste adequados para um *software* ou função que estejam de acordo com o critério de adequação definido anteriormente [82].

2.2 Teste de Mutação

O critério Análise de Mutantes (*Mutant Analysis*), é um critério de teste da técnica baseada em defeitos e foi introduzido por DeMillo et *al.* [26]. Este critério fundamenta-se em dois pressupostos: hipótese do programador competente (*competent programmer hypothesis*) e efeito do acoplamento (*coupling effect*) [26]. O primeiro pressuposto considera que os programadores escrevem códigos que geram programas corretos ou próximos do que é correto. Diante disso, é possível afirmar que os defeitos são introduzidos nos programas através de pequenos desvios sintáticos que alteram a sua semântica. O segundo pressuposto assume que defeitos mais complexos estão relacionados a defeitos simples. Assim, se o teste consegue encontrar defeitos simples no código, então isso pode levar a descoberta de defeitos mais complexos [112].

A análise de mutantes consiste em gerar programas modificados (mutantes) a partir do programa original P. Essas modificações são realizadas através de operadores de mutação que inserem pequenos desvios sintáticos no programa original (código fonte ou objeto). Assim, o programa P e os mutantes são executados com um conjunto T casos de testes e se o programa P apresentar resultados diferentes dos resultados apresentados por um mutante P', então o mutante é considerado morto. Se o programa P retornar resultados incorretos e o mutante resultados corretos, então programa P contém o defeito descrito pela modificação gerada em P'. Para melhorar o conjunto de casos de teste T, o testador pode prover dados de teste adicionais com o intuito de matar os mutantes que não foram mortos. Em algumas situações não é possível matar todos os mutantes, ou seja, não existem casos de teste que gerem resultados diferentes quando aplicados ao mutante. Esses mutantes são chamados mutantes equivalentes. Embora sintaticamente diferentes, o mutante equivalente e o programa original são semanticamente iguais, pois não existe caso de teste que diferencie P e P'. Além disso, existem mutantes que apesar de corretos sintaticamente podem apresentar problemas de execução, tais como uma divisão por zero ou um laço infinito. Estes mutantes são chamados anômalos e devem ser descartados do conjunto total de mutantes.

Um ponto a ser destacado é em relação à classificação da ordem dos mutantes, dado o número de mutações que um mutante possui [50]. Os mutantes que sofrem apenas uma mutação, criados a partir da aplicação de um operador de mutação, são denominados mutantes de primeira

ordem (*First Order Mutants*, FOMs). Os mutantes que sofrem duas ou mais mutações, criados a partir de duas ou mais aplicações de operadores são denominados mutantes de ordem superior (*Higher Order Mutants*, HOMs).

Adicionalmente, para verificar se um caso de teste realmente matou um mutante (ou revelou um defeito), esse deve satisfazer três condições oriundas do modelo RIP (*Reachability*, *Infection*, *Propagation*) [27]:

1. Alcançabilidade: o comando que sofreu mutação deve ser executado (alcançado) pelo caso de teste;

2. Necessidade: após a execução da mutação o programa original e o mutante devem assumir estados intermediários diferentes;

3. Suficiência: o estado da infecção deve ser propagado até o final, e uma saída diferente em relação ao programa original deve ser produzida.

Um dos objetivos do teste de mutação é avaliar o quanto um conjunto de casos de teste é adequado [26], o que é calculado através do escore de mutação (*Mutation Score*, MS) [26] apresentado na Equação 2.1. Essa equação retorna um valor entre 0 e 1. Quanto maior o valor, maior a capacidade dos casos de teste matarem os mutantes, ou seja, mais adequado é o conjunto de casos de teste para o programa em teste [25].

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

Onde:

- $DM(P,T)$: número de mutantes de P mortos pelo conjunto de teste T;
- $M(P)$: número total de mutantes não anômalos gerados a partir do programa P;
- $EM(P)$: número de mutantes equivalentes a P.

Visando a melhorar a aplicabilidade da Análise de Mutantes é relevante destacar alguns pontos de grande importância:

1. Evitar mutantes equivalentes;

2. Selecionar um subconjunto reduzido de mutantes que conduzam a bons casos de teste;

3. Selecionar um subconjunto de casos de teste que sejam eficazes em matar mutantes.

Idealmente busca-se encontrar mutantes não equivalentes, porém a verificação da equivalência entre dois programas exige esforço humano por ser um problema computacionalmente indecidível [64]. Geralmente uma grande quantidade de mutantes é gerada e então selecionar um conjunto de casos de teste adequado requer muito esforço. Além disso, é importante que o conjunto de casos de teste tenha uma boa cobertura, ou seja, mate a maior quantidade de mutantes possíveis e seja pequeno de modo a diminuir o esforço computacional [64].

Devido ao alto custo associado à Análise de Mutantes técnicas têm sido propostas para reduzir o custo da mutação. Essas estratégias visam a reduzir o número e o custo de execução dos mutantes, e são descritas na seção a seguir.

2.3 Estratégias para Redução do Custo de Mutação

Diferentes estratégias para redução do custo da Análise de Mutantes têm sido propostas visando a aumentar a eficiência e a viabilidade da Análise de Mutantes [25], e podendo ser classificadas em dois grupos como apresentados a seguir [64].

2.3.1 Redução de Mutantes

Dentre as principais estratégias para reduzir o número de mutantes estão: Seleção de Mutantes (*Mutant Sampling*), Agrupamento de Mutantes (*Mutant Clustering*), Mutação Seletiva (*Selective Mutation*) e Mutação de Ordem Superior (*Higher Order Mutation*). Através dessas estratégias um subconjunto do total de mutantes gerados é selecionado, reduzindo consequentemente o custo associado, adicionalmente visando a não redução da eficácia do critério dada pelo escore de mutação.

A Seleção de Mutantes [13] seleciona um subconjunto de mutantes do conjunto de mutantes gerados, geralmente através de uma seleção aleatória. Primeiramente todos os mutantes são gerados, posteriormente x% dos mutantes são selecionados para análise e o restante é descartado.

O Agrupamento de Mutantes [58], ao contrário da Mutação Aleatória, escolhe um subconjunto utilizando algoritmos de agrupamento (*clustering algorithms*). Esses algoritmos classificam os mutantes em diferentes grupos de acordo com adequação dos casos de teste, em que é garantido que cada mutante no mesmo agrupamento seja morto pelo mesmo conjunto de casos de teste ou conjuntos similares. Nessa técnica um subconjunto de mutantes de cada agrupamento é selecionado e o restante é descartado.

A Mutação Seletiva visa a construir um menor conjunto de operadores de mutação que geram um subconjunto de todos os mutantes possíveis sem perder a eficácia do teste [100], ou seja, sem que o escore de mutação seja reduzido. Muitas vezes um conjunto denominado essencial de operadores de mutação é utilizado. Desse modo, dado um conjunto de operadores de mutação O, um subconjunto O' é gerado de modo a reduzir o número de mutantes que possam ser mortos por T. Sendo que, T é também capaz de matar todos os mutantes gerados pelo conjunto completo O. Assim, um conjunto reduzido de mutantes e operadores é utilizado, mantendo o escore.

A Mutação de Ordem Superior foi introduzida por Jia e Harman [61], e visa a substituir os FOMs por HOMs. A utilização de HOMs visa a identificar combinações de defeitos simples que representem defeitos complexos [102].

2.3.2 Redução do Custo de Execução de Mutantes

Dentre as principais estratégias utilizadas para a redução do custo de execução de mutantes (tempo de execução e o custo da geração) estão: tradução *bytecode*, esquema de mutante (*Mutant Schema Generation, MSG*), MUSIC (*MUtant Schema Improved with extra Code*), execuções paralelas e em relação ao tipo de mutação. A técnica de execução paralela [88] distribui entre diferentes processadores a execução dos mutantes.

A tradução *bytecode* [77] é uma técnica em que o mutante não precisa ser compilado, o código executável é gerado diretamente, realizando a modificação no arquivo compilado. Essa técnica verifica e realiza a modificação na representação intermediária dos programas Java, o *bytecode*. A vantagem dessa técnica está no processamento que pode ser realizado sem a necessidade do código fonte de um programa e pode ser realizado sob demanda em tempo de

carregamento, quando a máquina virtual Java (*Java Virtual Machine*, JVM) carrega um arquivo de classe.

O esquema de mutante [124] compõe todas as mutações e o código original no mesmo arquivo, então apenas uma cópia do programa em teste é necessária. Através dessa técnica é possível reduzir o espaço de armazenamento necessário para manter todos os arquivos em disco, além da necessidade do carregamento da classe para cada mutante, uma vez que o mesmo já está carregado em memória.

MUSIC [87] melhora o esquema de mutante a um custo reduzido, reduzindo o número total de execuções no Teste de Mutação identificando quando um mutante não necessita ser executado, além de reduzir o tempo gasto na identificação de *time out*.

O tipo de mutação flexibiliza o rigor, aumentando-o ou reduzindo-o quanto ao Teste de Mutação. A redução, realizada pelo testador, em relação ao custo do Teste de Mutação é feita através da criação de casos de teste menos rigorosos. Nesse procedimento apenas uma técnica é utilizada por vez. Dentre as principais pode-se citar: Mutação Forte (*Strong Mutation*), Mutação Fraca (*Weak Mutation*) e Mutação Flexível (*Flexible Mutation* ou *Firm Mutation*).

A Mutação Forte [26] refere-se à mutação tradicional. Onde um mutante P' de um dado programa P é dito morto apenas se dada uma saída os resultados sejam diferentes. Desse modo, os mutantes são executados até o fim, o que contribui para resultados mais precisos, porém gastando muito tempo.

A Mutação Fraca [57] surgiu para otimizar o tempo de execução da mutação forte. Ao contrário da Mutação Forte, a Mutação Fraca não necessita verificar ao fim da execução o resultado diferente, sendo que a verificação ocorre imediatamente logo após o trecho modificado - detecção do defeito. Desse modo, esse tipo de mutação é menos rigoroso em relação à Mutação Forte, o que resulta em resultados menos precisos, no entanto, o tempo gasto é menor.

A Mutação Flexível [129] está entre os tipos de Mutação Forte e Mutação Fraca em termos de eficiência e eficácia visando a superar as desvantagens desses tipos de mutações. Um mutante morto é identificado entre o estado após a execução do defeito (Mutação Fraca) e a saída (Mutação Forte).

2.4 Mutação de Ordem Superior (*Higher Order Mutation*)

Uma mutação de ordem superior gera mutantes que são criados por duas ou mais modificações, ou seja, pela combinação de 2 ou mais mutantes de primeira ordem. Um HOM h é constituído por um conjunto de FOMs $f_1, f_2, ..., f_n$.

Em 1992 o teste com mutação de ordem superior foi introduzido para estudo do efeito de acoplamento [99]. Nesse estudo foi identificado que os casos de teste desenvolvidos para FOMs matavam um maior percentual de mutantes quando aplicado a mutantes oriundos de mutação de segunda ordem (*Second Order Mutation*, SOM).

Durante muitos anos o Teste de Mutação com HOMs foi considerado computacionalmente caro, bem como impraticável [61]. A criação de HOMs ocorre de maneira exponencial [61], dado que um HOM é construído de diferentes FOMs. Dessa maneira, supondo que um dado programa em teste contivesse n FOMs, este iria gerar n^n HOMs [61], inviabilizando assim o seu uso.

Alguns estudos empíricos identificaram que a maioria dos defeitos que conseguem se aproximar a defeitos reais são os defeitos ditos complexos [33, 113]. Todavia, um defeito complexo não pode ser descrito através de apenas uma modificação (FOMs) [99]. Simular tal defeito é possível apenas através de mutação de ordem superior [50]. Destes estudos, há resultados [113] que revelam que a probabilidade de se revelar um defeito introduzindo apenas

uma mudança sintática é de apenas 4% e que 90% dos defeitos encontrados, de fato, são defeitos complexos.

Desse modo, recentemente o Teste de Mutação de Ordem Superior voltou a ser foco e assim há estudos com o intuito de viabilizar uma redução dos custos relacionados a essa abordagem. Estudos nesse contexto têm sugerido estratégias quanto a forma de criação [71, 110], classificação dos tipos de HOMs [61, 101] e evidências da redução da quantidade de mutantes equivalentes [65, 99, 105].

2.4.1 Classificação

Jia e Harman [61] apresentam um estudo para o Teste de Mutação de Ordem Superior de modo a classificá-lo em seis tipos diferentes (Figura 2.1), criando assim a primeira caracterização para HOMs. Essa classificação visa a tratar os HOMs em termos de acoplamento e da relação *subsuming*. O acoplamento segue o pressuposto do efeito do acoplamento [26], onde dado um conjunto de teste que mata os FOMs, este também contém casos que matam o HOM, denominando, assim, esse HOM como um "HOM acoplado" caso contrário o HOM é dito "desacoplado". Já a notação *subsuming* representa os HOMs que são mais difíceis de matar que seus FOMs constituintes.

Na Figura 2.1 são apresentados exemplos da classificação dos HOMs. Cada pseudo[1] diagrama de Venn ilustra uma representação entre um HOM h e seus FOMs constituintes ($f_1, f_2, ..., f_n$), sendo h exemplificado através de um mutante de segunda ordem. Para cada diagrama considera-se: o retângulo como sendo todos os dados de teste possíveis; a região cinza como sendo o conjunto de casos de teste que matam h (T_h); os dois círculos brancos centrais como sendo o conjunto de casos de teste possíveis que matam cada FOM constituinte ($T_1, T_2, ..., T_n$), onde a união deles é representada através de T_f ($T_f = \bigcup_i T_i$) e a intersecção através de $\cap T_f$ ($T_f = \bigcap_i T_i$). Dessa maneira, através de cada diagrama é possível identificar a relação do tamanho do conjunto de casos de teste necessário para matar o HOM e para matar os FOMs.

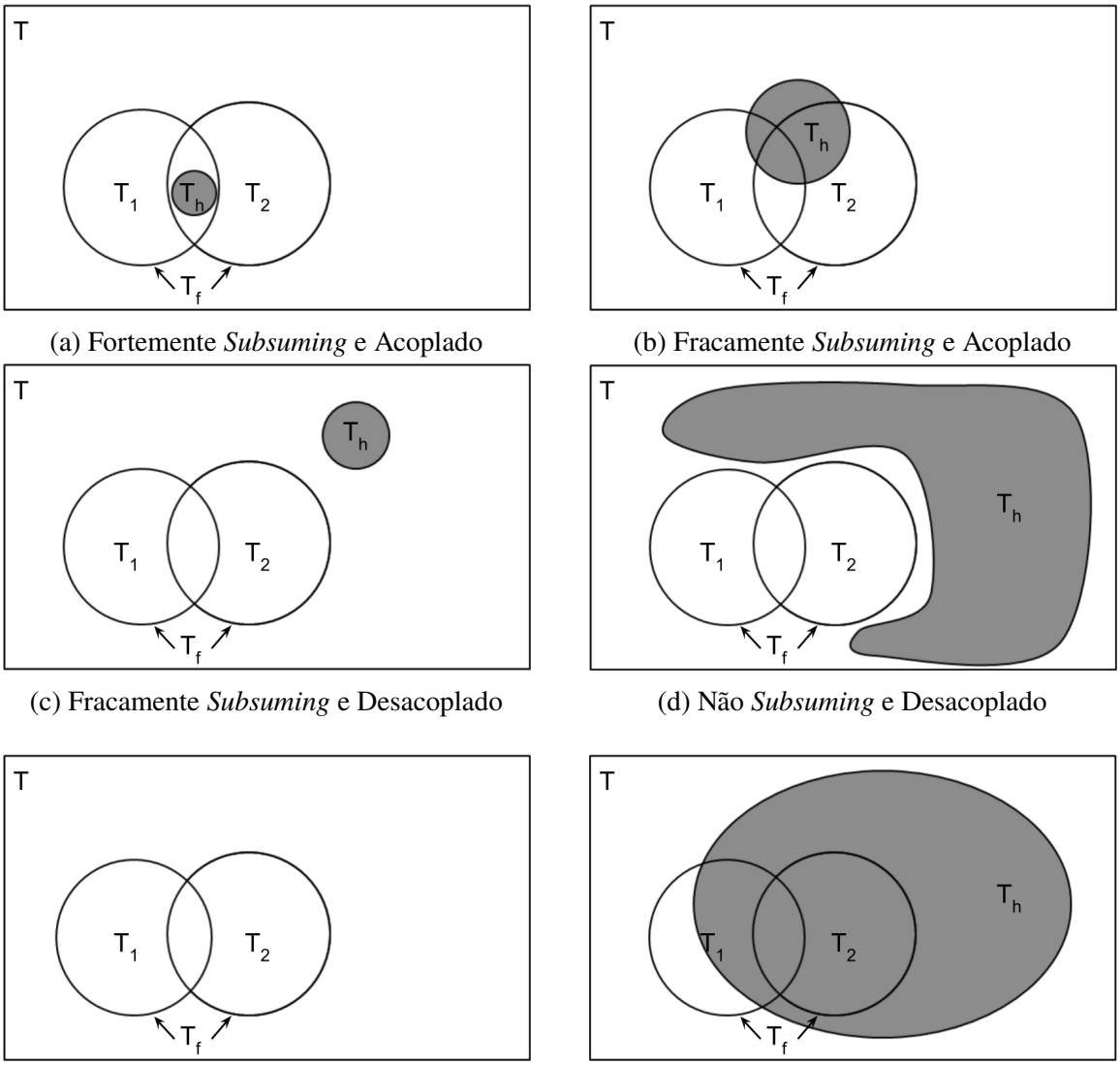
Os HOMs do tipo acoplados são representados na Figura 2.1 através de uma área sombreada (cinza) (T_h) que sobrepõe T_f (Figuras 2.1(a), 2.1(b) e 2.1(f)), caso essa área não sobreponha T_f, esses HOMs são desacoplados (Figuras 2.1(c) e 2.1(d)). Quando não há T_f (Figura 2.1(e)) este é definido como sendo um caso especial de HOM desacoplado, pois não há casos de teste que matam o HOM, sendo então este considerado como mutante equivalente. Os mutantes do tipos *subsuming* são aqueles que possuem T_h menor que T_f (Figuras 2.1(a), 2.1(b) e 2.1(c)), caso contrário são não *subsuming* (Figuras 2.1(d), 2.1(e) e 2.1(f)). A classificação dos tipos de HOMs proposta em [61] é detalhada na Tabela 2.1.

Há ainda outro estudo em relação a classificação dos HOMs, no qual Omar et *al.* [101] introduzem a notação sobre *subtle* HOMs. *Subtle* HOMs representam os HOMs que não são mortos pelos casos de teste que matam todos os seus FOMs constituintes, caso das Figuras 2.1(c) e 2.1(d). Os *subtle* HOMs produzem defeitos complexos que não podem ser simulados através de FOMs e dessa forma, podem ser utilizados como meio de melhorar a eficácia do teste.

2.5 Ferramentas para o Teste de Mutação

Realizar atividades relacionadas ao critério Análise de Mutantes, mesmo considerando a utilização de estratégias para redução de custo, é uma atividade complexa. Por isso, a utilização

[1] O diagrama é dito pseudo diagrama de Venn por ser utilizado para representar o tamanho dos conjuntos de teste e não os conjuntos



(a) Fortemente *Subsuming* e Acoplado

(b) Fracamente *Subsuming* e Acoplado

(c) Fracamente *Subsuming* e Desacoplado

(d) Não *Subsuming* e Desacoplado

(e) Não *Subsuming* e Desacoplado (Equivalente)

(f) Não *Subsuming* e Acoplado

Figura 2.1: Classificação dos tipos de HOMs (adaptada de [50])

Tabela 2.1: Definição dos tipos de HOMs

Tipo (Figura)	**Definição**	**Descrição**
Fortemente *Subsuming* e Acoplado (2.1(a))	$T_h \subseteq \cap T_f$ e $T_h \neq \emptyset$	Chamado também de *strongly subsuming* HOM (SSHOM), esse tipo de HOM é apenas morto por um subconjunto da interseção dos casos de teste que matam cada FOM, e que constitui ($\cap T_f$). A dificuldade de encontrar um caso de teste da intersecção de T_f que mate os FOMs constituintes não é demasiada, porém eles devem também matar o SSHOM e é aí que reside a dificuldade.
Framacamente *Subsuming* e Acoplado (2.1(b))	$\|T_h\| < \|T_f\|$, $T_h \cap T_f \neq \emptyset$ e $T_h \neq \emptyset$	Esse tipo de HOM é caracterizado dessa forma devido a T_h ser menor que T_f (acoplado). Entretanto, somente parte de T_h está na intersecção de T_f, sendo assim fracamente *subsuming*.
Fracamente *Subsuming* e Desacoplado (2.1(c))	$\|T_h\| < \|T_f\|$, $T_h \cap T_f = \emptyset$ e $T_h \neq \emptyset$	Semelhante ao HOM fracamente *subsuming* e acoplado, entretanto, não possui T_h sobrepondo T_f.
Não *Subsuming* e Desacoplado (2.1(d))	$\|T_h\| \geq \|T_f\|$, $T_h \neq \emptyset$ e $T_h \cap T_f = \emptyset$	Possui T_h maior ou igual a T_f e não há sobreposição de T_h em T_f.
Não *Subsuming* e Desacoplado (Equivalente) (2.1(e))	$T_h = \emptyset$	Esse tipo de HOM não possui T_h, sendo considerado equivalente.
Não *Subsuming* e Acoplado (2.1(f))	$\|T_h\| \geq \|T_f\|$ e $T_h \cap T_f \neq \emptyset$	Esse HOM é considerado sem valor, pois sobrepõe a interseção em T_f e é maior que T_f. Assim, o conjunto de casos de teste que matam este HOM é maior do que o conjunto que mata os seus FOMs constituintes, além de que tais conjuntos também são utilizados por seus FOMs constituintes.

de ferramentas de suporte é fundamental. Diversas ferramentas estão disponíveis para diferentes linguagens como C, C++ e Java.

Judy [81] é uma ferramenta desenvolvida em Java com extensões AspectJ, linguagem de Programação Orientada a Aspectos (POA), que implementa a abordagem FAMTA Light (*fast aspect-oriented mutation testing algorithm*) para melhorar o desempenho do Teste de Mutação. Essa ferramenta visa a ser flexível quanto a integração com novos operadores que possam vir a ser implementados. FAMTA Light é uma abordagem que possui as vantagens dos mecanismos de *pointcut* & *advice* da POA para, eventualmente, definir pontos no fluxo do programa (*pointcuts*) durante a execução de um método, para invocar métodos e capturar valores de propriedades numa classe, além de métodos relacionados (*advices*).

Javalanche [119] é um *framework open source* para Teste de Mutação em linguagem Java. Essa ferramenta utiliza apenas um pequeno conjunto de operadores de mutação, entretanto, esses operadores têm se mostrado suficientes no Teste de Mutação. Essa ferramenta ainda utiliza uma técnica de redução do número de FOMs e o número de casos de teste que necessitam ser executados em cada FOM. Uma característica dessa ferramenta é a classificação dos mutantes em relação ao impacto nas funções dos programas.

MuJava [77] (*Mutation System for Java*) é uma das ferramentas existentes para Teste de Mutação em linguagem Java sendo considerado o sucessor do JavaMut [19], antecedendo até mesmo o JUnit, contudo ainda recebendo atualizações. MuJava provê uma grande gama de operadores de mutação para Java, tanto os operadores tradicionais de mutação (adaptados para orientação a objetos) quanto operadores no nível de classe. MuJava gera automaticamente os mutantes, executa-os junto a um conjunto de testes, posteriormente apresenta a pontuação das mutações em relação ao conjunto de teste. As principais funções dessa ferramenta são: (i) geração de mutantes; (ii) análise de mutantes; (iii) gerenciamento de casos de teste fornecidos pelo usuário. Essa ferramenta ainda implementa abordagens que automaticamente detectam alguns tipos de FOMs equivalentes.

Bacterio [86] é uma das mais recentes ferramentas desenvolvidas para aplicações Java e implementa quase todas as estratégias de redução do custo de mutação que têm sido implementadas (exceção da técnica de *compiler integration* do JavaMut que foi substituída pela tradução *bytecode*). Essa ferramenta foi a primeira a implementar *Flexible Weak Mutation* e a melhorar a análise no nível de sistema.

Proteum/IM 2.0 [83] é uma ferramenta, para linguagem C, que aplica tanto o teste de unidade quanto o teste de integração, pois integra em um único ambiente a ferramenta Proteum e Proteum/IM. Ambas as ferramentas, Proteum e Proteum/IM, são ambientes compilados baseadas em interfaces com janelas e *scripts*. Contudo, essas ferramentas não possuem um gerador automático de casos de teste nem apoio à determinação automática de mutantes equivalentes.

A geração, execução e avaliação de HOMs é uma atividade que demanda demasiado tempo e custo computacional, assim, com o intuito de auxiliar no Teste de Mutação de HOMs foram desenvolvidas ferramentas para diversas linguagens. São elas: MILU, HOMAJ e Bacterio (descrito anteriormente).

MILU [62] é uma ferramenta para linguagem C e suporta os 77 operadores de Richard et *al.* [117] para criar o conjunto de FOMs. Essa ferramenta fornece uma linguagem de *script* flexível, a customização de operadores de mutação, redução do custo do Teste de Mutação, utilização de bibliotecas compartilhadas para chamadas de mutantes e suporte através de interface gráfica.

HOMAJ é uma ferramenta Java para AspectJ que utiliza as abordagens propostas em Omar et *al.* [101] determinando a localização da inserção de defeitos em AspectJ, e automatiza o processo de geração de FOMs e HOMs. Essa ferramenta ainda classifica os HOMs, analisa seus resultados e implementa diversas estratégias com o intuito de encontrar *subtle* HOMs, tais como: Algoritmos Genéticos, Busca Local, Busca Aleatória, Busca Guiada e Busca Enumerada. Essa ferramenta utiliza em conjunto a ferramenta MuJava para a geração de FOMs.

Outras ferramentas também foram utilizadas para o Teste de Mutação de Ordem Superior como o Judy em [80] e a Proteum em [105], entretanto, essas ferramentas foram adaptadas para o teste com HOMs.

A existência de ferramentas possibilita auxiliar alunos de Engenharia de *Software* e pesquisadores a adquirirem experiência na comparação, seleção e definição de estratégias de teste, além de conceitos básicos. Adicionalmente, propicia uma maior qualidade e produtividade

para a atividade de teste, bem como reduzir erros humanos quando da aplicação manual do Teste de Mutação, e ainda oferece o suporte ao teste de regressão.

2.6 Considerações Finais

A atividade de teste de *software* utiliza técnicas e critérios que foram propostos com o intuito de reduzir custos e auxiliar na avaliação de conjuntos de teste. Dentre estes critérios destaca-se o critério baseado em defeitos Análise de Mutantes. A Análise de Mutantes vem se mostrando promissora devido a sua capacidade de revelar. De modo a auxiliar na redução do custo da Análise de Mutantes diferentes estratégias têm sido propostas visando à redução de mutantes e ao custo de execução associado, bem como a criação de ferramentas que auxiliem a utilização de tal critério.

Nesse contexto, o presente trabalho visa a reduzir o custo do teste de mutação, reduzindo o número de mutantes. O foco do trabalho é Mutação Forte e o teste de Mutação de Order Superior. Para isso, os HOMs são gerados com o uso de algoritmos de busca, assuntos dos próximos capítulos.

Capítulo 3

Otimização Multi-Objetivo e Hiper-Heurísticas

Comumente as pessoas deparam-se com problemas, por exemplo, encontrar o carro mais econômico com a maior potência e com um preço acessível ao mesmo tempo, entretanto, ao tentar resolver esses problemas existem inúmeras possíveis soluções. O espaço de todas as soluções possíveis é chamado de espaço das soluções, de busca ou de estados. Todavia, encontrar uma solução no espaço de busca não necessariamente resume-se a encontrar o maior ou o menor valor, como no exemplo descrito anteriormente.

Um abordagem de otimização visa a escolher a melhor solução no espaço de busca através de uma função que possua um objetivo específico (função objetivo) e que direciona a busca para um valor desejado [46]. A função objetivo descreve um problema através de uma função matemática.

Os problemas de otimização são classificados quanto ao número de objetivos (funções objetivo) que os compõem, os quais podem possuir ou não restrições. Os problemas que possuem apenas um objetivo a ser otimizado para ser maximizado ou minimizado são denominados mono-objetivo [20]. Já os problemas que possuem mais de um objetivo a ser otimizado são denominados multi-objetivos [20]. Um exemplo de função objetivo seria considerar numa busca o carro x mais econômico, podendo exemplificar com uma função $f(x) = economia(x)$, onde essa função representaria o valor de economia de um carro x.

Os problemas de otimização multi-objetivo (*Multi-Objective Optimization Problems*, MOPs) possuem mais de um objetivo a ser otimizado, geralmente com objetivos conflitantes onde a melhora de um objetivo piora os demais. Nesse contexto não há apenas uma solução adequada, mas sim um conjunto de soluções [20]. Assim, busca-se nos MOPs encontrar um conjunto de soluções que possuam o melhor *trade-off* entre os objetivos de modo a otimizar simultaneamente todos os objetivos do problema [20].

Dentro do espaço de busca é possível encontrar inúmeras soluções com bons *trade-offs* (equilíbrio entre os objetivos), porém é necessário determinar quando os objetivos de uma solução são melhores ou piores do que outra solução. Para isso utiliza-se o conceito de Dominância de Pareto [107] ilustrado na Figura 3.1.

Por meio desse conceito é possível determinar as soluções que possuem os melhores *trade-offs* dentro do espaço de busca em relação as outras soluções encontradas, denominadas soluções não-dominadas. Essas soluções formam a Fronteira de Pareto [20]. Na Figura 3.1 considera-se que ambos os objetivos F_1 e F_2 sejam de maximização, assim, os pontos em negrito representam as soluções não-dominadas e as soluções dominadas são representadas em branco. É possível determinar se uma solução S_1 domina ($\prec$) uma solução S_2 quando [20]:

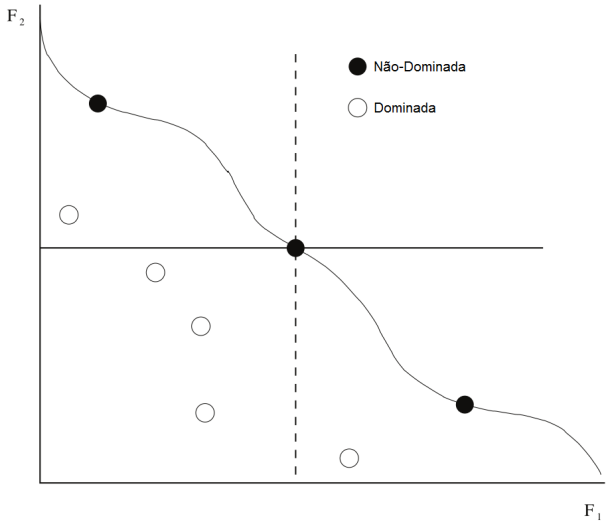


Figura 3.1: Dominância de Pareto (adaptada de [20])

- S_1 não é pior que S_2 em todos os objetivos;
- S_1 é melhor em pelo menos um dos objetivos.

Meta-heurísticas são geralmente utilizadas para resolver MOPs. As meta-heurísticas são estratégias genéricas e/ou estruturas algorítmicas de busca que exploram eficientemente um espaço de soluções de diversos problemas de otimização [76]. As meta-heurísticas usam algoritmos em problemas onde se conhece pouco sobre a forma de uma solução ótima e não se sabe como encontrá-la. Um dos tipos de meta-heurísticas mais conhecidos são os algoritmos evolutivos descritos a seguir.

3.1 Algoritmos Evolutivos

Os algoritmos evolutivos (*Evolutionary Algorithms*, EAs) são baseados na evolução biológica [20] (processos evolutivos que ocorrem na natureza). Na natureza uma população de indivíduos é submetida a uma seleção natural dentro de seu ambiente, onde os indivíduos impróprios e fracos são confrontados com a extinção e os indivíduos fortes têm maiores chances de passarem seus genes às próximas gerações através da reprodução. Durante as gerações, os indivíduos mais aptos conseguem propagar os seus genes mais frequentemente, de modo que estes sejam dominantes na população. Os novos indivíduos são gerados a partir dos antigos, sofrendo mutações que podem oferecer vantagens na sobrevivência. Caso essas mudanças sejam mal sucedidas elas são eliminadas pela seleção natural. O fluxo do processo dos EAs é ilustrado através da Figura 3.2.

Os EAs são baseados em população, a qual é um conjunto de soluções candidatas (cromossomos). Essa população é evoluída geracionalmente, onde em seu processo reprodutivo são selecionados K pais e gerados, geralmente, dois filhos. Esses filhos são submetidos a uma recombinação de seus genes e também a mutação. O processo reprodutivo ocorre até que um número x de filhos seja gerado. Ao fim do processo reprodutivo apenas as soluções mais aptas (com melhor valor de *fitness*) são passadas à próxima geração. Por fim, dado um critério de parada para o processo geracional, é possível identificar a população final que representa as melhores soluções encontradas para o problema. O valor de *fitness* representa o quão boa uma solução é em relação aos seus objetivos [20].

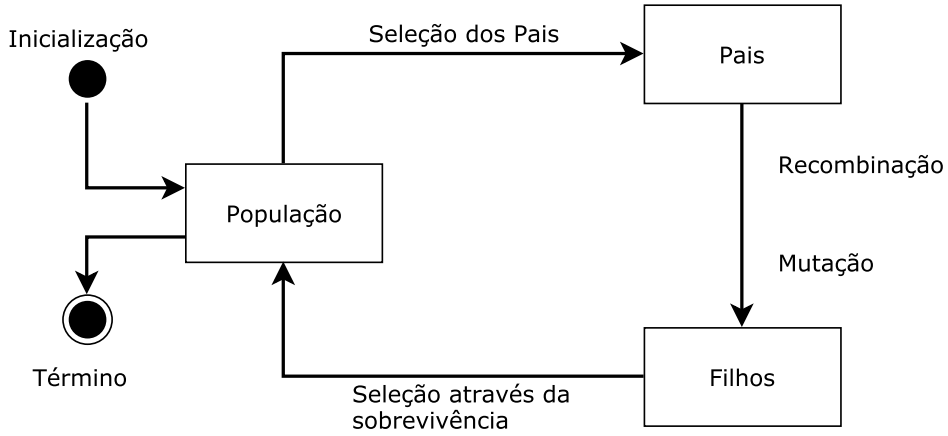


Figura 3.2: Fluxo do processo de um algoritmo evolutivo (adaptada de [31])

Algoritmos Evolutivos também são utilizados para resolver MOPs sendo denominados *Multiobjective Evolutionary Algorithms* (MOEAs). Dentre esses algoritmos o *Non-dominated Sorting Genetic Algorithm II* (NSGA-II) [24] e o *Strength Pareto Evolutionary Algorithm 2* (SPEA2) [131] destacam-se devido aos seus bons resultados, facilidade de implementação, além de serem utilizados com frequência na área de SBSE (*Search-Based Software Engineering*) [20]. Esses algoritmos são de especial interesse para esse trabalho e por isso são detalhados a seguir.

3.1.1 *Non-dominated Sorting Genetic Algorithm II* (NSGA-II)

O NSGA-II [24] é considerado um dos algoritmos mais eficazes para resolução de problemas multi-objetivos. A característica desse algoritmo é classificar as fronteiras através de dois procedimentos (algoritmos) de ordenação: a ordenação da população em um *ranking* por não-dominância (*Fast Non-dominated Sorting Algorithm*) e a ordenação individual por *crowding distance* (*Crowding Distance Sorting*), buscando por soluções bem distribuídas no espaço [20, 24]. O Pseudocódigo do NSGA-II é apresentado no Algoritmo 3.1.

Primeiramente, no NSGA-II, uma população P_0 de tamanho N deve ser gerada, onde, de forma padrão, geram-se soluções aleatórias e posteriormente avaliam-se as mesmas. O algoritmo então gera a população filha Q_t, com base na população pai P_t. Para isso, é realizada a seleção, seguida do cruzamento e mutação, de acordo com os operadores desejados. Posteriormente à geração dos novos indivíduos (filhos), cada indivíduo filho gerado é avaliado e seus valores objetivos são atribuídos. Após unem-se a população pai e a população filha em R_t, aplicam-se *Fast Non-Dominated Sorting* e *Crowding Distance Sorting* nessa união das populações gerando F. Enquanto o número máximo de gerações g não é atingido (critério de parada), cada iteração t executa uma série de passos e os melhores de F são passados à próxima geração. Para auxiliar a visualização, a Figura 3.3 apresenta o processo de iteração do NSGA-II.

O algoritmo de ordenação *Fast Non-dominated Sorting* é apresentado através do Algoritmo 3.2. *Fast Non-dominated Sorting* busca ordenar uma população em diferentes níveis de não-dominância formando diversas fronteiras. A cada geração de uma nova fronteira F_i, nível de não-dominância, as soluções selecionadas são removidas do conjunto de soluções P, o qual é uma cópia da população atual R_t a ser ordenada. Desse modo, o processo realiza a geração das fronteiras até que todas as soluções sejam classificadas em alguma fronteira.

Crowding Distance é um operador de diversidade que visa a garantir uma melhor distribuição das soluções. Essa métrica calcula, para cada fronteira F_i, a distância média que

Algoritmo 3.1: Pseudocódigo do NSGA-II

```
1  Entrada: N, g
2  Saída: Melhor conjunto de soluções não dominadas para o problema
3  Início
4      Inicializa a população P_0 de tamanho N;
5      Avalia-se cada indivíduo de P_0;
6      para t ← 0 até g faça
7          para i ← 0 até N/2 faça
8              se t < g então
9                  Seleção;
10                 Cruzamento;
11                 Mutação;
12                 Avalia-se os indivíduos criados;
13                 Guarda os indivíduos criados em Q_t;
14                 i = i + 2;
15             fim se
16         fim para
17         R_t ← P_t ∪ Q_t;
18         Aplica-se Fast Non-Dominated Sorting em R_t gerando vetores F_i de
             soluções não-dominadas;
19         Calcula-se o Crowding Distance Sorting de cada solução em F_i;
20         Popula P_{t+1} com as soluções dos melhores vetores F;
21         t + +
22     fim para
23     retorna Retorna as soluções não dominadas de P_t
24 Fim
```

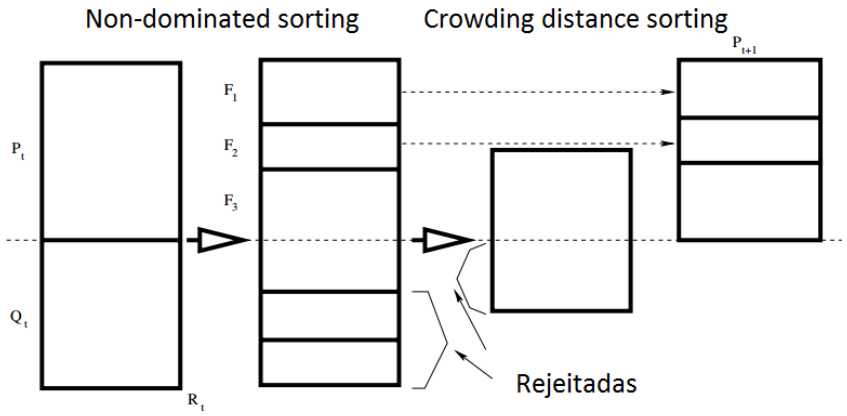


Figura 3.3: Funcionamento do NSGA-II (adaptada de [20])

uma solução i está de seus vizinhos $(i-1)$ e $(i+1)$, ilustrada na Figura 3.4. Posteriormente, as soluções são ordenadas de maneira decrescente e sendo privilegiado as soluções mais espalhadas no espaço de busca.

A ideia é que o operador de diversidade possa encontrar pontos extremos e priorizar os pontos mais distantes durante o processo de seleção a fim de espalhar os resultados ao longo da

Algoritmo 3.2: Pseudocódigo do *Fast Non-dominated Sorting*

```
1  Entrada: R_t
2  Saída: Vetores F_i de soluções não-dominadas
3  Início
4      P ← R_t;
5      F = ∅;
6      para i ← 0 até P = ∅ faça
7          F_i ← Soluções não-dominadas de P;
8          Remove de P as soluções presentes em F_i;
9          i + +;
10     fim para
11     retorna Vetores F_i de soluções não-dominadas
12 Fim
```

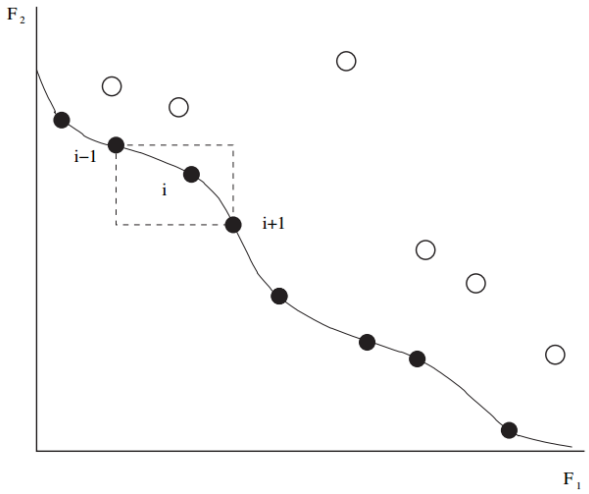


Figura 3.4: *Crowding Distance* (extraída de [20])

Fronteira de Pareto, garantindo um melhor *fitness* àquelas que estejam em áreas menos povoadas no espaço de busca.

3.1.2 *Strength Pareto Evolutionary Algorithm 2* (SPEA2)

O SPEA2 [131] (Algoritmo 3.3) é um aperfeiçoamento do seu antecessor SPEA. O SPEA utiliza um arquivo externo, também denominado de população externa, o qual mantém as soluções não-dominadas encontradas, e, em cada iteração, as soluções-não dominadas são repassadas à essa população externa que participa do procedimento de seleção. SPEA2 também conta com o cálculo do valor de força (*strength*) que é o valor de uma solução que se baseia na quantidade de soluções que dominam e que são dominadas por ela. SPEA2 preserva algumas características do seu antecessor, por exemplo, uso de um arquivo externo e o *strength* [20, 131]. Entretanto, três aspectos principais os diferenciam [131]:

- A estratégia de atribuição da função de *fitness* no SPEA2 considera, para cada indivíduo, tanto o número de soluções que o dominam quanto o número de soluções dominadas por ele;

- SPEA2 utiliza uma técnica de estimativa de densidade da vizinhança incorporada à função de *fitness* dos indivíduos. Assim, o processo de busca leva em consideração a diversidade da população;
- No SPEA2 o algoritmo de agrupamento foi substituído por um novo método de truncamento, que previne a exclusão de soluções localizadas nas extremidades.

Algoritmo 3.3: Pseudocódigo do *SPEA2* (adaptado de [20])

```
1  Entrada: N, g
2  Saída: Soluções de A
3  Início
4      Inicializa a população P_0 de tamanho N;
5      Cria um arquivo externo A_0 vazio;
6      para t ← 0 até g faça
7          Calcula o valor de fitness dos indivíduos em P_t e A_t;
8          Copia todos os indivíduos não-dominados de P_t + A_t para A_{t+1};
9          Usa o operador de truncamento para remover indivíduos de A_{t+1} quando a
             capacidade máxima do arquivo é extrapolada;
10         Se a capacidade máxima de A_{t+1} não for atingida, copia indivíduos
             dominados de P_t + A_t para A_{t+1};
11         Executa torneio binário para seleção de indivíduos para se reproduzirem;
12         Aplica recombinação e mutação para gerar filhos;
13     fim para
14     retorna Soluções de A
15 Fim
```

SPEA2, primeiramente, gera de forma aleatória uma população inicial P_0, de tamanho N, que é avaliada. Posteriormente é gerado um novo arquivo externo vazio. Dado um critério g, a cada iteração t é avaliado o valor de *fitness* da população atual e do arquivo externo. Posteriormente são unidos a população atual e o arquivo externo, removendo as soluções não-dominadas e executando o operador de truncamento, caso o tamanho do arquivo externo seja ultrapassado. Por fim, são aplicados os operadores de seleção, de recombinação e mutação para a geração de novos filhos.

3.2 Hiper-Heurísticas

Algoritmos multi-objetivos têm sido utilizados para resolver diferentes problemas nas mais variadas áreas de aplicação. Entretanto, não há um guia de como proceder na seleção e configuração desses algoritmos [14], e se todos os algoritmos de busca fossem avaliados em todos os problemas existentes com uma configuração adequada ao problema, todos teriam o mesmo desempenho médio [128]. Nesse contexto, surgiu o conceito de hiper-heurística para possibilitar uma abordagem mais flexível e uma metodologia que automatiza o *design* e o *tunning* [14] dos algoritmos.

Inicialmente hiper-heurísticas foram definidas como heurísticas que escolhem heurísticas [22], no entanto, outra definição, mais recente e mais empregada, as define como um conjunto de abordagens ou como uma metodologia de alto nível com o intuito de proporcionar um *design* ou

ajuste automático de métodos heurísticos de modo a resolver problemas computacionalmente difíceis [14]. Adicionalmente, há outra definição de hiper-heurística que considera níveis e trata hiper-heurísticas como sendo heurísticas de alto nível. Nesta definem-se três critérios [18]: (i) uma hiper-heurística gerencia um conjunto de heurísticas de baixo nível (*Low-Level Heuristic*, LLH); (ii) busca um bom método que resolva o problema, mais do que uma boa solução; (iii) utiliza apenas informações limitadas, específicas do problema, denominada barreira de domínio (Figura 3.5).

Figura 3.5: Barreira de domínio (adaptada de [118])

A barreira de domínio, ou barreira do conhecimento, é uma parte importante da hiper-heurística presente entre a hiper-heurística e as LLHs. A barreira de domínio é uma interface que abstrai o conhecimento sobre o problema (domínio) de toda a abordagem, restringindo a comunicação entre a hiper-heurística e a abordagem em relação às LLHs e a função de avaliação [22]. Essa barreira visa a permitir que a mesma estratégia de hiper-heurística possa ser usada para qualquer problema, somente mudando as LLHs implementadas. Na Figura 3.5 ainda é ilustrado o domínio do problema em que as LLHs são aplicadas, o qual contém informações sobre o problema como: a representação do problema, instâncias, funções de avaliação (função *fitness*), etc.

A principal diferença entre hiper-heurísticas e heurísticas convencionais é que elas abordam um conceito que visa a trabalhar no espaço de busca de heurísticas ao invés do espaço de busca de soluções, visando a constituir adequadas combinações de algoritmos (ou parte deles) com o intuito de obter um melhor desempenho [14].

3.2.1 Classificação

Uma hiper-heurística pode ser classificada em duas dimensões, em relação a sua natureza e a sua capacidade de aprendizado (ou *feedback*) [15] (Figura 3.6).

De acordo com a natureza do espaço de busca a hiper-herística pode ser classificada como heurística de seleção de heurísticas existentes, ou heurística de geração de heurísticas através de componentes pré-existentes. Um segundo nível na dimensão da natureza ainda pode ser segmentado para definir os paradigmas de busca do tipo construtiva e perturbativa [56], com relação às LLHs utilizadas pela hiper-heurística. Heurísticas de construção criam um componente da solução por vez, já as heurísticas de perturbação trabalham com uma solução completa e a cada ponto de decisão modificam a solução com o intuito de gerar uma nova solução (perturbar a solução, ou seja, iterativamente melhorá-la) [15].

A segunda dimensão relaciona-se com a forma de aprendizado realizado durante a aplicação da hiper-heurística. Se algum tipo de informação (*feedback*) é utilizado no processo de busca, a hiper-heurística pode ser considerada de aprendizado, caso não haja nenhum *feedback*, a hiper-heurística é considerada sem aprendizado. Assim, havendo o aprendizado a hiper-heurística pode ser classificada em: *on-line* ou *off-line*. Hiper-heurísticas *on-line* tentam aprender durante o processo de resolução de uma instância de um problema, e hiper-heurísticas *off-line* aplicam

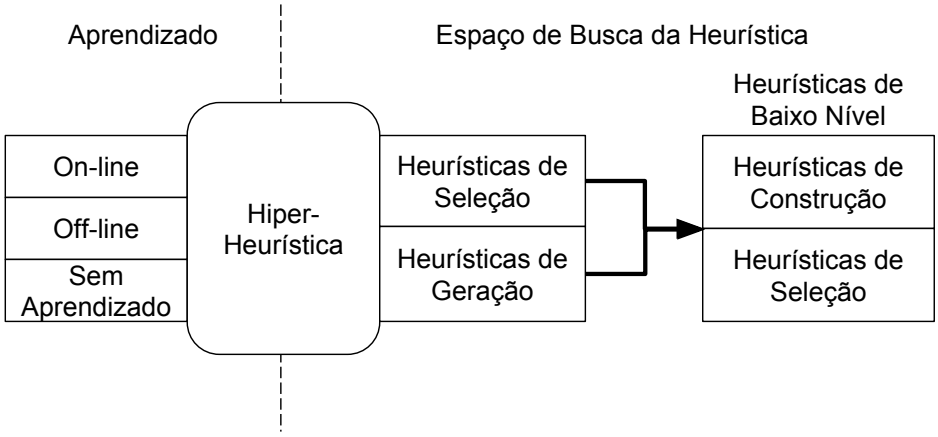


Figura 3.6: Classificação das Hiper-Heurísticas (adaptada de [118])

um método genérico para a resolução a partir de um conjunto de treinamentos prévios. Existem trabalhos utilizando metodologias híbridas, em que há a combinação de heurísticas construtivas com perturbativas [42], bem como de heurísticas de seleção com geração [67].

3.2.2 Métodos de Seleção

Uma hiper-heurística de seleção de heurísticas perturbativas geralmente possui um par de componentes, sendo: (i) o método de seleção, conhecido também como operador de seleção, o qual visa a selecionar uma LLH e (ii) o método de aceitação que aceita ou rejeita uma nova solução gerada [14]. Os métodos de seleção utilizam técnicas para a seleção de uma LLH em um dado momento da busca, podendo ser ou não com aprendizado. Assim, esses métodos utilizam otimização adaptativa dinâmica para aprender estratégias, através de uma máquina de aprendizado, sem a necessidade de supervisão [14, 49], decidindo quando uma determinada LLH deve ser aplicada durante o processo de busca.

Existem inúmeras técnicas de seleção com ou sem aprendizado (mais simples). As técnicas sem aprendizado selecionam, por exemplo, de modo aleatório ou exaustivo uma LLH. Dentre as principais técnicas de seleção com aprendizado há as baseadas em pontuação (*ranked based*), que atribuem ou atualizam uma pontuação a uma LLH, aplicada de acordo com regras de pontuação para o desempenho da LLH. Há também as técnicas que utilizam o ajuste de memória, o qual avalia uma série temporal (histórica) de aplicações das LLHs [14]. Quando são realizadas a atribuição ou atualização da pontuação do desempenho de uma LLH, procedimento chamado de *Credit Assigment*, uma estratégia deve ser aplicada para a seleção de uma LLH, tais como as estratégias: *max*, que seleciona uma LLH com pontuação máxima, e roleta, que utiliza probabilidades de seleção baseadas na pontuação [14].

Dentre os métodos de seleção destacam-se o método *Choice Function* (Função de Seleção, CF) [22], pela simplicidade e melhora nos resultados obtidos, e o *Multi-Armed Badit* (MAB) [37], pelos resultados promissores na área de hiper-heurística. Esses métodos lidam com o dilema da Exploração versus Intensificação (*Exploration vs Exploitation*, EvE). Exploração visa a explorar adequadamente o conjunto de operadores estabelecido de modo a dar chance a todos os operadores, uma vez que um operador não selecionado pode se tornar melhor em uma aplicação na busca no futuro [72]. Já a Intensificação visa a selecionar com uma maior frequência o operador que obtém bons resultados [72]. Os métodos CF e FRR-MAB são descritos a seguir.

Choice Function

O método *Choice Function* (CF) é do tipo *ranked based* para a seleção de uma LLH [22]. O CF visa a manter um escore de cada LLH, avaliando, de forma adaptativa, a qualidade de cada aplicação de LLH de modo a escolher a cada ponto de decisão aquela que possua o maior escore de todos para ser aplicada no processo. Para isso, o escore de uma LLH (h_i) é dado através de três componentes (Equação 3.1): o quão bom foi sua aplicação individualmente ($f_1(h_i)$); o quão bom foi sua aplicação quando utilizada em cooperação com outra LLH ($f_2(h_j, h_i)$); e quanto tempo se passou desde sua última aplicação ($f_3(h_i)$).

$$f(h_i) = \alpha f_1(h_i) + \beta f_2(h_j, h_i) + \delta f_3(h_i) \tag{3.1}$$

onde h_i é uma heurística de baixo nível que está sendo avaliada, f_1 é o desempenho individual de uma heurística de baixo nível, f_2 é o desempenho de pares de heurísticas de baixo nível (h_i e h_j); e f_3 é o tempo de CPU decorrido desde que h_i foi executada. No primeiro e segundo componentse priorizam-se as LLHs com melhor desempenho (intensificação). Por fim, no último componente, possibilita-se a diversidade (exploração) da aplicação dando chance de uma LLH ser aplicada novamente. Os parâmetros α, β e δ variam em um intervalo [0,1] e são pesos para as funções f_1, f_2 e f_3, respectivamente, permitindo balancear a busca entre os fatores de intensificação e exploração.

Posteriormente, houve uma simplificação do *Credit Assigment* do CF [79], descrito na Equação 3.2.

$$f(h) = \alpha f_1(h) + \beta f_2(h) \tag{3.2}$$

Esta versão contém apenas dois componentes f_1 e f_2, respectivamente. Analogamente à equação original, define-se que dada uma LLH h obtém-se seu escore $f(h)$ através da função $f_1(h)$, a qual reflete a melhoria recente de h, e da função $f_2(h)$, a qual reflete o tempo de CPU desde a última vez que h foi utilizada. Os parâmetros α e β são usados para equilibrar o dilema EvE.

Fitness-Rate-Rank based Multi-Armed Bandit

Fitness-Rate-Rank based Multi-Armed Bandit (FRR-MAB) trata-se de um algoritmo proposto por Fialho [72] para o problema da Seleção Adaptativa de Operadores (*Adaptive Operator Selector*, AOS). Esse algoritmo foi baseado no algoritmo *Upper Confidence Bound* (UCB) [5], voltado a solucionar o problema *Multi-Armed Bandit* (MAB).

AOS [37] é um paradigma de abordagem *on-line* capaz de, forma autônoma, selecionar entre diferentes operadores de variação, utilizando informações do histórico de evolução para decidir quando utilizar determinado operador para solucionar problemas de otimização em um processo de um EA [37, 90]. Assim, AOS está dentro da área de hiper-heurística como sendo uma técnica de abordagem *on-line* que utiliza ajuste de memória [14]. Na Figura 3.7 são ilustrados os dois principais componentes de um AOS: *Credit Assigment* e o Operador de Seleção.

O *Credit Assigment* trata de como atribuir uma recompensa (*reward*) a um operador baseado na qualidade de sua aplicação, e o Operador de Seleção irá decidir como selecionar um operador dentre diferentes operadores considerando os *rewards* recebidos recentemente (registro/histórico de crédito) [72, 90]. O *Credit Assigment* ainda pode ser dividido em duas partes: métrica de qualidade, a qual determina o quão bom foi a aplicação de um determinado operador, e atribuição de *reward*, a qual determina como utilizar a qualidade avaliada de modo a atualizar as informações de adequação do operador [72].

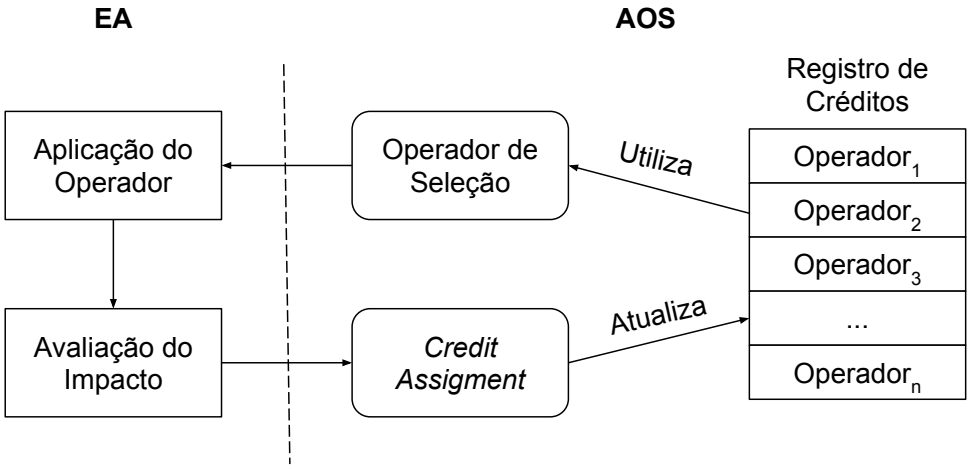


Figura 3.7: Esquema Geral da Seleção Adaptativa de Operadores (adaptada de [90])

EvE têm sido estudado pela comunidade estatística no problema de *Multi-Armed Bandit* (MAB) [5, 37, 70]. O problema MAB é o cenário no qual um jogador joga em um conjunto de máquinas caça-níquéis que mesmo idênticas produzem diferentes ganhos. Após jogar é recebida uma recompensa (*reward*) aleatória, assim, objetiva-se maximizar a soma das recompensas. Dentre os inúmeros algoritmos propostos para tratar do problema MAB, o algoritmo *Upper Confidence Bound* (UCB) [5] é o que se destaca por oferecer garantias de otimalidade assintótica no total de *rewards* acumulado [72], posteriormente apareceram outros algoritmos que tiveram o UCB como inspiração estendendo-o, como por exemplo, o *Fitness-Rate-Rank based Multi-Armed Bandit* (FRR-MAB) [72], que vem se destacando quanto seus resultados focando na tarefa do *Credit Assigment* [44, 72] e por isto será utilizado e descrito nesta dissertação.

O FRR-MAB visa a automaticamente selecionar um determinado operador a ser aplicado, através do aprendizado *on-line*, onde a qualidade de um operador é avaliada pela melhoria do *fitness* que esse realizou [72]. Para isso, é utilizado um mecanismo de decaimento (*decaying*) para calcular o valor do *reward* final de cada operador em relação às aplicações realizadas em uma determinada janela de tempo (*sliding window*, SW). Posteriormente, o próximo operador a ser selecionado é baseado nesses valores de *reward* [72]. Desse modo, o FRR-MAB seleciona o melhor operador de acordo com a Equação 3.3.

$$op_t = \underset{i=\{1\ldots K\}}{\operatorname{argmax}} \left(FRR_{i,t} + C \times \sqrt{\frac{2 \times ln \sum_{j=1}^{K} n_{j,t}}{n_{i,t}}} \right) \tag{3.3}$$

onde o objetivo é selecionar o melhor operador (op) através da estratégia *max*. Para cada ponto de tempo t dentro de um conjunto K de operadores, que possua um valor empiricamente estimado ($FRR_{i,t}$) em um intervalo que depende do número de vezes ($n_{i,t}$) que esse foi aplicado anteriormente. Sendo utilizado um parâmetro de fator de escala (C) para controlar o *trade-off* entre Exploração (o termo presente na raiz quadrada) e Intensificação (o primeiro termo), originalmente proposto por Fialho et *al.* [38].

O método FRR-MAB possui dois procedimentos particulares, sendo o primeiro o *Credit Assigment* (Algoritmo 3.4) e o segundo consiste na Seleção de Operador [72].

O primeiro passo no procedimento do *Credit Assigment* é realizar o cálculo da qualidade relacionada à aplicação de um determinado operador de modo a mensurar seu impacto na busca,

Algoritmo 3.4: Procedimento do *Credit Assigment* (adaptado de [72])

Início
  para cada $i \in K$ **faça**
    $Reward_i = 0.0$;
    $n_i = 0$;
  fim para
  para cada $elemento \in SlidingWindow$ **faça**
    i = elemento.*GetOperador()*;
    FIR = elemento.*GetFIR()*;
    $Reward_i = Reward_i + FIR$;
    $n_i + +$;
  fim para
  Ranqueamento dos $rewards$ em ordem decrescente ($Reward_i$);
  $Rank_i$ = valor de *ranking* do $Reward_i$;
  para cada $i \in K$ **faça**
    $Decay_i = D^{Rank_i} \times Reward_i$;
  fim para
  $DecaySum = \sum_{i=1}^{K} Decay_i$;
  para cada $i \in K$ **faça**
    $FRR_i = \dfrac{Decay_i}{DecaySum}$;
  fim para
Fim

para isso, utiliza-se o método de taxa de melhoria de *fitness* (*Fitness Improvement Rate*, FIR) [72] identificado através da Equação 3.4.

$$FIR_{i,t} = \frac{pf_{i,t} - cf_{i,t}}{pf_{i,t}} \tag{3.4}$$

O FIR é empregado de modo a não utilizar os valores brutos das melhorias de *fitness* causadas pelo uso de determinado operador, pois isso varia de problema para problema e diferentes fases de um problema de otimização. Assim, o uso direto desse valor poderia deteriorar a eficiência do algoritmo [37, 72]. Desse modo, através da Equação 3.4 é calculado o FIR, alcançado por um operador i em um ponto de tempo t, onde $pf_{i,t}$ representa o valor de *fitness* das soluções pais e $cf_{i,t}$ o valor de *fitness* das soluções filhas [72]. Posteriormente, os FIRs dos operadores recentemente utilizados são armazenados em uma *sliding window* (SW) de tamanho W, a qual é organizada como uma estrutura de dados FIFO, onde os valores mais recentes são adicionados ao final enquanto os mais antigos registros são removidos do início de modo a manter um tamanho constante. A Figura 3.8 mostra como se dá o armazenamento do FIR relacionado ao seu respectivo operador.

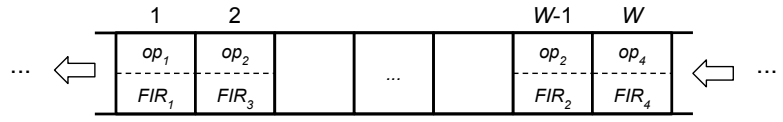


Figura 3.8: Ilustração do procedimento FIFO na *Sliding Window* (adaptada de [72])

Através do uso da SW é possível avaliar um operador sem que ele seja prejudicado pelo seu desempenho em uma fase muito precoce, o qual pode ser irrelevante para o seu desempenho atual. Dessa forma, garante-se que a informação da FIR na SW refere-se a uma situação atual da pesquisa [72]. Posteriormente, é calculado o $Reward_i$ de um operador i, o qual é a soma de todos os valores de FIR para o operador i na SW. Em seguida, é determinado um ranqueamento, de modo decrescente, de todos os $rewards$ dos operadores presentes na SW. Assim, define-se um $Rank_i$ que representa o valor de *ranking* do operador i, o qual prioriza os melhores operadores. Depois, um fator de decaimento $D \in [0, 1]$ é utilizado com o intuito de transformar o *Reward* inicial de acordo com a posição relativa em relação ao *reward* dos outros operadores ($Decay_i$), sendo que quanto menor o valor de D mais esse influencia para o melhor operador. Por fim, os valores de decaimento dos *rewards* são normalizados resultando no *fitness-rate-rank* (FRR) de cada operador i. Esses valores (FRR_i) são então utilizados pelo procedimento de seleção de operador.

O segundo procedimento (Algoritmo 3.5), seleção de operadores, seleciona novos operadores visando a gerar novas soluções. Esse procedimento assemelha-se ao do algoritmo UCB original [5], sendo a principal diferença o uso dos valores de FRR com índice de qualidade ao invés da média de todas as recompensas recebidas por um determinado operador [72]. Sendo que, n_i indica a quantidade de vezes que um operador i foi utilizado em suas W recentes aplicações na SW. Além disso, o FRR-MAB começa a atuar somente quando todos os operadores tenham sido previamente utilizados ao menos uma vez na pesquisa, dando assim chance a todos de serem igualmente selecionados [72].

Algoritmo 3.5: Procedimento para a seleção de um operador baseado em FRR-MAB (adaptado de [72])

Início
  se *todos os operador ainda não foram selecionados* **então**
    op_t = um operador do conjunto K de operadores selecionado de forma aleatória;
  senão
    $$op_t = \underset{i=\{1\ldots K\}}{\operatorname{argmax}}\left(FRR_{i,t} + C \times \sqrt{\frac{2 \times ln \sum_{j=1}^{K} n_{j,t}}{n_{i,t}}}\right);$$
  fim se
Fim

3.3 Considerações Finais

A utilização de algoritmos de otimização auxilia na busca por soluções adequadas para resolver problemas nas mais variadas áreas de aplicação. Dentre os algoritmos de busca destacam-se os algoritmos multi-objetivos devido à característica multi-objetivo dos problemas do mundo real. Todavia, muitas vezes o uso de uma abordagem baseada em busca demanda um alto conhecimento para realizar adaptações nos detalhes de implementação e configurações desses algoritmos. Para lidar com esse problema o uso do conceito de hiper-heurística surge como uma abordagem mais flexível, a qual possibilita abstrair os detalhes de implementação e configuração em relação ao problema, e pode ser aplicada em diferentes domínios.

Devido às vantagens em relação à hiper-heurística, o presente trabalho visa à utilização desse conceito através de uma hiper-heurística de seleção com aprendizado *on-line* para auxiliar na aplicação do teste de Mutação de Ordem Superior. Para isso, são utilizados os algoritmos multi-objetivos NSGA-II [24] e SPEA2 [131]. Ambos algoritmos têm sido amplamente empregados para resolver diferentes problemas em SBSE [53]. Além disso, são utilizados três métodos de seleção de LLHs: *Choice Function* (CF), adaptado por Maashi et *al.* [79], o *Fitness-Rate-Rank based Multi-Armed Bandit* (FRR-MAB) [72] e método de seleção aleatório. Os dois primeiros foram utilizados com sucesso na literatura [48, 79] e o método aleatório é utilizado para comparação.

No próximo capítulo são apresentados os principais trabalhos relacionados que utilizam algoritmos de otimização para geração de HOMs e que utilizam hiper-heurística para resolver problemas da Engenharia de Software.

Capítulo 4

Trabalhos Relacionados

Na literatura, estudos fomentados pelo problema de geração de HOMs propuseram estratégias relacionadas à forma de criação [71, 110] e classificação [61, 95, 96, 101] desses mutantes, assim como à redução da quantidade de mutantes equivalentes [65, 105] e à utilização de técnicas de SBSE (*Search-Based Software Engineering*) que possibilitam aprimorar os procedimentos de geração e seleção desses mutantes.

Com o intuito de melhor resolver os problemas de otimização observa-se um crescente interesse da comunidade de SBSE na utilização de hiper-heurísticas, sendo que muitos autores apontam o uso de hiper-heurísticas como uma tendência e tema de pesquisa futura na área de SBSE [3, 49, 98]. Contudo, são ainda poucos os trabalho que aplicam hiper-heurística em SBSE [8, 17, 35, 36, 48, 60, 69, 84, 121] e não há trabalhos que abordem a utilização de hiper-heurísticas no contexto de HOMs.

Desse modo, este capítulo descreve os principais trabalhos relacionados à essa dissertação, abordando seus dois temas principais: (i) geração e criação de HOMs e (ii) uso de hiper-heurística na área de SBSE.

4.1 Estratégias para geração de HOMs

Diante dos esforços dos pesquisadores em reduzir os custos relacionados à geração de mutantes, execução e análise de resultados, Polo et *al.* [110] propõem estratégias para geração de mutantes de segunda ordem (*Second Order Mutants*, SOMs) com o intuito de reduzir a quantidade de mutantes sem diminuir o escore de mutação.

Nesse estudo, foram introduzidas as seguintes estratégias: *LastToFirst*, *DifferentOperators* e *RandomMix*. Primeiramente, os autores tratam os FOMs como uma lista ordenada de acordo com a ordem em que cada FOM foi gerado, posteriormente aplicam nessa lista as estratégias propostas. Desse modo, a estratégia *LastToFirst* combina os mutantes, da lista de FOMs, o primeiro com o último, o segundo com o penúltimo e assim por diante. A estratégia *DifferentOperators* combina mutantes provenientes de diferentes operadores, ou seja, um mutante proveniente de um operador O_1 não será combinado com outro mutante gerado pelo mesmo operador O_1. A estratégia *RandomMix* combina aleatoriamente dois mutantes quaisquer.

Diante das estratégias propostas os autores obtiveram uma redução em aproximadamente 50% do tamanho do conjunto de mutantes em relação aos FOMs sem diminuir o escore de mutação com as estratégias *LastToFirst* e *RandomMix*, enquanto *DifferentOperators* reduziu em média 40% o número de mutantes. Em relação a redução no número de mutantes equivalentes houve uma redução dos SOMs em relação aos FOMs, passando de 18,66% para 5% em média para todas as estratégia propostas.

Após os bons resultados obtidos pelas estratégias propostas por Polo et *al.* [110], Papadakis e Malevris [105] propõem cinco novas estratégias baseadas nas estratégias propostas por Polo et *al.* [110]. São elas: *First2Last*, *SameNode*, *SameUnit*, *SU_F2Last* e *SU_DiffOp*.

A estratégia *First2Last* ordena os FOMs em relação à ordem em que o trecho de código que foi alterado aparece no código fonte, posteriormente seleciona e combina o primeiro mutante com o último, o segundo mutante com o penúltimo e assim por diante. *SameNode* seleciona os mutantes a partir do mesmo bloco base. *SameUnit* seleciona os mutantes a partir da mesma unidade do programa. *Same Unit FirstToLast*, denominada *SU_F2Last*, e *Same Unit Different Operators*, denominada *SU_DiffOp*, selecionam os mutantes candidatos, baseadas na aplicação das estratégias *First2ToLast* e *DifferentOperators*, aplicando-as individualmente para cada unidade do programa.

O objetivo do trabalho é comparar os benefícios das estratégias de FOMs e SOMs. Para isso, os autores avaliaram em relação a uma perspectiva do problema com mutantes equivalentes e realizaram uma comparação medindo os fatores de custo do processo de teste como: número de mutantes produzidos, número de mutantes equivalentes, o número de casos de teste necessários e o número de defeitos revelados por cada estratégia.

Para determinar a comparação do melhor custo benefício os autores utilizaram duas métricas, a eficácia do teste, que calcula o número de casos de teste em relação ao número de defeitos revelados, e o custo da eficácia, que calcula o número de casos de teste mais o número de mutantes equivalentes em relação ao número de defeitos revelados. Desse modo, os autores constataram que a estratégia com a melhor eficácia do teste foi a estratégia *SU_F2Last* e a pior foi a *SameNode*. Em relação ao custo da eficácia, a melhor estratégia foi a *RandomMix*, estratégia de Polo et *al.* [110], e a pior, foi a *SameNode*.

Além disso, as estratégias dos FOMs foram mais efetivas em revelar defeitos do que as estratégias dos SOMs, contudo, SOMs são mais eficientes que os FOMs. SOMs foram menos custosos, em relação ao número de mutantes produzidos e em relação ao número de casos de teste necessários exigidos. As estratégias adotadas nesse estudo, para gerar SOMs, possibilitaram reduzir os mutantes equivalentes entre 80% e 90%.

Assim como Papadakis e Malevris [105], Mateo et *al.* [89] estenderam o trabalho de Polo et *al.* [110]. Mateo et *al.* [89] focaram no teste funcional no nível de sistema com SOMs, buscando erros nas suas funcionalidades. Nesse estudo, foram utilizadas as estratégias propostas por Polo et *al.* [110] e a estratégia *Each-Choice* que foi adaptada de Ammann e Offut [2]. Assim como Polo et *al.* [110], assume-se a existência de uma lista de FOMs. Dessa maneira, a estratégia *Each-Choice* seleciona sequencialmente os mutantes da lista de FOMs e combina-os para formar um novo mutante de ordem superior. Além disso, nesse estudo, a estratégia *LastToFirst* de Polo et *al.* [110] foi renomeada para *FirstToLast*, porém manteve seu procedimento original.

Na primeira parte do estudo foram gerados os mutantes através das estratégias. As estratégias conseguiram reduzir em 50% o número de mutantes, dependendo da estratégia, em relação ao número de FOMs, porém não foram identificados mutantes equivalentes. Dentre as estratégias utilizadas, a estratégia *FirstToLast* se destacou por reduzir o maior número de mutantes, seguida das estratégias *RandomMix* e *Each-Choice*. O escore de mutação das estratégias dos FOMs alcançaram, em geral, aproximadamente 80%, enquanto a análise de mutantes com SOMs alcançaram 95%.

Posteriormente, na segunda parte do estudo, foi gerada uma matriz informando por qual caso de teste cada mutante era morto. Em seguida, foram definidos três algoritmos baseados no algoritmo de Busca Gulosa: *Max*, seleciona *t* casos de teste que matam o maior número de mutantes; *Min*, *t* casos de teste que matam o menor número de mutantes e; *Rdm*, seleciona

aleatoriamente *t* casos de teste. Esses algoritmos foram aplicados para reduzir o número de casos de teste.

Os algoritmos conseguiram reduzir o número de casos de teste necessários, entretanto, houve a diminuição do escore de mutação. Dentre os algoritmos propostos o algoritmo *Min* obteve pouca diferença com relação à diminuição do escore de mutação, aproximadamente 2,5%, porém selecionou mais casos de teste do que o algoritmo *Max*. O algoritmo *Rdm* ficou entre os algoritmos *Min* e *Max*.

Os autores concluíram que SOMs reduzem significativamente os custos sem diminuir muito o escore de mutação. Além disso, os autores recomendam a utilização de HOMs para testes de grandes aplicações, devido ao custo reduzido, e para pequenas ou críticas aplicações o teste com FOMs, pois o teste com HOMs está associado a uma diminuição no escore de mutação.

4.2 Geração de HOMs baseada em busca

Embora possam ser encontrados estudos em relação à geração, execução e análise de mutantes, esses estudos não abordam uma classificação de mutantes de ordem superior. Assim, Jia e Harman [61] propuseram uma classificação para esses mutantes (detalhada na Seção 2.4.1). Ainda nesse trabalho, Jia e Harman [61] realizaram um estudo empírico com os algoritmos de otimização baseados em busca. Os algoritmos escolhidos foram o algoritmos de Busca Gulosa (*Greedy*), Genético e Subida de Encosta (*Hill Climbing*).

O objetivo do trabalho era, além de classificar os HOMs, encontrar *Strongly Subsuming* HOMs (SSHOMs), pois os SSHOMs podem ser utilizados para aumentar a qualidade do conjunto de testes e por isso são considerados valiosos, porém muito raros de se obter [61]. Os SSHOMs podem ainda ser utilizados também como uma técnica de redução de custo, através da substituição dos FOMs por um único HOM, pois os casos de teste que matam o HOM também matam seus FOMs constituintes. Desse modo, visando a encontrar mutantes capazes de gerarem defeitos sutis que são aqueles defeitos que necessitam de casos de teste mais elaborados para serem revelados [61].

A representação de um HOM foi realizada através de um vetor de inteiros. Cada elemento do vetor representa um operador de mutação aplicado, enquanto os índices indicam a posição em que o operador de mutação foi aplicado. Como métrica de *fitness* de um HOM foi avaliada a sua facilidade de ser morto em relação aos seus FOMs constituintes, variando seu valor entre 0 e 1. Quando o valor é igual a 0 indica que não há casos de teste que matam o mutante, consequentemente um potencial mutante equivalente. Quando o valor é igual a 1 indica que é um mutante que pode ser morto por qualquer caso de teste.

Assim, nesse trabalho constatou-se que há SSHOMs em cada programa estudado, sendo que de todos os HOMs encontrados aproximadamente 15% era do tipo SSHOM. Dentre os algoritmos estudados, o algoritmo genético foi o mais eficiente para encontrar os mutantes do tipo SSHOM. O algoritmo *Hill Climbing* encontrou os HOMs com maiores valores de *fitness*, enquanto a Busca Gulosa encontrou os HOMs de maiores ordens.

Omar et *al.* [101] estendeu o trabalho de Jia e Harman [61] para Java e AspectJ, sendo introduzida a notação de *subtle* HOMs. Essa notação busca a aprofundar a análise da classificação dos HOMs propostos por Jia e Harman [61], em relação aos mutantes classificados por eles como Fracamente *Subsuming* e Desacoplado, e Não *Subsuming* e Desacoplado. De acordo com Omar et *al.* [101], *subtle* HOMs são HOMs que não são mortos pelos mesmos casos de teste que matam os seus FOMs constituintes, representando assim, um novo defeito que não foi testado e servindo como base para gerar novos dados de teste para melhorar a eficácia do conjunto de

casos de teste. Desse modo, o estudo utilizou duas técnicas de otimização baseada em busca para encontrar *subtle* HOMs, Algoritmo Genético e Busca Local, além da Busca Aleatória.

O estudo utilizou como função objetivo uma abordagem de detecção de *subtle* HOMs, e HOMs que pudessem se tornar *subtle* HOMs caso um FOM fosse adicionado ou removido. Na otimização, um HOM foi representado através de um *array* dimensional de *strings*. Cada elemento no *array* representa uma linha de código do programa em teste. O *fitness* de um HOM calcula a diferença entre o HOM e seus FOMs constituintes em relação à diferença entre os casos de teste que matam o HOM e à união dos casos de teste que matam individualmente os FOMs constituintes. O valor da função varia entre 0 e 2 e, através dos valores da função objetivo, nesse estudo, os HOMs foram classificados em:

- Inteiramente Acoplado: quando o valor é igual a 0 (pior valor) e representa os casos onde não há diferença entre o conjunto de casos de teste que matam o HOM e a união dos casos de teste que matam individualmente seus FOMs constituintes, ou seja, os casos de teste que matam o HOM matam os FOMs;

- Parcialmente Acoplado: quando o valor está entre 0 e 1 e representa os casos onde há diferença entre os casos de teste que matam o HOM e a união dos casos de teste que matam individualmente seus FOMs constituintes, ou seja, alguns casos de teste que matam os FOMs matam o HOM;

- Desacoplado: quando o valor é igual a 1 e representa os HOMs que são mortos por um conjunto de casos de teste totalmente diferente da união dos casos de teste que matam seus FOMs constituintes;

- *Subtle*: quando o valor é igual a 2 (melhor valor) e representam um novo defeito que não pode ser revelado por nenhum caso de teste no conjunto de casos de teste fornecido T.

Desse modo, os autores realizaram a análise dos resultados que evidenciaram uma maior geração de mutantes equivalentes pelos algoritmos de Busca Local e o Algoritmo Genético em relação a Busca Aleatória, pois, de acordo com os autores, os mecanismos de seleção favoreceram os HOMs com maiores valores de *fitness*, igual a 2, e esses não eram mortos por nenhum caso de teste, tratando-se assim de casos como *subtle* HOMs. Em relação ao número de *subtle* HOMs, a Busca Local encontrou mais mutantes na maioria dos programas estudados e o Algoritmo Genético encontrou os *subtle* HOMs de maior ordem.

De acordo com os autores, através desse estudo foi demonstrada a capacidade da utilização de técnicas baseadas em busca em encontrar *subtle* HOMs, sendo os HOMs de segunda e terceira ordens os mais fáceis de serem encontrados. O procedimento adotado nesse trabalho foi precursor da ferramenta HOMAJ [103].

Omar et *al.* [102] estenderam o trabalho anterior [101] e introduziram três novas técnicas de busca para encontrar *subtle* HOMs; Busca Local Guiada (BLG), Busca Aleatória Restrita (BAR) e Busca Enumerada Restrita (BER). BLG utiliza uma heurística para focar na combinação de FOMs que sejam mais prováveis de produzir *subtle* HOMs. BAR utiliza uma configuração de parâmetros para limitar a busca para uma região do espaço de busca onde *subtle* HOMs são mais prováveis de serem encontrados. BER enumera todos os HOMs candidatos em uma sequência pré-definida, começando na região do espaço de busca onde HOMs são esperados de serem encontrados. Todos os FOMs equivalentes foram identificados manualmente e não foram considerados no processo de busca. Os algoritmos tiveram como critério de parada a criação de 50.000 HOMs.

As técnicas de BLG, Busca Local e BER obtiveram melhores resultados em encontrar *subtle* HOMs. BER obteve melhores resultados em encontrar um maior número de *subtle* SOMs. A técnica BAR obteve melhores resultados quando comparados com a Busca Aleatória, o que sugere que limitar o espaço de busca para o espaço de HOMs de baixa ordens provê melhores resultados devido a produzirem um maior número de *subtle* HOMs. Além disso, os resultados mostraram que as linguagens de programação avaliadas (Java e AspectJ) não afetaram as técnicas de busca.

Posteriormente Omar et *al.* [104] estenderam os trabalhos anteriores [101, 102], adicionando o objetivo relacionado com a dificuldade em matar um HOM e mantendo o objetivo de encontrar *subtle* HOMs. Para isso utilizaram seis técnicas: Algoritmo Genético, Busca Local, Busca Local Guiada por Interação de Dados (BLGD), Busca Local Guiada por Casos de Teste (BLGC), Busca Aleatória Restrita e Busca Enumerada Restrita. BLGD gera um conjunto menor de HOMs na vizinhança do que a Busca Local, visando evitar a criação e avaliação de HOMs que são inteiramente acoplados aos seus FOMs. BLGC é similar ao BLGD, porém essa técnica explora apenas HOMs vizinhos que possuem FOMs constituintes que podem ser mortos por casos de teste similares. Isto é, onde exista ao menos um par de FOMs que possam ser mortos por um casos de teste em comum. As técnicas de Busca Local, BLGD e BLGC foram mais eficientes do que as outras técnicas em encontrar *subtle* HOMs.

Landgon et *al.* [71] propuseram uma técnica diferente da empregada em trabalhos anteriores ao utilizar Programação Genética (*Genetic Programming*, GP) junto com o NSGA-II para explorar o espaço de HOMs que fossem realísticos e difíceis de serem mortos. Para isso, no algoritmo de GP foram utilizadas duas funções objetivo: diferença semântica, relacionada à quantidade de casos de teste que se comportam de maneira diferente, e diferença sintática, relacionada ao número de mudanças em relação ao programa em teste.

O algoritmo de GP ficou encarregado de realizar a criação e atribuição dos valores de *fitness* dos HOMs e o NSGA-II realizou a seleção dos mutantes através de duas funções objetivo: minimizar o número de testes necessários e minimizar a diferença sintática entre o mutante e o programa em teste. Após a seleção os mutantes selecionados são repassados novamente ao algoritmo de GP para a próxima geração. Cada mutante foi representado por instruções gramaticais para serem utilizados pelo GP.

O objetivo principal desse estudo foi investigar o relacionamento entre a diferença sintática e semântica. Nos experimentos realizados foram encontrados HOMs que eram mais difíceis de matar comparados com os FOMs.

Harman et *al.* [51] introduziram uma abordagem híbrida denominada SHOM, baseada em *Dynamic Simbolic Execution* (DSE) e *Search Based Software Testing* (SBST). Essa abordagem busca efetivamente matar FOMs e HOMs visando o preenchimento de suficiência dos mutantes através de testes baseados em busca. Para isso, a abordagem realiza a geração de dados de teste procurando a adequação da mutação forte e a capacidade de matar os FOMs e HOMs através da identificação de um ponto de mutação. Em seguida, utiliza a DSE para gerar dados de teste que permitam matar o mutantes. Posteriormente, utiliza o SBST para procurar caminhos do ponto de infecção até alguma saída para propagá-la. O algoritmo *Hill-Climbing* é utilizado na busca por dados de entrada que propaguem a infecção e para isso o algoritmo utiliza como função de *fitness* a interrupção máxima do fluxo de controle para aumentar a probabilidade de forte adequação.

Os resultados sugerem que ao utilizar essa abordagem pode-se obter uma melhoria média de aproximadamente 15% e 16% nos escore de mutação de primeira e segunda ordem. Além disso, foi possível gerar dados de teste capazes de matar tanto FOMs quando HOMs. Ao analisar os resultados utilizando o modelo RIP (*Reachability*, *Infection*, *Propagation*), os resultados apontam que a SHOM foi capaz de matar até 38% dos FOMs que estavam vivos utilizando

alcançabilidade e suficiência, 36% dos mutantas vivos utilizando apenas alcançabilidade. Em relação ao SOMs, 48% dos que estavam vivos foram mortos com alcançabilidade e necessidade, por sua vez 41% foram mortos apenas utilizando alcançabilidade.

SSHOMs são mutantes valiosos, permitem substituir seus FOMs constituintes, e são raros de se encontrar na análise de mutantes com HOMs, por isso, Harman et *al.* [52] propuseram um estudo voltado a encontrar esses mutantes. Os autores estudaram a redução do número de mutantes e o número de casos testes como resultado da substituição dos FOMs pelos SSHOMs.

Nesse estudo, foram utilizados os algoritmos de Busca Gulosa e Genético. A Busca Gulosa teve como objetivo remover mutantes de um mutante SSHOM, sem que a remoção de um mutante prejudicasse a qualidade do SSHOM. A representação de um HOM foi definida como sendo um vetor de pares de inteiros. O comprimento de vetor representa o número de FOMs que constituem o HOM, sendo cada índice do elemento do vetor a localização de um FOM e o tipo de mutação aplicada. A função objetivo visou a reduzir o número de casos de teste que matam todos os FOMs e ao mesmo tempo aumentar o número de casos de teste que matam o HOM.

Os autores obtiveram uma redução no número de mutantes necessários, substituindo os FOMs pelos SSHOMs, em aproximadamente 45% e a eficiência do teste (número de casos de teste necessários) pôde ser melhorada em 14%, enquanto a eficácia do teste foi simultaneamente melhorada entre 5,6% e 12% (número de casos de teste que matam tanto FOMs quanto os HOMs).

Nguyen e Madeyski [95] introduziram uma nova classificação estendida de HOMs baseada na combinação dos conjuntos de casos de teste que matam o HOM e os conjuntos de casos de teste que matam os FOMs constituintes. Além disso, utilizaram o algoritmo NSGA-II para gerar e buscar HOMs. A abordagem mostrou-se adequada em encontrar SSHOMs, entretanto foram encontrados muitos HOMs equivalentes. Os autores definiram como objetivos: (i) minimizar o número de casos de teste que matam o HOM e também matam todos seus FOMs constituintes; (ii) minimizar o número de casos de teste que matam o HOM, mas não matam nenhum dos seus FOMs constituintes e (iii) minimizar o número de casos de teste que matam o HOM e podem matar algum de seus FOMs constituintes. Posteriormente esse trabalho foi estendido [96] com a aplicação do εMOEA [23, 66], εNSGA-II [1, 66] e NSGA-III, entretanto, o εNSGA-II apresentou os melhores resultados.

4.3 Hiper-Heurísticas aplicadas à Engenharia de *Software*

Basgalupp et *al.* [8] aplicaram uma hiper-heurística *off-line* visando à geração de algoritmos que criam árvores de decisão, utilizadas na predição de esforço de *software*. Esse estudo obteve resultados que identificaram que os algoritmos gerados pela hiper-heurística foram melhores que os melhores resultados obtidos por alguns algoritmos do estado da arte ou outras heurísticas tradicionais.

Carvalho et *al.* [17] propuseram uma hiper-heurística denominada MOCAITO-HH, utilizando o método de seleção *Choice-Function* através de uma abordagem *on-line*. Esse estudo abordou a seleção de algoritmos meta-heurísticos para estabelecer sequência de módulos para o teste de integração visando a minimização do custo em relação a construção de *stubs*. O estudo utilizou os algoritmos evolutivos NSGA-II, SPEA2 e IBEA, sendo que o desempenho de cada algoritmo foi realizado de acordo com os indicadores de *Hypervolume*, *Spread*, *Algorithm Effort* (AE) e *Ratio of Nondominated Individuals* (RNI).

Os resultados obtidos com a hiper-heurística apresentaram equivalência estatística com a execução do melhor algoritmo em cada experimento, porém há a vantagem de não ser necessária

a realização de configuração e avaliação para determinar o melhor algoritmo, possibilitando assim um menor esforço do testador.

Kumari et *al.* [69] utilizaram uma hiper-heurística para solucionar o problema de clusterização de módulos, selecionando heurísticas de baixo nível (*Low-Level Heuristics*, LLH), compostas por operadores de seleção, recombinação e mutação, enquanto a otimização está sendo executada. Os resultados obtidos mostraram um melhor desempenho da hiper-heurística em relação a um algoritmo evolutivo convencional para todos os problemas investigados.

Jia e Harman [60] aplicaram uma hiper-heurística de aprendizado *on-line* para aprender e aplicar estratégias de teste combinatorial. Os resultados foram analisados e comparados com resultados de técnicas do estado da arte e com os melhores resultados obtidos na literatura. A hiper-heurística obteve um bom desempenho em vários casos.

Guizzo et *al.* [48] propuseram a hiper-heurística denominada HITO (*Hyper-heuristic for the Integration and Test Order Problem*) que utiliza o NSGA-II para a resolução do problema ITO (*Integration and Test Order*) afim de reduzir o custo de *stubbing*. O problema ITO consiste em encontrar uma sequência de unidades (a menor parte de um *software*, por exemplo, classes, métodos e procedimentos) a serem testadas de modo que o custo de *stubbing* seja minimizado. Um *stub* é uma emulação de uma unidade que posteriormente será descartada quando a unidade for desenvolvida. O custo de *stubbing* é o gasto de recursos no desenvolvimento de *stubs*.

As heurísticas de baixo nível aplicadas junto com o algoritmo NSGA-II eram compostas de um par de operadores de recombinação e mutação. Os métodos de seleção aplicados foram *Choice-Function* e o *Sliding Multi-Armed Bandit*. Os resultados obtidos foram melhores que os obtidos pelo NSGA-II convencional.

O principal objetivo da HITO é a seleção e aplicação de uma LLH durante o processo de reprodução, baseando-se em seus desempenhos anteriores, além de possuir uma arquitetura voltada a uma forma genérica de uso. A HITO utiliza o tempo desde a última aplicação de uma LLH e o conceito de dominância para avaliar a qualidade de uma LLH. O tempo de aplicação é o tempo t decorrido desde que uma LLH foi aplicada pela última vez, sendo representado por um valor que é incrementado a cada nova seleção de LLH e zerado esse valor quando selecionada determinada LLH. O valor de t é compreendido em um intervalo $[0...+\infty[$. A dominância é avaliada através da dominância $\prec$ dos filhos em relação aos pais e para determinar o valor da dominância é utilizada a Equação 4.1 [48].

$$r_h = \frac{1}{\mid P \mid . \mid C \mid} * \sum_{p \in P} \sum_{c \in C} \begin{cases} 1 & \text{se } c \prec p \\ 0 & \text{se } p \prec c \\ 0.5 & \text{caso contrário} \end{cases} \tag{4.1}$$

O valor de dominância r para uma LLH h é dado pela avaliação dos pais P, onde p é um pai em P, e dos filhos C, onde c é um filho em C. Realiza-se a soma dos valores das comparações através de uma comparação para cada pai e filho. Um filho domina um pai ($c \prec p$) quando os valores da função objetivo do filho são iguais ou superiores aos do pai. Se um pai domina um filho ($p \prec c$) o filho é pior que o pai. Quando não há dominância entre o pai e filho, ambos são igualmente bons. Por fim, o valor da soma da dominância é multiplicado pela divisão de 1 em relação a multiplicação do tamanho de P e C.

Para resolver este mesmo problema, Mariani et *al.* [84] propuseram uma hiper-heurística baseada em Evolução Gramatical denominada GEMOITO (*Grammatical Evolution hyper-heuristic for the Multi-objectice Integration and Test Order problem*) que gera algoritmos multi-objetivos. A GEMOITO foi comparada com os algoritmos NSGA-II e SPEA2, além da hiper-heurística HITO (descrita anteriormente). Os resultados mostraram que GEMOITO

pode gerar algoritmos multi-objetivos que são estatisticamente melhores ou equivalentes aos algoritmos comparados.

Utilizando as mesmas ideias da HITO, Strickler et *al.* [121] aplicaram o conceito de hiper-heurística *on-line* para derivar conjuntos de produtos para o teste do Modelo de Características (*Feature Model*, FM). Os métodos de seleção utilizados foram o FRR-MAB (*Fitness Rate Rank based Multi-Armed Bandit*), devido aos bons resultados obtidos em outros trabalhos, e a seleção aleatória. Nesse trabalho foram selecionados três algoritmos multi-objetivos: NSGA-II, SPEA2 e IBEA, os quais foram avaliados considerando três objetivos: número de produtos selecionados, escore de mutação e cobertura *pairwise*. Dentre os algoritmos multi-objetivos avaliados o NSGA-II se destacou pelos seus resultados, posteriormente nesse algoritmo foi aplicado o conceito de hiper-heurística e comparado com o NSGA-II convencional considerando os três objetivos descritos anteriormente. Os resultados mostraram que a utilização de hiper-heurística provê melhores resultados do que os algoritmos convencionai. Quando comparados os métodos de seleção, o FRR-MAB apresentou um desempenho geral um pouco melhor que o aleatório, sendo que o aleatório não apresentou resultados com diferença estatística.

Ainda para o mesmo problema Ferreira et *al.* [35] aplicaram o conceito de hiper-heurística *on-line* no algoritmo MOEA/D juntamente com três métodos de seleção: UCB (MOEA/D-UCB), UCB-V (MOEA/D-V) e UCB-Tuned (MOEA/D-UCB-Tuned). Na avaliação dos resultados o algoritmo MOEA/D-UCB-Tuned se destacou, porém sem diferença estatística para os outros algoritmos. Posteriormente, o algoritmo MOEA/D-UCB foi selecionado devido ao seu baixo tempo computacional e comparado com o algoritmo MOEA/D-DRA. Nessa avaliação, o algoritmo MOEA/D-UCB apresentou melhores resultados. Entretanto, o algoritmo MOEA/D-UCB obteve desempenho similar quando comparado com o algoritmo MOEA/D utilizando o método de seleção aleatória (MOEA/D-RAND), os quais apresentaram equivalência estatística.

Em um trabalho posterior Ferreira et *al.* [36] compararam diferentes algoritmos. Além dos três objetivos considerados anteriormente foi utilizado um quarto objetivo baseado na similaridade dos produtos gerados. Nesse trabalho, os autores utilizaram o conceito de hiper-heurística *on-line* com o método de seleção UCB e aleatório (RAND) para avaliar os algoritmos NSGA-II, SPEA2, IBEA e MOEA/D-DRA. De acordo com os resultados apresentados o algoritmo NSGA-II utilizando o método de seleção UCB (NSGA-II-HH) obteve os melhores resultados em geral quando comparado aos outros algoritmos. Posteriormente, o algoritmo NSGA-II-HH foi comparado com o algoritmo NSGA-II utilizando o método de seleção aleatória (NSGA-II-RAND). Nessa avaliação, os resultados obtidos pelos algoritmos foram semelhantes, porém o NSGA-II-RAND obteve melhor desempenho considerando o tempo de execução. Dessa forma, os autores concluíram que a utilização de uma seleção aleatória é suficiente para encontrar boas soluções, porém para sistemas maiores o uso do NSGA-II-HH deve ser considerado, pois obtém melhores resultados. Além disso, os autores avaliaram o impacto da utilização de três e quatro objetivos, sendo que o uso de similaridade não implica na degradação no valor dos outros valores objetivo, mas sim auxilia a geração de maiores e melhores conjuntos de produtos.

4.4 Discussão

Nesta seção são abordados os aspectos relevantes encontrados em cada trabalho e identificadas oportunidades de pesquisa. Dessa forma, os trabalhos são separados por tipo de aplicação, estratégias ou otimização com HOMs, assim, dispostos nas Tabelas 4.1 e 4.2, sendo que as linhas representam os trabalhos e as colunas suas características.

Através da Tabela 4.1, observa-se que os trabalhos se atentaram em avaliar condições em que o escore de mutação dos HOMs fosse próximo ou melhor do que o escore obtido com

os mutantes gerados tradicionalmente. O trabalho de Polo et *al.* [110] foi precursor dos outros trabalhos apresentados e propôs estratégias eficientes em reduzir o número de mutantes, reduzir o número de mutantes equivalentes e preservar o escore de mutação próximo do teste com FOMs. Dentre todas as estratégias apresentadas a que mais se destacou foi a *LastToFirst*, ou *FirstToLast* como denominada em trabalhos posteriores.

Tabela 4.1: Trabalhos Relacionados - Estratégias para Mutantes de Ordem Superior

Referência	**Objetivo Geral**	**Estratégias**
Polo et *al.* [110]	Reduzir a quantidade de SOMs sem diminuir o escore de mutação	*LastToFirst*, *DifferentOperators* e *RandomMix*
Papadakis e Malevris [105]	Determinar o impacto da qualidade do teste com SOMs em relação à Mutação Forte e Mutação Fraca	*FirstToLast*, *SameNode*, *SameUnit*, *SU_F2Last* e *SU_DiffOp*
Mateo et *al.* [89]	Reduzir a quantidade de SOMs sem diminuir o escore de mutação considerando o teste no nível de sistema	*FirstToLast* (*LastToFirst*), *DifferentOperators*, *RandomMix* e *Each-Choice*

Analisando os trabalhos que abordam otimização baseada em busca (Tabela 4.2) é possível identificar que todos os trabalhos utilizam algoritmos evolutivos. Além disso, predomina a utilização de algoritmos mono-objetivos, sendo apenas os trabalhos de Landgon et *al.* [71] e Nguyen e Madeyski [95, 96] que aplicam otimização multi-objetivo utilizando os algoritmos NSGA-II, εMOEA, εNSGA-II e NSGA-III. Os trabalhos utilizam diferentes métricas para as funções de *fitness* como: número de casos de teste, escore de mutação, diferença semântica, diferença sintática e habilidade em revelar defeitos. Os trabalhos apresentados utilizam técnicas de SBSE no contexto de HOMs com bons resultados, assim como evidenciam a importância da utilização dos algoritmos evolutivos, cujo uso predomina nos trabalhos apresentados. As abordagens adotadas foram principalmente direcionadas em encontrar SSHOMs e *subtle* HOMs.

Tabela 4.2: Trabalhos Relacionados - Otimização com Mutantes de Ordem Superior

Referência	**Objetivo Geral**	**Algoritmos**	***Fitness***
Jia e Harman [61]	Classificar os HOMs e encontrar SSHOMs	Algoritmo Genético, Busca Gulosa e *Hill Climbing*	A facilidade de um HOM ser morto em relação aos seus FOMs constituintes
Omar et *al.* [101]	Encontrar *subtle* HOMs	Algoritmo Genético, Busca Gulosa e Seleção Aleatória	A diferença entre o HOM e seus FOMs constituintes em relação à diferença entre os casos de teste que matam o HOM e a união dos casos de teste que matam individualmente os FOMs constituintes

Continua na próxima página

Tabela 4.2 – continuação da página anterior

Referência	Objetivo Geral	Algoritmos	*Fitness*
Omar et *al.* [102]	Encontrar *subtle* HOMs	Busca Local Guiada, Busca Aleatória Restrita e Busca Enumerada Restrita	A diferença entre o HOM e seus FOMs constituintes em relação à diferença entre os casos de teste que matam o HOM e a união dos casos de teste que matam individualmente os FOMs constituintes
Omar et *al.* [104]	Encontrar *subtle* HOMs	A lgoritmo Genético, Busca Local, Busca Local Guiada por Interação de Dados (BLGD), Busca Local Guiada por Casos de Teste (BLGC), Busca Aleatória Restrita e Busca Enumerada Restrita	Dificuldade em matar um HOM e a diferença entre o HOM e seus FOMs constituintes em relação à diferença entre os casos de teste que matam o HOM e a união dos casos de teste que matam individualmente os FOMs constituintes
Landgon et *al.* [71]	Encontrar HOMs que fossem realísticos e difíceis de serem mortos	Programação Genética e NSGA-II	GP: diferença semântica e diferença sintática. NSGA-II: minimizar o número de testes necessários e minimizar a diferença sintática entre o mutante e o programa em teste
Harman et *al.* [51]	Geração de dados de teste	*Hill Climbing*	A interrupção máxima do fluxo de controle para aumentar a probabilidade de forte adequação
Harman et *al.* [52]	Encontrar SSHOMs	Algoritmo Genético e Busca Gulosa	A redução do número de casos de teste que matam os HOMs

Continua na próxima página

Tabela 4.2 – continuação da página anterior

Referência	Objetivo Geral	Algoritmos	*Fitness*
Nguyen e Madeyski [95]	Classificar os HOMs e encontrar SSHOMs	NSGA-II	Reduzir o número de casos de teste: (i) que matam o HOM e também matam seus FOMs constituintes; (ii) que matam o HOM e não matam seus FOMs constituintes; e (iii) que matam o HOM e matam alguns de seus FOMs constituintes
Nguyen e Madeyski [96]	Classificar os HOMs e encontrar SSHOMs	NSGA-II, εMOEA, εNSGA-II e NSGA-III	Reduzir o número de casos de teste: (i) que matam o HOM e também matam seus FOMs constituintes; (ii) que matam o HOM e não matam seus FOMs constituintes; e (iii) que matam o HOM e matam alguns de seus FOMs constituintes

Diante dos trabalhos apresentados nas Tabelas 4.1 e 4.2, observa-se um direcionamento das pesquisas em, além de gerar adequadamente um HOM, encontrar HOMs que: (i) possuam uma alta qualidade, geralmente essa qualidade é avaliada através de métricas, tais como: dificuldade em ser morto e alto escore de mutação; (ii) permitam reduzir o número de casos de teste a serem utilizados; (iii) substituam os seus FOMs constituintes de modo a reduzir o número de mutantes necessários; e (iv) revelem novos defeitos. Assim, considerando os trabalhos apresentados, são identificadas oportunidades de contribuição no estudo com HOMs, que são descritas a seguir.

- A intensificação dos estudos com as estratégias propostas por Polo et *al.* [110], visto seus resultados promissores;
- Apesar da inúmera gama de estratégias, essas foram utilizadas separadamente, nesse caso a escolha de determinadas estratégias em momentos diferentes da geração de HOMs poderia aumentar a qualidade dos HOMs gerados;
- A predominância da otimização mono-objetivo, sendo que apenas três trabalhos abordam otimização multi-objetivo, entretanto utilizam apenas os algoritmos NSGA-II, eMOEA, eNSGA-II e NSGA-III. Dessa forma, tratar o problema de HOMs com algoritmos multi-objetivos se torna um importante tópico de pesquisa que pode ser explorado. Nesse sentido diferentes algoritmos podem ser utilizados;
- Os trabalhos buscam encontrar unicamente SSHOMs ou *subtle* HOMs, não visando a encontrá-los ao mesmo tempo durante a otimização, o que poderia possibilitar determinar uma melhor exploração da busca;

- Apesar dos trabalhos aplicarem técnicas de otimização baseadas em busca, nenhum trabalho utilizou uma abordagem com hiper-heurística, que vem sendo apontada por alguns autores como uma tendência e tema de pesquisa futura na área de SBSE. A aplicação de uma abordagem com hiper-heurística, pode ser mais flexível no que possibilita abstrair os detalhes de implementação e configuração em relação ao problema, e pode contribuir no estudo com HOMs, assim, se tornando um tópico de pesquisa a ser explorado.

4.5 Considerações Finais

Os trabalhos apresentados confirmam que a utilização de HOMs tem se tornado uma área de estudo interessante devido aos seus bons resultados, os quais possibilitam a aplicação do teste de mutação de ordem superior através de diferentes abordagens, técnicas e objetivos. Além disso, os estudos que aplicaram técnicas de otimização no contexto de HOMs obtiveram resultados que possibilitam aumentar a qualidade dos mutantes gerados, encontrar determinados tipos de mutantes, diminuir o número de HOMs a serem gerados, dentre outros, embora haja poucos trabalhos relacionados a aplicação de otimização multi-objetivo. Entretanto, os estudos com HOMs são muito recentes na área de teste de *software* baseado em busca (*Search-Based Software Testing*, SBST) e seu campo de estudo pode ser ainda explorado e aprimorado com novas estratégias e abordagens.

Alguns dos estudos com hiper-heurística apresentados constatam que essa é uma técnica promissora que pode auxiliar os estudos na área de SBSE, embora hiper-heurísticas ainda não tenham sido aplicadas no contexto de HOMs. No trabalho do Guizzo et *al.* [48] é utilizado o conceito de dominância como forma de avaliar as hiper-heurísticas, uma proposta que obteve bons resultados e foi explorada com outros métodos de seleção em outros trabalhos [35, 36, 121]. Dessa forma, espera-se que a utilização do conceito de dominância possa obter bons resultados no contexto de HOMs.

Assim sendo, os trabalhos relacionados descritos neste capítulo, assim como as oportunidades de pesquisa identificadas motivaram a abordagem para criação de HOMs baseada em hiper-heurística, proposta no próximo capítulo.

Capítulo 5

Abordagem baseada em Hiper-heurística para Geração de Mutantes de Ordem Superior

Como mencionado no capítulo anterior, a análise de trabalhos da literatura mostra oportunidades de pesquisa relacionadas à criação de HOMs (Seção 4.4). Dentre estas podem-se citar: (i) a utilização de outros algoritmos multi-objetivos ainda não utilizados tais como o SPEA2. O uso e análise de outros algoritmos multi-objetivos poderá levar a uma melhora nos resultados. Além disso, o uso de diferentes objetivos poderá levar a uma melhoria nos valores de *fitness* das soluções encontradas, ou seja, no conjunto de HOMs gerados; e (ii) o uso de hiper-heurística neste contexto pode facilitar o trabalho do testador reduzindo esforços para implementar uma solução de busca, configurar seus algoritmos e utilizar as principais vantagens das diferentes estratégias (não baseados em busca) existentes.

Diante disso, neste capítulo é proposta a abordagem denominada *Hyper-Heuristic for Generation of Higher Order Mutants* (HG4HOM). HG4HOM visa a tratar os HOMs como um problema combinatorial, gerando HOMs que possam revelar defeitos, minimizar o número de mutantes ou substituir seus FOMs constituintes sem perder a eficácia do teste com relação ao escore de mutação. Para isso, HG4HOM realiza a aplicação de hiper-heurística na seleção de estratégias para a construção dos melhores conjuntos de mutantes de ordem superior com o uso de algoritmos multi-objetivos.

Para propor uma solução baseada em busca na área de SBSE, são necessários alguns itens importantes [53]: (i) uma representação adequada de uma solução para possibilitar sua manipulação pelo algoritmo de busca; (ii) operadores que melhorem uma solução e explorem o espaço de busca; e (iii) uma maneira de avaliar adequadamente a qualidade da solução através de uma função de *fitness*. Esses itens relativos à abordagem sendo proposta neste trabalho são apresentados a seguir.

5.1 Representação da População

Supondo que seja gerado um conjunto $F = \{f_1, f_2, ..., f_m\}$ de mutantes de primeira ordem (FOMs). Um indivíduo S da população é composto por um conjunto de mutantes de ordem superior (HOMs), tal que $S = \{h_1^{n_1}, ..., h_K^{n_K}\}$, de tal maneira que K seja mínimo e menor que m. Sendo que cada HOM h^n é composto por um conjunto de n FOMs, $h^n = \{f_1, f_i, ..., f_n\}$. No indivíduo não é necessário que os mutantes possuam a mesma ordem, ou seja, h_1 pode possuir ordem $n_1 = 3$, h_2 ordem $n_2 = 2$, e assim por diante até h_K com uma ordem n_K. Entretanto,

estabelece-se por definição que $2 \leq n_K \leq l$, onde l é o limite para n_K desejado e sendo fornecido pelo testador. Se esse limite não for especificado o valor padrão é $l = m$.

Considerando este fato o indivíduo na população é dado por um vetor S, ou seja, é um conjunto de HOMs, no qual cada HOM h_i^j é dado por um vetor de seus FOMs constituintes, conforme ilustrado no exemplo da Figura 5.1.

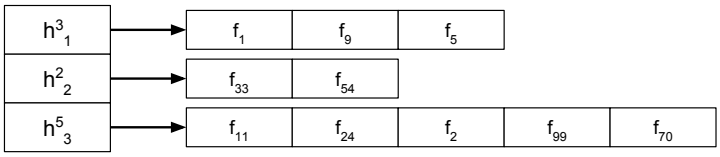


Figura 5.1: Representação da estrutura do indivíduo

No exemplo, o indivíduo S é composto por 3 HOMs. O primeiro mutante h_1^3 possui ordem 3 e é formado por 3 FOMs: f_1, f_9 e f_5. O segundo mutante h_2^2 possui ordem 2 e é formado por 2 FOMs: f_{33} e f_{54}. O terceiro mutante h_3^5 possui ordem 5 e é formado por 5 FOMs: f_{11}, f_{24}, f_2, f_{99} e f_{70}.

Desse modo, a representação da população é ilustrada através da Figura 5.2, a qual mostra um exemplo de população de tamanho três com seus respectivos indivíduos gerados.

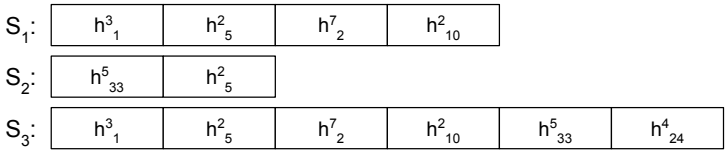


Figura 5.2: Exemplo representativo da população

Um HOM pode estar contido em mais de um indivíduo, entretanto um HOM não pode estar contido mais de uma vez em um mesmo indivíduo. Dessa maneira pode-se intensificar a diversividade em um indivíduo e permitir uma melhor exploração do espaço de busca. Tal representação visa a avaliação da força de um conjunto para que durante o processo de busca dos melhores conjuntos, seja possível encontrar os melhores mutantes.

5.2 Função de *Fitness*

Baseados em trabalhos relacionados [50, 52, 61, 71, 101–103, 105, 110] foram definidos três objetivos como forma de avaliar um indivíduo (S) da população, através do cálculo de *fitness*. A ideia é construir indivíduos com poucos HOMs, entretanto, esses HOMs devem possuir uma alta qualidade. Um HOM possui qualidade quando esse é classificado como sendo um *Strongly Subsuming* HOM (SSHOM) (Figura 2.1(a)), podendo ser utilizado para minimizar o número de mutantes e substituir seus FOMs constituintes sem perder a eficácia do teste em termos de escore de mutação, ou como sendo um *subtle* HOM, que também são desejados. Estes HOMs não são mortos por casos de teste que matam seus FOMs constituintes e podem estar associados a defeitos não revelados pelos FOMs (Figuras 2.1(c) e 2.1(d)). Considerando estes pontos, são propostos os seguintes objetivos.

5.2.1 Objetivo 1

O primeiro objetivo corresponde ao número de HOMs presentes em um indivíduo S em relação ao número de FOMs. Esse número é representado através da razão entre o número de mutantes presentes no indivíduo (K) e o número total de FOMs gerados (m), de acordo com a Equação 5.1. Esse objetivo visa a tratar o crescimento exponencial do número de mutantes para o teste de mutação com HOMs. Consequentemente, visa a reduzir o custo do teste de mutação com HOMs quando comparado aos FOMs, relacionado ao número de mutantes a serem utilizados. Desse modo, o primeiro objetivo visa a minimizar o número de mutantes presente em um indivíduo.

$$f(S) = \frac{K}{m} \tag{5.1}$$

5.2.2 Objetivo 2

O segundo objetivo foi adaptado de trabalhos anteriores [101–103] e busca mensurar os HOMs que compõem um indivíduo em relação a capacidade em revelar defeitos entre cada HOM e seus FOMs constituintes (*subtle* HOM). De acordo com a classificação de Jia e Harman [61], esse objetivo prioriza os *subtle* HOMs denominados Fracamente *Subsuming* e Desacoplados. Dessa forma, o objetivo busca encontrar HOMs com alta probabilidade de revelar defeitos não revelados por seus FOMs constituintes. Para isso, é definida inicialmente a Equação 5.2 de maximização que soma o número de HOMs Fracamente *Subsuming* e Desacoplados (*Weakly Subsuming and De-coupled*, WSD) presentes no indivíduo.

$$sumWSD(S) = \sum_{i=1}^{K} WSD(h_i^n) \tag{5.2}$$

Cada HOM (h^n) presente no indivíduo é avaliado através da Equação 5.3 para determinar se o mesmo corresponde a um HOM Fracamente *Subsuming* e Desacoplado.

$$WSD(h^n) = \frac{\frac{|\ (T_f \cup T_h)\ | - |\ (T_f \cap T_h)\ |}{|\ (T_f \cup T_h)\ |} + decoupled(h^n)}{2} \tag{5.3}$$

Suponha que $T = \{T_1, ..., T_i\}$ seja o conjunto de casos de testes para um sistema em teste e o conjunto T_f o subconjunto de T que mata os mutantes em F, sendo F um conjunto $F = \{f_1, f_2, ..., f_m\}$ de mutantes de primeira ordem (FOMs). Um mutante h^n pode possuir casos de teste em T que o matam, assim, T_h representa o subconjunto de T que mata h^n. Dessa maneira, a Equação 5.2 avalia a diferença de detecção de defeitos analisando a dissimilaridade dos conjuntos de casos de teste que matam o HOM e a união de todos os casos de teste que matam individualmente os FOMs constituintes.

A Equação 5.3 é definida em dois termos, onde o primeiro termo avalia a detecção de defeitos em um HOM e seus FOMs constituintes e o segundo termo identifica se um HOM é morto por algum caso de teste que também mata algum dos seus FOMs constituintes para que possa ser considerado um HOM Fracamente *Subsuming* e Desacoplado (Equação 5.4).

$$decoupled(h^n) = \begin{cases} 1 & \text{se } |T_h| < |T_f|,\ |T_h \cap T_f| = \varnothing \text{ e } T_h \neq \varnothing \\ 0 & \text{caso contrário} \end{cases} \tag{5.4}$$

O segundo termo serve para penalizar um HOM que não é desejado. Um HOM deve conter: (i) um conjunto de casos de teste que o matam, não sendo um mutante equivalente ($T_h \neq \emptyset$); (ii) um conjunto de casos de teste que matam o HOM com tamanho inferior ao do conjunto de casos de teste que matam seus FOMs constituintes ($|T_h| < |T_f|$); e (iii) um conjunto de casos de teste que matam o HOM e que não contenha casos de teste que matam algum dos seus FOMs constituintes ($|T_h \cap T_f| = \emptyset$).

5.2.3 Objetivo 3

O terceiro objetivo é semelhante ao segundo objetivo, entretanto visa mensurar o conjunto de HOMs que compõem um indivíduo em relação a capacidade de um HOM substituir seus FOMs constituintes sem perder a eficácia do teste, classificados como *strongly subsuming* HOMs (SSHOMs). A Equação 5.5 de maximização soma o número de SSHOMs presentes no indivíduo.

$$sumSSHOM(S) = \sum_{i=1}^{K} SSHOM(h_i^n) \tag{5.5}$$

Para determinar se um HOM presente no indivíduo corresponde a um SSHOM o mesmo é analisado através da Equação 5.6.

$$SSHOM(h^n) = \frac{\frac{|\ ((\cap T_f) \cap T_h)\ |}{|\ ((\cap T_f) \cup T_h)\ |} + intersection(h^n)}{2} \tag{5.6}$$

Ao contrário da Equação 5.3, a Equação 5.6 avalia a similaridade dos conjuntos de casos de teste que matam o HOM e a intersecção de todos os casos de teste que matam individualmente os FOMs constituintes ($\cap T_f$). Para isso, o segundo termo (Equação 5.7): (i) identifica se o HOM foi morto por algum caso de teste que esteja na interseção dos conjuntos de casos de teste dos seus FOMs constituintes ($T_h \subset \cap T_f$); (ii) não seja um HOM equivalente ($T_h \neq \emptyset$); e (iii) o conjunto de casos de teste que matam o HOM possui tamanho inferior ao do conjunto de casos de teste que matam seus FOMs constituintes ($|T_h| < |T_f|$).

$$intersection(h^n) = \begin{cases} 1 & \text{se } |T_h| < |T_f|,\ T_h \subset \cap T_f \text{ e } T_h \neq \emptyset \\ 0 & \text{caso contrário} \end{cases} \tag{5.7}$$

O segundo termo serve para penalizar um HOM caso esse não seja exclusivamente um SSHOM. Quando o conjunto de casos de teste do HOM (T_h) for maior que o conjunto resultante da união dos conjuntos associados aos seus FOMs (T_f) constituintes e se T_h não estiver na intersecção de T_f, além de considerar casos nos quais existam um conjunto de casos de teste que matem o HOM.

5.3 Operadores Genéticos

Devido a abordagem utilizar algoritmos genéticos multi-objetivos e visando a tratar adequadamente a manipulação dos indivíduos para o problema de permutação, os quais são de tamanho variável, foi realizada uma revisão da literatura com o intuito de identificar um conjunto de operadores utilizados para seleção e geração de HOMs ou que fossem capazes de se adequar a abordagem proposta. Esses trabalhos foram descritos no Capítulo 4. Posteriormente, com o

conjunto pré-definido de todos os operadores foi realizada uma seleção dos operadores para o trabalho, destes alguns foram adaptados.

5.3.1 Operadores de Recombinação

Para a estratégia de recombinação foi adotada a abordagem *One-Point Crossover* [55], conhecida também como *Single Point Crossover*.

No presente trabalho o problema a ser tratado é do tipo combinatorial, em que de acordo com a Seção 5.1 um indivíduo é de tamanho variável e composto por um conjunto de HOMs, sendo cada gene do indivíduo constituído de um HOM h_i^n de ordem n e índice i. Quando ocorre a realização do *Single Point Crossover* para esse tipo de problema é necessário tratar a junção dos indivíduos para não ocorrer a repetição de genes e tratar o ponto de corte devido ao tamanho variável do indivíduo.

De modo a tratar o problema de tamanho variável dos indivíduos o *Single Point Crossover* foi adaptado para realizar a recombinação seguindo algumas estratégias que são definidas a seguir.

- **Estratégia 1**: se ambos os indivíduos possuem tamanho superior a 1 gene, o ponto de corte é dado por um ponto determinado aleatoriamente para cada indivíduo;
- **Estratégia 2**: se o tamanho de um dos indivíduos é igual a 1 gene, o ponto de corte é feito somente no indivíduo com tamanho superior a 1, e realizada a combinação das partes com o indivíduo com tamanho de 1 gene, ou seja, o indivíduo com tamanho de 1 gene é combinado duas vezes;
- **Estratégia 3**: se ambos os indivíduos tem tamanho igual a 1 gene a operação não é realizada.

Para cada estratégia aplicada é realizada a união dos genes, onde cada gene repetido no indivíduo filho é removido e ao final da estratégia formam-se dois filhos. Para melhor exemplificar, as estratégias são ilustradas nas Figuras 5.3 e 5.4.

A Figura 5.3 exemplifica a seleção de dois indivíduos com tamanho superior a 1 gene. O primeiro indivíduo contém um tamanho ímpar e o segundo par. Posteriormente à identificação do tamanho de ambos os indivíduos é então selecionada a estratégia adequada a cada um. Para ambos os indivíduos é aplicada a Estratégia 1. Ao fim, os dois indivíduos selecionados geram dois novos indivíduos.

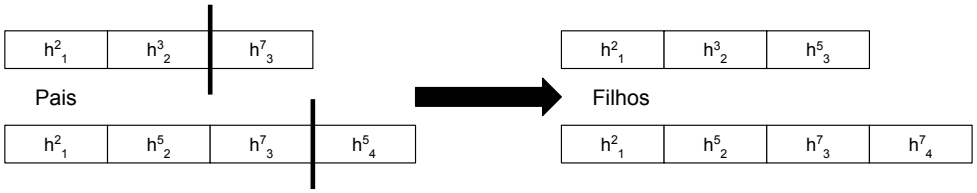


Figura 5.3: Esquema de recombinação de pais com tamanho maior que 1 gene

Já a Figura 5.4 trata da recombinação quando um dos indivíduos selecionados é de tamanho igual a 1. Para o tratamento dessa situação é aplicada a Estratégia 2, que seleciona o indivíduo que contém tamanho superior a 1 gene e determina um ponto de corte aleatório. Posteriormente, o indivíduo com tamanho igual a 1 é recombinado com as duas partes do outro indivíduo, verifica-se a existência de elementos repetidos e geram-se os dois novos indivíduos.

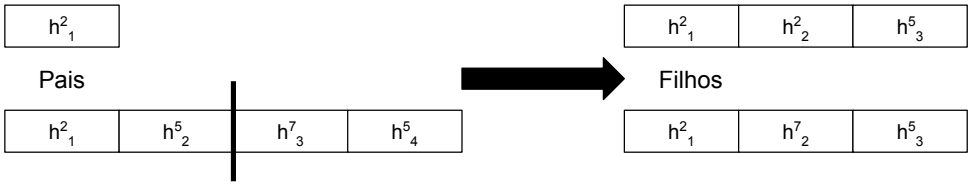


Figura 5.4: Esquema de recombinação de pai com tamanho igual a 1

5.3.2 Operadores de Mutação

O procedimento de mutação de um indivíduo tem a função de realizar alterações em um ou mais genes de um indivíduo a partir de seu estado inicial. Neste trabalho, utilizam-se três tipos de operadores de mutação: adição (*add*), remoção (*remove*) e troca (*swap*).

Os operadores do tipo adição e troca utilizam o processo de criação de HOM (Seção 5.4). Devido ao comportamento análogo dos operadores de adição e troca, o procedimento em comum realizado por ambos é apresentado no Algoritmo 5.1.

Algoritmo 5.1: Procedimento de mutação do tipo adição/troca

```
1  Entrada: S, F, l, Ω, θ
2  Saída: S
3  Início
4      se θ >= λ então
5          h^n ← Chama o processo de criação de HOMs passando Ω, l, F;
6          se h^n ! = ∅ então
7              Realiza o procedimento de Adição ou Troca em S utilizando h^n;
8              Remove os HOMs repetidos em S;
9          fim se
10     fim se
11     retorna S;
12 Fim
```

S - indivíduo;
F - o conjunto de todos os FOMs, $F = \{f_1, f_2, ..., f_m\}$;
l - limite da ordem do mutante superior;
Ω - estratégia a ser aplicada;
θ - probabilidade de mutação;
λ - probabilidade de mutação gerado aleatoriamente;
h^n - um HOM construído de n FOMs.

Na Linha 4 é verificada a probabilidade de mutação. Posteriormente, na Linha 5 é gerado um novo HOM h^n através do processo de criação de HOMs, descrito na Seção 5.4. Na Linha 6 é verificado se o mutante gerado é vazio, o que significa que o HOM gerado apresentou algum erro e caso isso ocorra a operação de mutação não é aplicada. Por fim, se o HOM gerado não é vazio (não apresentou nenhum erro) o mesmo é adicionado ou trocado por outro HOM presente no indivíduo S, Linha 7. Na linha 8 é verificado se o procedimento que alterou o indivíduo não acrescentou algum HOM que já estava presente no indivíduo e caso isso ocorra o HOM repetido é removido.

A partir das estratégias descritas nas Seções 5.4 e 5.6.3 foram definidos operadores de mutação para a manipulação de um indivíduo da população. Os operadores de mutação baseados nas estratégias adaptadas são descritos e apresentados na Tabela 5.1. Na tabela é identificado o

tipo de operação (Coluna 1), o nome do operador proposto (Coluna 2) e a descrição explanando quanto ao tipo e estratégia empregada para a criação de um HOM (Coluna 3).

Tabela 5.1: Operadores de Mutação

Operação	**Operador**	**Descrição**
Adição	*AddFirst2Last*	Adiciona um novo mutante ao indivíduo utilizando a estratégia de criação de HOM *First2Last*.
	AddRandomMixLimited	Adiciona um novo mutante ao indivíduo utilizando a estratégia de criação de HOM *Random* com restrição.
	AddDiffOp	Adiciona um novo mutante ao indivíduo utilizado a estratégia de criação de HOM *DiffOp*.
	AddEach-Choice	Adiciona um novo mutante ao indivíduo utilizando a estratégia de criação de HOM *Each-Choice*.
Troca	*SwapFirst2Last*	Troca um mutante do indivíduo por outro novo mutante utilizando a estratégia de criação de HOM *First2Last*.
	SwapRandomMixLimited	Troca um mutante do indivíduo por outro novo mutante utilizando a estratégia de criação de HOM *Random* com restrição.
	SwapDiffOp	Troca um mutante do indivíduo por outro novo mutante utilizando a estratégia de criação de HOM *DiffOp*.
	SwapEach-Choice	Troca um mutante do indivíduo por outro novo mutante utilizando a estratégia de criação de HOM *Each-Choice*.
Remoção	*Remove*	Remove um mutante do indivíduo.

Os operadores do tipo adição realizam a adição de um novo mutante de ordem superior, gerado a partir de determinada estratégia, no indivíduo selecionado. Os operadores do tipo troca realizam a substituição de um mutante, selecionado aleatoriamente, presente no indivíduo por um novo mutante de ordem superior, gerado a partir de determinada estratégia. Por fim, o operador do tipo remoção seleciona aleatoriamente um mutante do indivíduo e o remove. Devido a possibilidade de haver somente um HOM presente no indivíduo a operação de remoção não é realizada nesse caso.

Na Figura 5.5 é apresentado o exemplo de cada tipo de mutação. Na figura é exemplificada a adição de um novo mutante de ordem 4, no final do indivíduo (índice 4), sendo representado por h_4^4. Posteriormente é ilustrada a troca do mutante h_2^6 pelo mutante h_4^2 na posição 2. Por fim, é ilustrada a remoção de um mutante, h_2^6. Para cada tipo de mutação foi representado o novo mutante (ou o mutante removido) através da cor cinza e o fluxo de um novo indivíduo através de uma seta, onde o primeiro indivíduo, parte superior, sofre uma mutação e gera um novo indivíduo, parte inferior.

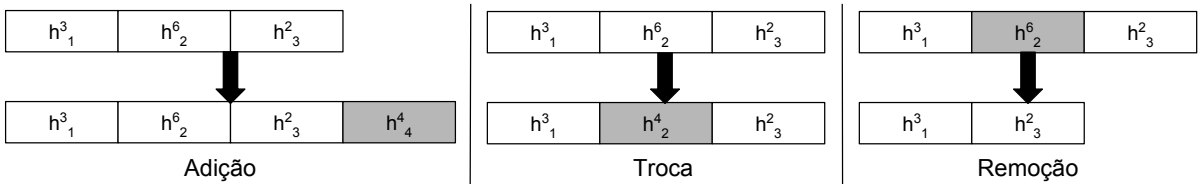


Figura 5.5: Exemplo de aplicação dos tipos de operadores de mutação

5.3.3 Operador de Seleção

Os operadores de seleção utilizados para selecionar os melhores indivíduos são os operadores: Torneio Binário (*Binary*), *K-Tournament*, *N-ary Tournament* e Aleatório (*Random*).

Na seleção por Torneio são selecionados aleatoriamente n indivíduos da população para o torneio com igual probabilidade. Dessa seleção, um indivíduo será escolhido de acordo com uma probabilidade θ, previamente definida, vencendo o indivíduo com o melhor *fitness*. O operador por Torneio é o mais utilizado, pois ele oferece a vantagem de não necessitar a realização da comparação com todos os indivíduos da população [7].

No caso do Torneio Binário são escolhidos dois indivíduos aleatórios, o indivíduo com melhor *fitness* vence o torneio e é selecionado. *K-Tournament* aplica k torneios binários (comparações) entre soluções aleatórias. *N-ary Tournament* seleciona a melhor solução dentre n soluções escolhidas aleatoriamente. A seleção aleatória seleciona aleatoriamente um indivíduo.

Os operadores selecionados realizam a seleção de dois indivíduos candidatos à aplicação dos operadores de recombinação e mutação. Posteriormente a seleção e aplicação dos operadores de recombinação e mutação, são gerados novos indivíduos que se juntarão a população.

5.4 Processo de Criação de HOMs

O processo de criação de HOMs (PCH) é apresentado na Figura 5.6. Nesse processo, a determinação dos HOMs equivalentes não é realizada devido à complexidade da tarefa. Primeiramente deve ser definida uma estratégia Ω de seleção de FOMs, um limite l para a ordem do HOM a ser criado, tal que $2 \leq n \leq l \leq m$, e um conjunto de mutantes de primeira ordem F.

Após selecionados os FOMs que constituirão o HOM, um id baseado nos nomes dos FOMs é gerado, por exemplo, a um HOM formado pelos FOMs `AOIS_1` e `ROR_13` será atribuído o id `AOIS_1_and_ROR_13`. Através desse id é verificado se há algum HOM já criado anteriormente junto a um repositório que armazena os HOMs criados, caso o HOM exista ele é retornando ao procedimento que solicitou o PCH. Se essa verificação não fosse realizada um HOM poderia ser criado e avaliado inúmeras vezes, necessitando ser executado com o conjunto de casos de teste disponíveis todas as vezes, o que prejudicaria o desempenho computacional.

Em seguida, se não houver nenhum HOM no repositório de HOMs que possua o id igual ao id gerado, é verificado junto ao repositório de erros se há algum HOM com esse id. Caso exista algum HOM nesse repositório, então, o PCH irá ignorar a criação de um HOM com os FOMs selecionados e retornará ao procedimento que solicitou o PCH. O repositório de erros contempla os HOMs que ao serem criados apresentaram erros como problemas relacionados a união dos FOMs, compilação ou sobreposição de mutação.

Posteriormente, caso não exista nenhum HOM no repositório de erros com o id igual ao id gerado, é verificado se os FOMs selecionados para compor o novo HOM possuem sobreposição de mutação e caso isso ocorra o HOM é salvo no repositório de erros. Então, o PCH irá ignorar a criação do HOM com os FOMs selecionados e retornará ao procedimento que solicitou

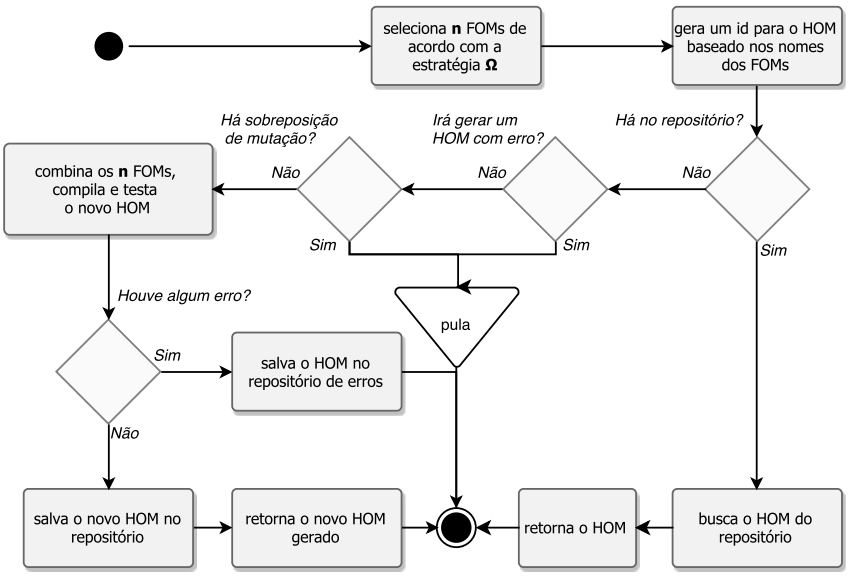


Figura 5.6: Processo de criação de HOMs

o PCH. A sobreposição de mutação é uma restrição que visa a assegurar que ao gerar um HOM os seus FOMs constituintes não possuam mutações na mesma linha e mesmo trecho de código. Considerando a criação de um SOM, dois FOMs com mutações na mesma linha de código $A > B \;\&\&\; B < C$ podem apenas ser combinados se as mutações não são na mesma sub-expressão, por exemplo, uma deve ser em $A > B$ e outra em $B < C$.

Por outro lado, caso não exista sobreposição de mutação, então, as mutações dos FOMs selecionados são combinadas e um novo HOM é gerado que em seguida é compilado e testado. Caso ocorra algum erro durante esse processo, por exemplo, erro de compilação ou execução, o HOM é considerado anômalo e descartado. Seu id é então incluído no repositório de erros e em seguida o PCH encerra-se, caso contrário, o HOM é salvo no repositório de HOMs e retornado ao procedimento que solicitou o PCH.

Devido ao comportamento análogo da aplicação das estratégias o procedimento em comum realizado por elas é apresentado no Algoritmo 5.2.

Em cada estratégia empregada seleciona-se um número n de FOMs de F que ainda não foram utilizados para compor algum HOM já gerado pela estratégia. Quando ocorre de não haver mais FOMs presentes em F que ainda não foram selecionados para compor algum HOM, então, a estratégia é "reiniciada". O reinício é um processo em que todos os FOMs de F podem novamente serem selecionados.

Se durante a aplicação da estratégia alguns FOMs forem selecionados, porém não haver mais FOMs que ainda não foram selecionados para compor algum HOM, então, a estratégia é reiniciada apenas para aquele momento da seleção de FOMs e os FOMs já selecionados pela estratégia são marcados como selecionados. Desse modo, é possível que na próxima aplicação da estratégia ela seja reiniciada e todos os FOMs estejam novamente disponíveis para seleção. Essa situação pode ocorrer, por exemplo, durante a seleção de FOMs para compor um SOM e com um número de FOMs ímpar.

O PCH é totalmente independente das estratégias selecionadas, ou seja, elas não interagem. Cada estratégia mantém a sua informação em relação aos FOMs selecionados de forma separada, assim, uma estratégia 1 pode selecionar um mutante não utilizado de F e posteriormente uma estratégia 2 poderá selecionar o mesmo mutante.

Algoritmo 5.2: Aplicação das estratégias para geração de HOM

```
1  Entrada: Ω, F, l
2  Saída: FOMsSelecionados
3  Início
4      se Não há mais FOMs a serem selecionados de F então
5          Reinicia a lista de FOMs disponíveis a serem selecionados;
6      fim se
7      FOMsSelecionados ← ∅;
8      n ← gera aleatoriamente um número entre 2 e l;
9      i ← 0;
10     enquanto i < n faça
11         f ← ∅;
12         se Não há mais FOMs a serem selecionados de F então
13             Reinicia a lista de FOMs disponíveis a serem selecionados;
14             Remove da lista de FOMs disponíveis os FOMs presentes em
                FOMsSelecionados;
15         fim se
16         f ← Aplica a estratégia Ω;
17         Adiciona f em FOMsSelecionados;
18         Marca o FOM f como já selecionado em F;
19         i++;
20     fim enquanto
21     retorna FOMsSelecionados;
22 Fim
```

F - conjunto de todos os FOMs, $F = \{f_1, f_2, ..., f_m\}$;
l - limite da ordem do mutante superior;
Ω - estratégia a ser aplicada;
n - número de FOMs a serem selecionados;
i - contador de FOMs selecionados;
$FOMsSelecionados$ - FOMs selecionados pela estratégia Ω.

As estratégias empregadas para a mutação de um indivíduo foram adotadas e adaptadas de [89, 110]. Das estratégias investigadas, foram selecionadas as seguintes [110]: *FirstToLast*, *RandomMix* e *Different-Operators*, denotadas respectivamente como *First2Last*, *Random* e *DiffOp*. Além das três estratégias mencionadas ainda foi selecionada a estratégia *Each-Choice* de [89]. Tais abordagens apresentaram em experimentos uma redução no número de mutantes de ordem superior gerados, mantendo a eficácia do teste. Devido à facilidade de geração, execução e bons resultados encontrados com essas estratégias as mesmas foram escolhidas e adaptadas para a abordagem proposta.

A seguir são descritas as estratégias selecionadas, as quais foram adaptadas, sendo identificado somente o procedimento de seleção dos FOMs pela respectiva estratégia.

Estratégia *FirstToLast*

A estratégia *FirstToLast* (*First2Last*) visa a priorizar a seleção dos mutantes ainda não utilizados procurando nos extremos de uma lista de mutantes F, o topo e o final da lista.

O Algoritmo 5.3 apresenta o procedimento realizado na aplicação da estratégia *First2Last* para a geração de um mutante de ordem superior n. Na Linha 4 é inicializado o mutante de

primeira ordem f que conterá um mutante de F selecionado durante a execução da estratégia. Na Linha 5 é verificado se o mutante a ser selecionado será do topo ou do final de F de acordo o valor de i, o qual representa o número de mutantes já selecionados. Se o número de mutantes já selecionados é par o procedimento segue para a Linha 6, caso contrário para a Linha 8. Na Linha 6 é realizada a seleção do topo da lista e na Linha 8 é realizada a seleção do final da lista. Por fim, o mutante f, selecionado pela estratégia *First2Last* é retornado, Linha 10.

Algoritmo 5.3: Estratégia de mutação *First2Last*

```
1  Entrada: F, i
2  Saída: f
3  Início
4      f ← ∅;
5      se i % 2 == 0 então
6          f ← seleciona um mutante não utilizado do topo de F;
7      senão
8          f ← seleciona um mutante não utilizado do final de F;
9      fim se
10     retorna f;
11 Fim
```

A Figura 5.7 apresenta um exemplo de aplicação do operador em questão dada uma ordem $n = 2$. Na primeira parte é ilustrada a seleção dos mutantes do topo e do final da lista de mutantes não selecionados. Posteriormente, a geração do novo mutante de ordem superior. Por fim, a inclusão do novo mutante em um indivíduo S.

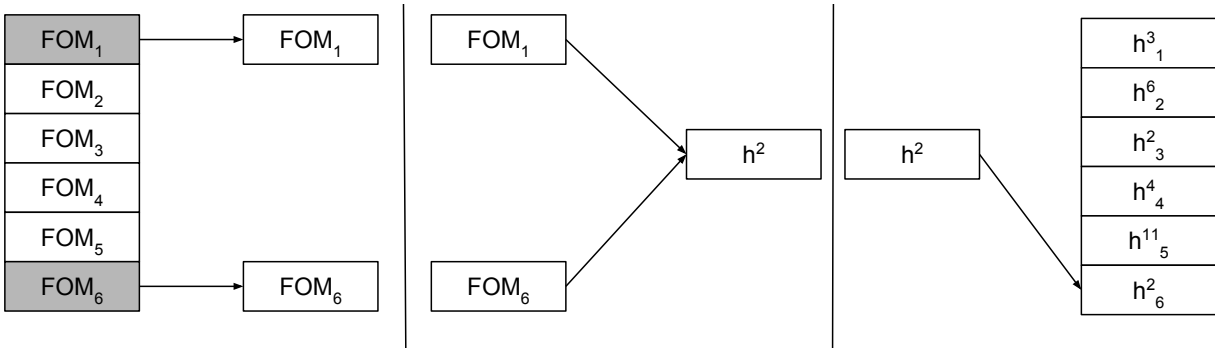


Figura 5.7: Exemplo de aplicação do operador *First2Last*

Estratégia *RandomMix*

A estratégia *RandomMix* (*Random*) seleciona aleatoriamente um mutante de primeira ordem que não esteja presente no mutante de ordem superior a ser criado.

O Algoritmo 5.4 apresenta o procedimento realizado na aplicação da estratégia *Random* para a geração de um mutante de ordem superior n. Na Linha 4 é inicializado o mutante de primeira ordem f que conterá um mutante de F selecionado durante a execução da estratégia. Na Linha 5 é realizada a seleção de forma aleatória de um mutante. Por fim, o mutante f selecionado pela estratégia *Random* é retornado, Linha 6.

A Figura 5.8 apresenta um exemplo de aplicação do operador em questão dado um limite de ordem $n = 2$. Na primeira parte é ilustrada a seleção dos mutantes de forma aleatória

Algoritmo 5.4: Estratégia de mutação *Random*

```
1 Entrada: F
2 Saída: f
3 Início
4 |   f ← ∅;
5 |   f ← seleciona aleatoriamente um mutante de F;
6 |   retorna f;
7 Fim
```

da lista de mutantes de primeira ordem. Posteriormente, a geração do novo mutante de ordem superior. Por fim, a inclusão do novo mutante em um indivíduo S.

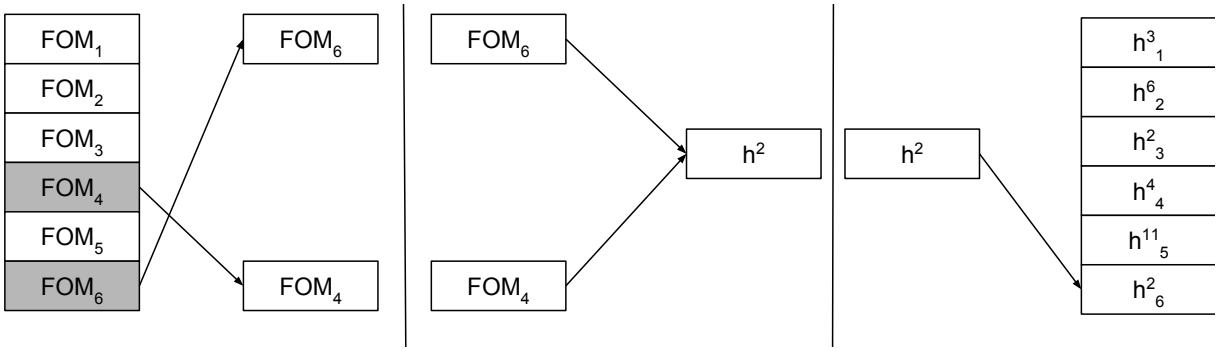


Figura 5.8: Exemplo de aplicação do operador *Random*

Estratégia *Different-Operators*

A estratégia *Different-Operators* (*DiffOp*) visa a selecionar os mutantes, ainda não utilizados, produzidos por diferentes operadores. Esse algoritmo combina mutantes provenientes de diferentes operadores de mutação, por exemplo, se um mutante foi produzido por um operador O_1 ele não será combinado com outro mutante produzido pelo mesmo operador e sim com de outro operador como O_2.

O Algoritmo 5.5 apresenta o procedimento realizado na aplicação da estratégia *DiffOp* para a geração de um mutante de ordem superior n. Na Linha 4 é inicializado o mutante de primeira ordem f que conterá um mutante de F selecionado durante a execução da estratégia. Na Linha 5 é realizada a seleção de um mutante ainda não selecionado do topo de F gerado com um operador diferente dos operadores dos mutantes presentes em $FOMsSelecionados$. Por fim, o mutante f selecionado pela estratégia *DiffOp* é retornado, Linha 6.

A Figura 5.9 apresenta um exemplo de aplicação do operador em questão dado um limite de ordem $n = 2$. Na primeira parte é ilustrada a seleção dos mutantes inciando pelo topo da lista e gerados com diferentes operadores, identificado entre parênteses, da lista de mutantes não selecionados. Posteriormente, a geração do novo mutante de ordem superior. Por fim, a inclusão do novo mutante em um indivíduo S.

Estratégia *Each-Choice*

A estratégia *Each-Choice* visa a selecionar os mutantes ainda não utilizados de forma sequencial, procurando a partir do topo da lista de mutantes não utilizados até o final.

Algoritmo 5.5: Estratégia de mutação *DiffOp*

```
1 Entrada: F, FOMsSelecionados
2 Saída: f
3 Início
4 |  f ← ∅;
5 |  f ← seleciona um mutante não utilizado do topo de F gerado com um
  |    operador diferente dos mutantes presentes em FOMsSelecionados;
6 |  retorna f;
7 Fim
```

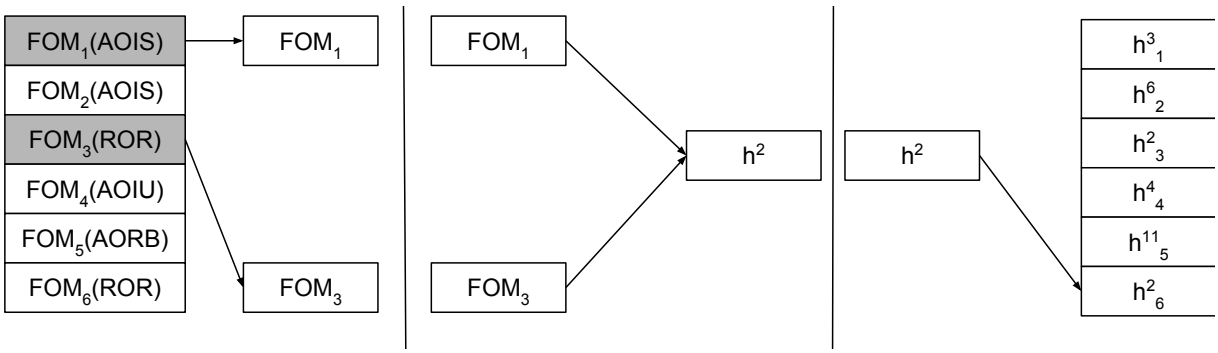


Figura 5.9: Exemplo de aplicação do operador *DiffOp*

O Algoritmo 5.6 apresenta o procedimento realizado na aplicação da estratégia *Each-Choice* para a geração de um mutante de ordem superior n. Na Linha 4 é inicializado o mutante de primeira ordem f que conterá um mutante de F selecionado durante a execução da estratégia. Na Linha 5 é realizada a seleção de um mutante ainda não selecionado do topo de F. Por fim, o mutante f selecionado pela estratégia *Each-Choice* é retornado, Linha 6.

Algoritmo 5.6: Estratégia de mutação *Each-Coice*

```
1 Entrada: F, FOMsSelecionados
2 Saída: f
3 Início
4 |  f ← ∅;
5 |  f ← seleciona um mutante não utilizado do topo de F e que não esteja
  |    presente em FOMsSelecionados;
6 |  retorna f;
7 Fim
```

A Figura 5.10 apresenta um exemplo de aplicação do operador em questão dado um limite de ordem $n = 2$. Na primeira parte é ilustrada a seleção de um par de mutantes a partir do topo da lista de mutantes não selecionados e de forma sequencial. Posteriormente, a geração do novo mutante de ordem superior. Por fim, a inclusão do novo mutante em um indivíduo S.

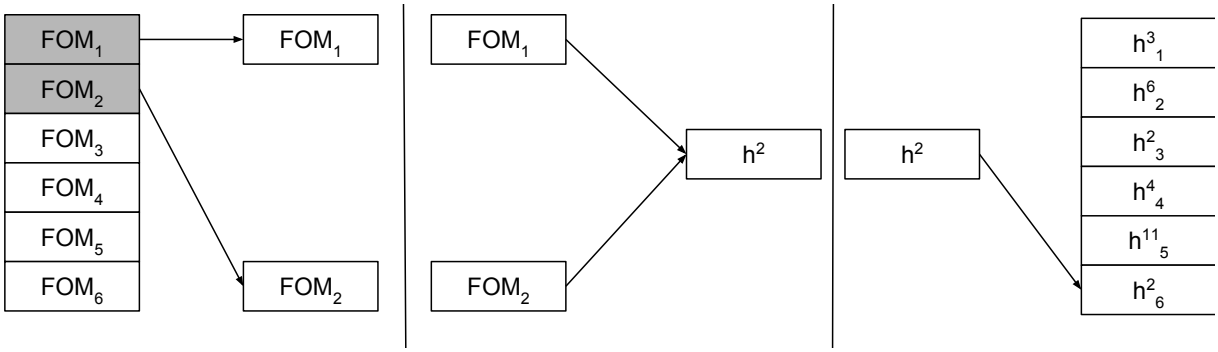


Figura 5.10: Exemplo de aplicação do operador *Each-Choice*

5.5 HG4HOM

A abordagem proposta, denominada *Hyper-Heuristic for Generation of Higher Order Mutants* (HG4HOM), trata-se de uma abordagem multi-objetivo que utiliza hiper-heurística de seleção com aprendizado *on-line* visando a gerar conjuntos de mutantes de ordem superior (HOMs) mais adequados a um domínio de um problema a ser investigado.

HG4HOM foca na redução dos custos do teste de mutação através da redução de mutantes, porém as abordagens que aplicam estratégias de redução de mutantes, mesmo sendo mais práticas, possuem como desvantagem a baixa eficácia em relação as abordagens que utilizam um conjunto completo de mutantes [78]. Desse modo, HG4HOM trabalha em uma redução de mutantes e ao mesmo tempo na melhoria da qualidade dos mutantes gerados em relação ao número de defeitos revelados. Cada solução gerada pela abordagem visa à eficácia dos testes com uma quantidade de mutantes igual ou inferior ao número de mutantes de primeira ordem (FOMs) que seriam gerados pelo Teste de Mutação tradicional. A HG4HOM contém dois procedimentos iniciais essenciais: a configuração inicial do processo de otimização e a geração dos FOMs de um sistema em teste, os quais podem ser realizados paralelamente. A Figura 5.11 ilustra o fluxo de trabalho da abordagem HG4HOM.

A configuração inicial possui cinco componentes essenciais, os quais definem: (i) a criação da população inicial; (ii) o método de avaliação da qualidade das soluções (*fitness*); (iii) o algoritmo evolutivo multi-objetivo (*Multiobjective Evolutionary Algorithm*, MOEA); (iv) as heurísticas de baixo nível (LLHs); e (v) o método de seleção.

O procedimento de criação da população inicial define como deverá ser criada a primeira população do procedimento de otimização. Para isso, é necessário determinar qual estratégia será utilizada para gerar os HOMs que irão compor os indivíduos.

O MOEA representa um determinado algoritmo que será utilizado para encontrar as soluções para o problema. Para esse algoritmo são definidas as suas configurações, por exemplo, tamanho da população e o critério de parada. No MOEA selecionado é aplicada a hiper-heurística durante o processo evolutivo da seguinte forma: (i) selecionam-se as soluções candidatas; (ii) aplica-se uma heurística de baixo nível (LLH), escolhida pelo método de seleção, nas soluções selecionadas; (iv) avaliam-se as soluções; (v) avalia-se a LLH aplicada.

As LLHs definidas na abordagem são representadas através de conjuntos combinados de operadores de mutação e recombinação. As LLHs utilizam procedimentos de manipulação de uma solução para gerar novas soluções, tal manipulação poderá necessitar que um novo HOM seja criado para compor a solução. Por exemplo, quando aplicado um procedimento de mutação de adição de HOM ao indivíduo este irá aplicar a estratégia de geração de HOM do operador,

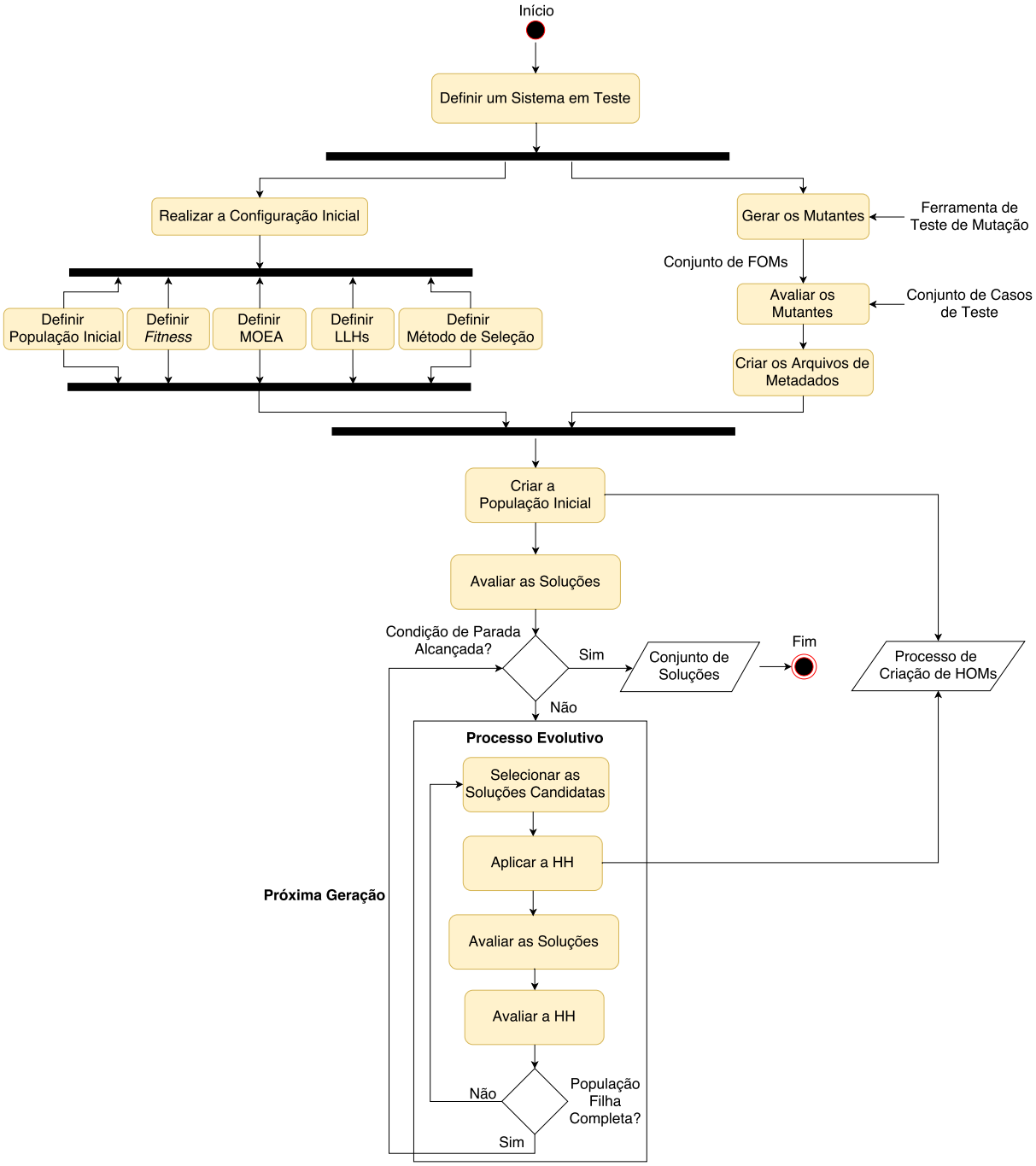


Figura 5.11: Representação do fluxo de trabalho da HG4HOM

gerando um novo HOM que ainda não pertence ao o indivíduo. Posteriormente, o HOM gerado é adicionado ao indivíduo.

Em seguida é necessário definir qual método de seleção a ser utilizado pela hiper-heurística. Para este trabalho os métodos de seleção utilizados são *Choice Function* (CF), adaptado por Maashi et *al.* [79], o *Fitness-Rate-Rank based Multi-Armed Bandit* (FRR-MAB) [72], além do procedimento aleatório de seleção. O método *Choice Function* foi escolhido por ser uma heurística de seleção já utilizada na área [48, 79] e o FRR-MAB [72] devido aos seus excelentes resultados focando na tarefa do *Credit Assigment* [44, 72]. No procedimento aleatório de seleção, uma LLH é selecionada aleatoriamente para ser aplicada.

O processo de geração de FOMs é realizado por uma ferramenta escolhida pelo usuário. Então, os FOMs gerados são executados e submetidos para avaliação com o conjunto de casos de teste fornecido. Através da avaliação é possível determinar o escore de mutação de cada FOM e por quais casos de teste ele é morto, bem como identificar quais são os FOMs anômalos. Finalizada a execução e avaliação dos mutantes, então, as informações obtidas com esses procedimentos são registradas em arquivos de metadados de cada mutante. Cada arquivo de metadados contém informações sobre a mutação, número da linha alterada, operador aplicado, o método ou operador alterado e a mutação atual do FOM. As informações presentes nos metadados serão utilizadas como base para a construção dos HOMs. No Código 5.1 é exemplificado o arquivo de metadados de um FOM, o qual utiliza o formato JSON[1].

Código 5.1: Exemplo de arquivo de metadados de um FOM (adaptada de [103])

```
{
        "className": "Cal.java",
        "equivalent": false,
        "lineNumber": 14,
        "mutantStatement": "day2--",
        "mutationScore": 0.0,
        "name": "AODS_1",
        "operator": "AODS",
        "originalStatement": "day2",
        "parent": "day2 - day1",
        "path": ".\CalProgram \...\ AODS_1\Cal.java",
        "testCases": []
}
```

O arquivo de metadados com a informação de cada FOM é gerado para possibilitar uma melhor manipulação dos FOMs, prevenindo assim um possível excesso de informações desnecessárias em memória, bem como possibilitar que a geração dos FOMs possa ser feita por qualquer ferramenta de escolha do usuário. Ao fim da geração dos FOMs é definido o conjunto de FOMs a serem utilizados no processo de otimização da abordagem.

Após a realização dos dois procedimentos iniciais, são derivados conjuntos de HOMs para o teste de mutação por meio de uma otimização multi-objetivo com hiper-heurística. Nesse processo de otimização, o MOEA gera uma população inicial que será posteriormente avaliada e submetida ao processo geracional. No processo geracional cada MOEA irá se comportar de acordo com sua funcionalidade, entretanto, no processo evolutivo é realizada a aplicação da hiper-heurística (seleção e aplicação de LLHs). O movimento de aceitação, o qual aceita ou rejeita uma solução gerada através de uma LLH, não é utilizado na HG4HOM, sendo passada esta responsabilidade ao MOEA, que irá selecionar as soluções mais adequadas durante seu processo geracional. Por fim, após todo o procedimento de otimização, é definido um conjunto de soluções que contém os conjuntos de HOMs derivados e representa a população final do MOEA. O processo de criação de HOMs utilizado durante a criação da população inicial e pelas LLHs é detalhado na Seção 5.4.

5.6 Aspectos de Implementação

Nesta seção são detalhados aspectos de implementação da abordagem proposta utilizando os itens descritos nas seções anteriores. Desse modo, são apresentadas particularidades do desenvolvimento, entretanto, as características do desenvolvimento podem ser alterados de acordo com a necessidade.

[1]JavaScript Object Notation - `http://www.json.org`

5.6.1 Aspectos Gerais

Primeiramente, necessita-se definir a linguagem de programação a ser utilizada. Nesse caso, a linguagem escolhida para a implementação da abordagem H4HOM foi Java. Definida a linguagem, então, foi verificado junto a literatura *frameworks* que facilitassem o desenvolvimento e auxiliassem na otimização de problemas. Desse modo, foi escolhido o *framework* jMetal 5.1 [93] que é um *framework* para manipulação de problemas de otimização. Esse *framework* possui uma estrutura com formulação de soluções, problemas, indicadores de qualidade, algoritmos, além de utilizar uma arquitetura consolidada e utilizada na literatura [45, 97]. A versão selecionada é a mais recente até o momento e foi escolhida devido a fácil manipulação e implementação, bem como a rapidez para a execução dos experimentos.

Em seguida, de acordo com a abordagem HG4HOM, necessita-se definir as configurações inciais e uma ferramenta de mutação a ser utilizada, esses itens são descritos a seguir.

O procedimento de criação da população inicial é realizado através de alguma das estratégias propostas: *First2Last*, *RandomMixLimited*, *Different-Operators* ou *Each-Choice*. A definição da melhor estratégia para a inicialização da população foi avaliada nos experimentos descritos no próximo capitulo.

A avaliação da qualidade das soluções é realizada de acordo com as funções de *fitness* propostas na Seção 5.2.

Para a definição dos MOEAs a serem utilizados verificou-se na literatura [20,36,48,121] aqueles que ganharam destaque devido à sua eficiência, facilidade de implementação e aqueles que já foram utilizados no campo de SBSE, desse modo, foram escolhidos o *Non-dominated Sorting Genetic Algorithm II* (NSGA-II) [24] e o *Strength Pareto Evolutionary Algorithm 2* (SPEA2) [131].

Posteriormente, definiram-se as LLHs a serem utilizadas através dos operadores de mutação e recombinação apresentados na Seção 5.3.2 e na Seção 5.3.1, respectivamente. Dessa forma, na Tabela 5.2 apresenta a combinação dos operadores de mutação: *AddFirst2Last*, *AddRandomMixLimited*, *AddDiffOp*, *AddEach-Choice*, *SwapFirst2Last*, *SwapRandomMixLimited*, *SwapDiffOp*, *SwapEach-Choice* e *Remove*; com o operador de recombinação *Single Point Crossover*. Além disso, o uso apenas do operador de recombinação sem mutação, bem como a aplicação do inverso, o uso apenas do operador de mutação sem a recombinação. Assim, propõem-se a utilização de 19 heurísticas de baixo nível, representadas por h1 até h19, para a aplicação na abordagem HG4HOM.

A ferramenta para a criação, geração e execução dos mutantes de primeira ordem e ordem superior selecionada é a ferramenta MuJava [77]. Essa ferramenta já foi utilizada em uma abordagem de geração de HOMs por Omar et *al.* [101], a qual serviu de inspiração para o estudo e desenvolvimento da abordagem HG4HOM. Dessa maneira, os procedimentos em relação ao uso da ferramenta para tratar os FOMs e HOMs são descritos nas Seções 5.6.2 e 5.6.3.

Durante a aplicação da hiper-heurística o desempenho de uma LLH é avaliado através do conceito de dominância, como proposto por Guizzo et *al.* [48]. Além disso, uma restrição foi aplicada em relação ao tamanho de um indivíduo. O limite definido para o tamanho de um indivíduo, visa o custo do processo de teste para o Engenheiro de *Software*. O custo do Teste de Mutação de ordem superior deve ser igual ou inferior ao custo do Teste de Mutação de primeira ordem. Dessa forma, o custo é tratado como a quantidade de HOMs em relação aos FOMs a serem testados, onde $K \leq m$. Caso um indivíduo possua um tamanho superior a um tamanho pré-determinado o mesmo é penalizado e descartado no processo de otimização.

Tabela 5.2: Composição das heurísticas de baixo nível

Mutação	Recombinação	LLH
λ	Single Point	h1
AddDiffOp	Single Point	h2
SwapDiffOp	Single Point	h3
AddEach-Choice	Single Point	h4
SwapEach-Choice	Single Point	h5
AddFirst2Last	Single Point	h6
SwapFirst2Last	Single Point	h7
AddRandomMixLimited	Single Point	h8
SwapRandomMixLimited	Single Point	h9
Remove	Single Point	h10
AddDiffOp	λ	h11
SwapDiffOp	λ	h12
AddEach-Choice	λ	h13
SwapEach-Choice	λ	h14
AddFirst2Last	λ	h15
SwapFirst2Last	λ	h16
AddRandomMixLimited	λ	h17
SwapRandomMixLimited	λ	h18
Remove	λ	h19

5.6.2 Criando FOMs

O procedimento de criação de FOMs, apresentado na Seção 5.5, é realizado utilizando a ferramenta MuJava [77]. Nessa ferramenta, para criar um FOM, a classe de um problema a ser mutado é fornecida como entrada com o conjunto de operadores a serem aplicados. Foram utilizados todos os operadores implementados pela ferramenta. A ferramenta MuJava foi alterada para, ao invés de gerar um arquivo com o *log* de cada classe mutada, gerar um arquivo de metadados com as informações de cada FOM. O processo assemelha-se ao realizado pelo trabalho de Omar et *al.* [101] que posteriormente foi desenvolvido em uma ferramenta denominada HOMAJ [103]. Após todos os metadados dos FOMs serem gerados os mesmos são armazenados em um repositório de FOMs o qual a abordagem irá utilizar. Esses FOMs, juntamente com os seus registros de metadados em arquivos, são então submetidos ao processo de otimização.

O processo de execução e avaliação dos mutantes gerados pela MuJava foi alterado do seu procedimento original, o qual quando ocorria *time-out*[2] de um mutante considerava que o mutante era morto por todos os casos de teste disponíveis. A alteração aplicada, além de otimizar o processo de teste anteriormente utilizado pela ferramenta MuJava a qual utilizava uma abordagem que demandava demasiado custos computacionais, avalia por quais casos de teste ocorre *time-out* e assim determinando corretamente por quais casos de teste o mutante é realmente morto.

5.6.3 Criando HOMs

O processo de geração de HOMs, descrito na Seção 5.4, é utilizado durante o processo de otimização da abordagem HG4HOM. Para esta implementação inicial o procedimento de criação de HOMs foi adotado o limite de ordem $l = 2$, ou seja, *Second Order Mutation* (SOM)

[2]Um período de tempo permitido para que um determinado programa possa ser executado.

- cada SOM é formado por duas mutações (par de FOMs). Dessa forma, para se criar um HOM são então selecionados dois FOMs, de acordo com uma estratégia de um operador de mutação, e então combinados. Para identificar como proceder com a união dos dois FOMs selecionados, necessita-se examinar os metadados dos FOMs. A partir da análise dos metadados é possível identificar como proceder a união, aplicando as mutações pontualmente de acordo com as informações dos arquivos de metadados. Posteriormente, o HOM é executado com o conjunto de casos de teste fornecido, e são registrados os casos de teste que matam o HOM em seu arquivo de metadados, bem como os seus FOMs constituintes. O arquivo de metadados dos HOMs, semelhante ao dos FOMs, é exemplificado no Código 5.2.

Devido à natureza estocástica da estratégia *RandomMix* uma restrição foi aplicada e denominada de *RandomMixLimited*, Algoritmo 5.7. A estratégia *RandomMixLimited* utiliza o mesmo procedimento de seleção aleatória de FOMs, porém quando o número de HOMs já gerados por essa estratégia for superior ao número de FOMs existentes a estratégia passa então a selecionar os FOMs de um HOM selecionado aleatoriamente dos HOMs já gerados.

Algoritmo 5.7: Estratégia de mutação *RandomMixLimited*

```
1  Entrada: F, l
2  Saída: FOMsSelecionados
3  Início
4      FOMsSelecionados ← ∅;
5      se Número de HOMs gerados pela estratégia < |F| então
6          se Não há mais FOMs a serem selecionados de F então
7              Reinicia a lista de FOMs disponíveis a serem selecionados;
8          fim se
9          n ← gera aleatoriamente um número entre 2 e l;
10         i ← 0;
11         enquanto i < n faça
12             f ← ∅;
13             se Não há mais FOMs a serem selecionados de F então
14                 Reinicia a lista de FOMs disponíveis a serem selecionados;
15                 Remove da lista de FOMs disponíveis os FOMs presentes em
                    FOMsSelecionados;
16             fim se
17             f ← seleciona aleatoriamente um mutante de F;
18             Adiciona f em FOMsSelecionados;
19             Marca o FOM f como já selecionado em F;
20             i++;
21         fim enquanto
22     senão
23         h^n ← seleciona aleatoriamente um HOM gerado pela estratégia;
24         FOMsSelecionados ← seleciona de h^n os seus FOMs constituintes;
25     fim se
26     retorna FOMsSelecionados;
27 Fim
```

Código 5.2: Exemplo de arquivo de metadados de um HOM

```
{
        "foms": {
                "AORB_4": {
                        "className": "Cal.java",
                        "equivalent": false,
                        "lineNumber": 14,
                        "mutantStatement": "day2 + day1",
                        "mutationScore": 0.1,
                        "name": "AORB_4",
                        "operator": "AORB",
                        "originalStatement": "day2 - day1",
                        "parent": "numDays = day2 - day1",
                        "path": ".\CalProgram \...\ AORB_4\Cal.java",
                        "testCases": [
                                "test2"
                        ]
                },
                "ROR_5": {
                        "className": "Cal.java",
                        "equivalent": false,
                        "lineNumber": 13,
                        "mutantStatement": "month2 != month1",
                        "mutationScore": 1.0,
                        "name": "ROR_5",
                        "operator": "ROR",
                        "originalStatement": "month2 == month1",
                        "parent": "if (month2 == month1) { [...] }",
                        "path": ".\CalProgram \...\ ROR_5\Cal.java",
                        "testCases": [
                                "test1",
                                "test2",
                                "test3",
                                "test4",
                                "test5",
                                "test6",
                                "test7",
                                "test8",
                                "test9",
                                "test10"
                        ]
                }
        },
        "equivalent": false,
        "mutationScore": 1.0,
        "name": "AORB_4_and_ROR_5",
        "path": ".\CalProgram \...\ AORB_4_and_ROR_5\Cal.java",
        "testCases": [
                "test1",
                "test2",
                "test3",
                "test4",
                "test5",
                "test6",
                "test7",
                "test8",
                "test9",
                "test10"
        ]
}
```

5.7 Considerações Finais

Este capítulo apresentou a abordagem proposta, *Hyper-Heuristic for Generation of Higher Order Mutation* (HG4HOM), uma hiper-heurística de seleção com aprendizado *on-line* que realiza a seleção de estratégias para a construção dos melhores conjuntos de mutantes de ordem superior através de otimização multi-objetivo. Além da apresentação da abordagem foram descritos os seus principais aspectos de implementação, avaliação de *fitness* e os operadores genéticos propostos.

Para essa abordagem é proposto o uso de dois algoritmos de otimização, NSGA-II [24] e SPEA2 [131], e de três objetivos como função de *fitness*. Os métodos de seleção definidos são o *Choice Function*, adaptado por Maashi et *al.* [79], o FRR-MAB [72] e a seleção aleatória. A avaliação da qualidade da aplicação das heurísticas de baixo nível é realizada através do conceito de dominância, como proposto por Guizzo et *al.* [48].

Aspectos de avaliação da estratégia proposta utilizando a implementação realizada são descritos no próximo capítulo.

Capítulo 6

Avaliação Experimental

Os procedimentos adotados para avaliar empiricamente a abordagem HG4HOM, bem como os resultados obtidos são descritos neste capítulo. A avaliação foi conduzida utilizando a implementação realizada no capítulo anterior, utilizando a ferramenta MuJava e mutantes de segunda ordem (SOMs). Os experimentos foram guiados de acordo com a seguinte questão de pesquisa relacionada à hipótese deste trabalho: "Como são os resultados produzidos pela abordagem HG4HOM quando comparados com os algoritmos multi-objetivos e estratégias tradicionais de geração de HOMs?".

Para responder a essa questão foram adotados os seguintes passos. Inicialmente, foi realizado o ajuste de parâmetros dos algoritmos multi-objetivos e da abordagem utilizando o *irace* [74][1]. Posteriormente, a abordagem HG4HOM foi avaliada utilizando o melhor algoritmo encontrado com a sua configuração adequada juntamente com os métodos de seleção: *Choice Function* (CF), FRR-MAB e aleatório (*Random*). Os resultados são avaliados utilizando indicadores de qualidade como Hipervolume, *Error Ratio*, *Euclidean Distance*, *Effect Size* e análise estatística com os testes de Friedman [39, 40] e Kruskal-Wallis [68].

Para uma melhor avaliação das soluções geradas de acordo com os objetivos definidos para a função de *fitness* e para observar a influência de cada um deles, foram realizados experimentos considerando separadamente três formulações, apresentadas na Tabela 6.1.

Tabela 6.1: Formulações para a função de *fitness*

Formulação	Descrição
2OWSD	A função de *fitness* é composta pelo objetivo 1 (Seção 5.2.1) que corresponde ao número de HOMs presentes em um indivíduo em relação ao número de FOMs e pelo objetivo 2 (Seção 5.2.2) que busca mensurar os HOMs que compõem um indivíduo em relação à capacidade em revelar defeitos entre cada HOM e seus FOMs constituintes
2OSSHOM	A função de *fitness* é composta pelo objetivo 1 (Seção 5.2.1) que corresponde ao número de HOMs presentes em um indivíduo em relação ao número de FOMs e pelo objetivo 3 (Seção 5.2.3) que visa a mensurar o conjunto de HOMs que compõem um indivíduo em relação à capacidade de um HOM substituir seus FOMs constituintes sem perder a eficácia do teste
3O	A função de *fitness* é composta pelo objetivo 1 (Seção 5.2.1) que corresponde ao número de HOMs presentes em um indivíduo em relação ao número de FOMs, pelo objetivo 2 (Seção 5.2.2) que busca mensurar os HOMs que compõem um indivíduo em relação à capacidade em revelar defeitos entre cada HOM e seus FOMs constituintes, e pelo objetivo 3 (Seção 5.2.3) que visa a mensurar o conjunto de HOMs que compõem um indivíduo em relação à capacidade de um HOM substituir seus FOMs constituintes sem perder a eficácia do teste

[1]Pacote de *software* que implementa procedimentos de configuração automática de algoritmos

6.1 Indicadores de Qualidade

A análise de desempenho dos algoritmos é realizada através de indicadores de qualidade. Muitos indicadores requerem uma Fronteira de Pareto (*Pareto Front*), porém essa fronteira ao se lidar com problemas do mundo real é desconhecida ou raramente existe. A estratégia mais comum utilizada é a construção de uma Fronteira de Pareto Ótima (PF_{true}), composta da união de todas as soluções obtidas por todos os k algoritmos em todas as suas n execuções realizadas [130].

Por essa razão foram definidos três conjuntos de soluções para a análise do desempenho dos algoritmos:

- PF_{approx}: conjunto de soluções não dominadas obtidas por uma execução do algoritmo;
- PF_{know}: conjunto de soluções obtidas com a união de todos os PF_{approx} obtidos através de n execuções de um algoritmo, removendo as soluções não dominadas e repetidas;
- PF_{true}: conjunto de soluções obtidas com a união de todos os PF_{know} obtidos com diferentes k algoritmos, removendo as soluções não dominadas e repetidas. Esse conjunto representa a aproximação da fronteira real.

Os resultados foram comparados utilizando como métrica principal o Hipervolume (HV) [132]. HV é a métrica de qualidade com o maior poder discriminatório dentre as métricas de qualidade conhecidas [133] e popularmente utilizada na comparação do desempenho de otimizadores multi-objetivos. Dada uma solução, a métrica de HV mede o tamanho da porção do espaço objetivo que é dominado por essa solução capturando em uma escalar ambos, a proximidade da solução para o conjunto ideal e sua propagação através do espaço objetivo. Essa métrica também tem propriedades matemáticas mais agradáveis do que outras métricas, sendo o único tipo de indicador que é estritamente monotônico[2]. Entretanto, o HV é muito sensível à escala dos objetivos e à presença de pontos extremos [6, 20, 108, 127]. Para se determinar os pontos de referência do HV são utilizados os valores máximos e mínimos encontrados de cada objetivo para uma determinada PF_{true} que corresponde à Fronteira de Pareto Ótima de um determinado problema/instância.

Além do HV, foi utilizado o valor de *Error Ratio* (ER) e *Euclidean Distance* (ED) na avaliação dos resultados. ER corresponde a proporção dos elementos do conjunto PF_{know} que estão presentes na PF_{true} [125], representado pela Equação 6.1.

$$ER(A) = \frac{\sum_{\alpha \in A^e}}{|A|} \tag{6.1}$$

onde, A representa o conjunto de soluções de PF_{know}, e e corresponde a um valor 0 quando a solução está presente na fronteira PF_{true} e 1 quando não está. O valor obtido pela equação varia entre 0 e 1, quanto menor o valor de ER maior é a quantidade de soluções de PF_{know} presentes em PF_{true}, dessa maneira, melhor é a fronteira em relação a quantidade de soluções ótimas encontradas.

ED (Equação 6.2) mensura a distância entre uma solução $P = (p_1, p_2, ..., p_n)$ e uma solução ideal $Q = (q_1, q_2, ..., q_n)$ num espaço de objetivos. A solução ideal possui os melhores valores possíveis em todos os objetivos do problema, por exemplo, considerando um problema de minimização de dois objetivos em que os valores encontram-se normalizados (entre 0 e 1) uma solução ideal seria o conjunto (0,0). Desse modo, essa métrica indica o quão próximo uma

[2]Quando uma aproximação do conjunto de Pareto domina outra, o valor indicador do primeiro será maior que o último.

solução está de ser ideal e para isso, quando menor o seu valor melhor é o *trade-off* de objetivos da solução.

$$ED = \sum_{i=1}^{n} (p_i - q_i)^2 \tag{6.2}$$

Após a aplicação das métricas acima descritas para avaliar o desempenho dos algoritmos, uma inferência estatística[3] deve ser aplicada para discernir se um conjunto de experimentos são significativamente diferentes em algum aspecto de outro [54]. Sendo que, os pesquisadores de SBSE tendem a confiar na utilização de inferência estatística como um meio de abordar a natureza inerentemente estocástica dos algoritmos baseados em busca [54].

Nesse campo estatístico pode ser empregado para análise de uma ou mais amostras o teste de hipótese [21]. Nesse teste são definidas duas hipóteses: i) a hipótese nula H_0: *status quo*[4], assume-se que não há nenhum efeito ou nenhuma diferença e; ii) a hipótese alternativa H_1: assume-se que há a presença de um efeito ou uma diferença (diferenças significativas entre algoritmos). O objetivo do teste de hipótese é rejeitar a hipótese nula e ao se aplicar um procedimento estatístico um nível de significância α é utilizado para determinar em qual nível a hipótese pode ser rejeitada. Além disso, é determinado o menor nível de significância para a rejeição de H_0 denominado *p-value*, quanto menor o *p-value* mais forte a evidência contra H_0. O valor de *p-value* determina se um teste de hipótese estatística é significativo ou não, indicando o quão significativo o resultado é: quando o *p-value* é menor que o valor de α adotado afirma-se que há uma significância estatística.

Outro elemento importante na análise dos resultados é a utilização de um teste estatístico[5] adequado. Para isso, é necessário determinar o tipo de distribuição dos dados (resultados) para a utilização mais adequada dos tipos de teste: paramétricos (distribuição normal) ou não não-paramétricos (distribuição anormal). Tipicamente em SBSE, os pesquisadores utilizam testes não paramétricos [54], levando em consideração a estocasticidade dos algoritmos baseados em busca. A identificação do tipo de distribuição dos dados é importante, pois a aplicação correta do tipo de teste mais indicado torna este mais poderoso em relação a outro. De acordo com Thode [123] um dos melhores testes para detectar a anormalidade dos dados é o Shapiro-Wilk W [120], o qual testa a hipótese nula de que a amostra vem de uma distribuição normal. Para o teste de Shapiro-Wilk W foi utilizada a função `shapiro.test` no R, o qual retorna um *p-value*, se o valor de *p-value* for menor que 0.05 significa que a distribuição é anormal. Adicionalmente, foi realizado um teste visual com o auxílio do gráfico Quantil-Quantil (Q-Q) da normal para a verificação da adequação de distribuição de frequência dos dados à uma distribuição de probabilidades. Nesse gráfico se a distribuição é normal os pontos tendem a formar uma linha semelhante a uma reta.

Posteriormente, o próximo passo foi a escolha do teste estatístico para avaliar as diferenças entre os algoritmos ou não, ou seja, determinar se existe um ou mais algoritmos cujo desempenho pode ser considerado como significativamente diferente. Desse modo, para avaliar o desempenho dos algoritmos quando comparados em múltiplos sistemas foi utilizado o teste de Friedman [39, 40] que utiliza os *rankings* dos dados ao invés de seu valor bruto. O teste de Friedman utilizado foi a função `friedmanTest` do pacote *"Statistical Comparison of Multiple Algorithms in Multiple Problems"* (scmamp) [16], o qual foi baseado na descrição realizada por Demšar [28]. Além disso, foi utilizado o teste de Iman e Davenport [59], derivado do teste de Friedman, com uma estatística menos conservadora e mais precisa, e por isso optou-se em

[3]Inferem-se propriedades (afirmações) de uma população inteira com base em uma amostra.

[4]A circunstância que está sendo testada.

[5]Todos os testes estatísticos foram aplicados utilizando a ferramenta R [122].

reportar o *p-value* obtido por esse teste. Posteriormente, caso a hipótese nula seja rejeitada (houve diferença estatística) foi utilizado o procedimento de *post-hoc* para identificar quais algoritmos realmente diferem entre si. Para isso, García e Herrera [41] propõem a utilização do procedimento de Bergmann-Hommel [9], o qual se destacou por ser mais eficiente.

De outro lado, para a comparação dos algoritmos em um determinado problema foi utilizado o Kruskal-Wallis [68]. Em seguida, o teste de Nemenyi [94] foi utilizado como *post-hoc* após o teste de Kruskal-Wallis, caso haja diferença estatística, e para isso foi utilizado a função `posthoc.kruskal.nemenyi.test` do pacote *"Pairwise Multiple Comparison of Mean Ranks Package"* (PMCMR) [109] do R, o qual roda múltiplas comparações, controlando a taxa de erro para estatísticas em ordem de classificação e por isso não necessita de ajuste de *p-value*. Além disso, foi utilizada a função `kruskalmc` do pacote *"pgirmess: Data Analysis in Ecology"* [43] do R que auxilia a determinar quais grupos são diferentes com as comparações em pares ajustadas adequadamente para comparações múltiplas. Caso tenham sido observadas diferenças superiores a um valor crítico os pares de grupos que estão sendo testados são considerados estatisticamente diferentes em um dado nível de significância.

Para medir a magnitude da diferença (*effect size*) para cada par de grupo de dados foi utilizada a métrica não paramétrica de Vargha and Delaney's ($\hat{A}_{12}$) [47, 126]. Essa métrica avalia uma magnitude que varia entre 0 e 1, sendo o valor da magnitude determinado pela probabilidade de um valor selecionado aleatoriamente da primeira amostra ser maior do que outro valor selecionado aleatoriamente da segunda amostra.

$$magnitude\ do\ effect\ size = \begin{cases} Insignificante & se\ \hat{A}_{12} < 0.56 \\ Pequeno & se\ 0.56 \leq \hat{A}_{12} < 0.64 \\ Médio & se\ 0.64 \leq \hat{A}_{12} < 0.71 \\ Grande & se\ \hat{A}_{12} \geq 0.71 \end{cases} \tag{6.3}$$

De acordo com a métrica $\hat{A}_{12}$ uma magnitude *Insignificante* não representa diferença estatística, pois há uma mínima diferença entre valores. As magnitudes *Pequeno* e *Médio* representam que pode haver ou não diferença estatística. Finalmente, a magnitude *Grande* representa que há uma grande diferença e que pode ser vista nos números sem muito esforço.

Todos os testes estatísticos utilizaram uma confiança de 95%, assim, caso seja apresentado um *p-value* inferior a 0.5 isso significa que há significância estatística.

6.2 Questões de Pesquisa

De modo a avaliar a abordagem proposta em várias perspectivas outras questões de pesquisa (QPs) foram derivadas da questão principal e são descritas a seguir.

QP1: *Qual algoritmo multi-objetivo é o melhor em relação aos objetivos propostos?* Para responder a essa questão foram comparados os algoritmos NSGA-II e SPEA2 em relação ao indicador de qualidade Hipervolume relativo às três formulações da função de *fitness* juntamente com o teste estatístico de Friedman para determinar o melhor algoritmo em relação a todos os sistemas a serem analisados.

QP2: *Quais são os resultados obtidos pelos métodos de seleção?* Esta questão avalia qual o melhor método de seleção: CF, FRR-MAB ou *Random*, em relação ao indicador de qualidade de

Hipervolume relativo às três formulações da função de *fitness*. Para analisar os valores obtidos é utilizado o teste estatístico de Kruskal-Wallis em cada sistema a ser avaliado.

QP3: *Como são os resultados do melhor algoritmo multi-objetivo quando comparados com a HG4HOM?* Essa questão visa a avaliar o melhor algoritmo multi-objetivo comparado aos resultados da abordagem HG4HOM em relação às três formulações da função de *fitness*. Para isso é comparado o valor de Hipervolume encontrado por cada cada algoritmo em cada formulação utilizando o teste estatístico de Kruskal-Wallis em cada sistema a ser avaliado.

QP4: *Como são os resultados da abordagem HG4HOM quando comparados às estratégias tradicionais de geração de HOMs em termos de escore de mutação e redução no número de casos de teste?* Uma vez determinado qual método de seleção e algoritmo ideal para a abordagem HG4HOM, essa é comparada com as estratégias tradicionais de geração de HOMs. Para isso, primeiramente foi determinado o número de casos de teste necessários para matar os HOMs em cada sistema para a abordagem em cada formulação e para cada estratégia. Posteriormente, os conjuntos de casos de teste determinados foram aplicados junto aos FOMs e calculado o escore de mutação correspondente. Por fim, a abordagem em cada formulação e cada estratégia foram comparadas considerando o conjunto de casos de teste necessários para os FOMs, bem como o escore de mutação obtido. Os valores foram analisados utilizando o teste estatístico de Friedman.

QP5: *Qual o desempenho da abordagem HG4HOM quando comparada às estratégias tradicionais de geração de HOMs em relação aos objetivos propostos?* Para responder essa questão, foram determinados valores de *fitness* para os HOMs gerados pelas estratégias tradicionais em relação às três formulações da função de *fitness*. Posteriormente, foram determinadas as melhores soluções em cada sistema pelas estratégias e pela abordagem para cada formulação, aquelas que obtiveram o menor valor de ED encontrado em relação ao conjunto PF_{true}. Para a situação de haver muitas soluções que possuam o valor de ED igual ao menor valor de ED encontrado foi considerada a primeira solução igual ao menor valor de ED. Em seguida, essas soluções são comparadas para avaliar o desempenho em encontrar HOMs que possam revelar defeitos e que possam substituir seus FOMs constituintes sem perder a eficácia do teste.

De modo a responder às questões de pesquisa foi definido um conjunto de sistemas (Seção 6.3). Posteriormente, foi utilizado um processo de configuração dos algoritmos multi-objetivos e também para os métodos de seleção CF e FRR-MAB.

6.3 Sistemas Utilizados

O estudo foi realizado com seis sistemas Java, já empregados na literatura [110, 111]. Eles foram selecionados para prover uma compatibilidade com os estudos previamente realizados e para facilitar a replicação do estudo. São eles: i) *Bisect*, sistema que calcula a raiz quadrada de *N* usando o método de bissetriz; ii) *Bub*, sistema que utiliza o procedimento de ordenação bolha; iii) *Find*, sistema que divide um vetor em dois outros segundo um valor de partição; iv) *Fourballs*, sistema que calcula o peso relativo de uma bola; v) *Mid*, sistema que calcula o valor central de três valores; e vi) *Triangle*, sistema que classifica o tipo de um triângulo de acordo com o comprimento de suas três arestas.

Para cada sistema estava disponível um conjunto de casos de teste inicial, os quais foram codificados no formato JUnit[6] e melhorados utilizando a ferramenta JavaCodeCoverage (JaCoCo) [73][7], formando assim o conjunto de testes T_i.

Inicialmente, a abordagem HG4HOM necessita que os FOMs sejam previamente gerados para serem aplicados no processo de otimização. Dessa forma, foi utilizada a ferramenta MuJava [77] e todos os seus operadores de mutação disponíveis. Esse procedimento gerou o conjunto de FOMs denominado MG. Posteriormente, o conjunto MG de cada sistema foi executado com o conjunto de teste T_i. Os mutantes vivos foram analisados e se possível novos casos de teste foram gerados. Nesse passo os mutantes equivalentes que não puderam ser mortos por nenhum caso de teste foram identificados manualmente, compondo o conjunto de mutantes equivalentes E. O conjunto M é composto de $MG - E$. Em seguida, foi definido o conjunto T, composto pelo conjunto T_i e os novos casos de teste gerados. Por fim, foi gerado o arquivo de metadados com a informação de cada FOM. Os conjuntos MG e T foram definidos como sendo, respectivamente, os conjuntos de FOMs e casos de teste a serem utilizados na abordagem. As informações de cada sistema são apresentadas na Tabela 6.2.

Tabela 6.2: Informações sobre os Sistemas

Sistema	**Linhas de Código**	$\|T\|$	$\|MG\|$	$\|M\|$	$\|E\|$
`bisect`	30	34	198	174	24
`bub`	40	261	172	144	28
`find`	56	144	299	277	22
`fourballs`	38	116	292	280	12
`mid`	46	136	317	257	60
`triangle`	77	259	605	526	79

6.4 Configuração dos Experimentos

Os algoritmos de otimização são projetados, em sua maioria, sem uma configuração específica, gerando soluções viáveis que melhor se aproximem das soluções ótimas. Segundo Eiben et *al.* [29], o desempenho de um algoritmo é determinado pela configuração de parâmetros do algoritmo o qual afeta diretamente a qualidade da solução final. Dado um tempo de execução adequado, uma determinada configuração de parâmetros para um algoritmo pode resultar em uma rápida convergência para boas soluções, porém isso impede de que o algoritmo explore adequadamente o espaço de busca para encontrar soluções melhores. Por outro lado, se dado um tempo de execução muito curto com capacidades de exploração mais elevados pode produzir resultados pobres. Dessa maneira, há a necessidade de um ajuste adequado de parâmetros do algoritmo.

Entretanto, a configuração de um algoritmo não é uma tarefa fácil uma vez que, além do espaço de soluções do problema sendo abordado, passa a existir o espaço de valores possíveis dos parâmetros de configuração do algoritmo [29]. Geralmente tais configurações são escolhidas com base na literatura [74] e devido a tal complexidade no processo de configuração de um algoritmo essa tarefa pode ser vista como um problema de configuração de algoritmo [75].

De modo a auxiliar na configuração de parâmetros há alguns métodos que auxiliam a configuração automática de parâmetros, dentre eles, o presente trabalho utiliza o *irace* (*Iterated Race for Automatic Algorithm Configuration*) [74]. *irace* é um pacote de *software* que implementa

[6]JUnit é um *framework* de teste de unidade para linguagem de sistemação Java.

[7]JaCoCo é uma ferramenta de análise bytecode para análise de cobertura de teste.

um número de procedimentos de configuração automática, dentre eles o procedimento *Iterated Racing* (Iterated F-race ou I/F-Race) que têm se mostrado uma abordagem promissora e vem sendo utilizada na literatura para a configuração automática de meta-heurísticas [30, 32]. Além disso, *irace* utiliza um mecanismo baseado em teste estatístico e têm mostrado bons resultados para algoritmos multi-objetivos [10, 11, 75]. Ao final da avaliação o *irace* retorna um conjunto de configurações do algoritmo que são estatisticamente equivalentes.

F-Race [12] é um procedimento de corrida (*racing*) [85] para seleção da melhor configuração dentre um determinado conjunto de configurações de algoritmo por meio da análise estatística utilizando o teste estatístico não paramétrico Friedman e seu teste de *post-hoc* associado [21]. Para que o teste estatístico possa ser aplicado um valor (custo) para as soluções geradas em relação ao seu desempenho deve ser atribuído. Para isso, o *irace* permite flexibilizar o método de avaliação das soluções geradas por cada configuração de modo que o usuário possa implementar ou utilizar a abordagem que melhor atenda ao seu problema. Dessa forma, neste trabalho as soluções geradas por cada configuração candidata foram avaliadas utilizando a métrica de qualidade de Hipervolume.

O espaço de configurações que o *irace* utiliza na avaliação de um determinado algoritmo necessita ser inicialmente definido através de um arquivo de parâmetros. O arquivo de parâmetros estabelece tipos, valores, intervalos e condições das possíveis configurações a serem geradas. Os tipos de parâmetros possíveis são: i) *c*, categório; ii) *i*, inteiro; iii) *o*, ordinal e iv) *r*, real. Os valores para os tipos *c* e *o* são determinados por um conjunto fixo, já os valores para *i* e *r* são determinados por um valor entre o valor máximo e o valor mínimo determinado. As condições representam determinadas regras de utilização de determinados parâmetros. Além disso, é necessária a definição do número máximo de configurações a serem geradas, denominado *budget*, e quais as instâncias (sistemas ou problemas) de treinamento em que o algoritmo será avaliado.

Dessa maneira, para realizar a configuração de parâmetros a serem utilizados no *irace* foram determinados parâmetros baseados e adaptados de outros experimentos conduzidos em trabalhos da literatura [4, 48, 84, 121] e são resumidos na Tabela 6.3.

No conjunto de parâmetros definido foi estabelecida uma variação da população (*Population Size*) entre 50 a 300 soluções, podendo tal população ser inicializada (*Initialization*) por alguma das quatro estratégias: *RandomMixLimited*, *FirstToLast*, *Different-Operators* e *Each-Choice* (Seções 5.3.2 e 5.6.3). Com o objetivo de reduzir o conjunto de FOMs e de modo a não gerar uma possível solução com tamanho excessivo foi definido que cada solução pode possuir um tamanho (*Hom Set Size*) entre 10% a 50% em relação ao número de FOMs existentes do sistema em teste. Para a etapa de seleção durante o procedimento de reprodução do algoritmo foram adotados os operadores (*Selection Operator*): *KTournament*, *Random*, *NaryTournament* e *Binary*; sendo que para o operador *KTournament* foi definido o tamanho de torneio (*Tournament Size*) possíveis: 2, 4, 6, 8 e 10. Os operadores de recombinação (*Crossover Operator*) são apenas duas opções, uma utilizando *SinglePoint* com probabilidade (*Crossover Probability*) entre 1 e 100% e outro sem a utilização de recombinação (λ). Os operadores de mutação (*Mutation Operator*) definidos são os operadores propostos na Seção 5.3.2: *AddFirst2Last*, *AddRandom*, *AddDiffOp*, *AddEach-Choice*, *SwapFirst2Last*, *SwapRandom*, *SwapDiffOp*, *SwapEach-Choice* e *Remove*; com probabilidade (*Mutation Probability*) entre 1 e 100% e outro sem a utilização de mutação (λ). Para cada configuração gerada pelo *irace* o algoritmo evolui as soluções por 2.000 avaliações de *fitness* (*Fitness Evaluations*).

Para garantir que a configuração de parâmetros não fosse afetada por *overtuning* foram utilizadas como instâncias de treinamento todos os sistemas definidos na Seção 6.2 em três formulações de funções de *fitness* distintas (Tabela 6.1), utilizando como configuração do *irace* os parâmetros definidos na Tabela 6.3. Para cada estratégia utilizada pelos parâmetros *Initialization*

Tabela 6.3: Parâmetros utilizados no *irace* para a configuração dos algoritmos multi-objetivos

Nome	Tipo	Valores	Condição
Algorithm	o	NSGA-II	
Population Size	i	[50, 300]	
Fitness Evaluations	c	2000	
Initialization	c	RandomMixLimited | FirstToLast | DifferentOperators | EachChoice	
Hom Set Size	r	[0.1, 0.5]	
Selection Operator	c	KTournament | Random | NaryTournament | Binary	
Tournament Size	c	[2, 4, 6, 8, 10]	Selection Operator == *KTournament*
Crossover Operator	c	λ | SinglePoint	
Crossover Probability	r	[0.01, 1]	Crossover Operator != λ
Mutation Operator	c	λ | AddDiffOp | SwapDiffOp | AddEachChoice | SwapEachChoice | AddFirst2Last | SwapFirst2Last | AddRandomMixLimited | SwapRandomMixLimited | Remove	
Mutation Probability	r	[0.01, 1]	Mutation Operator != λ
Budget	10.000		

e *Mutation Operator* (Tabela 6.3) foram adotados como parâmetros a serem utilizados no PCH (Seção 5.4): l igual a dois e F igual ao conjunto MG dos sistemas utilizados. Inicialmente a configuração foi realizada com o algoritmo NSGA-II e uma vez que o algoritmo foi configurado junto ao *irace* este gerou um conjunto de configurações para cada formulação, Tabela 6.4. Para uma melhor visualização o *RandomMixLimited* foi denominado como apenas *Random*.

Tabela 6.4: Melhores configurações encontradas pelo *irace* para a configuração dos algoritmos multi-objetivos

Formulação	Config.	Pop.	Init.	Hom Size	Selection	Crossover (Prob.)	Mutation (Prob.)
2OWSD	1	60	Random	0.25	Random	SinglePoint (0.94)	AddRandom (0.89)
	2	56	Random	0.41	Nary	SinglePoint (0.91)	SwapRandom (0.76)
	3	62	Random	0.39	Nary	SinglePoint (0.83)	SwapRandom (0.75)
	4	54	Random	0.28	Binary	SinglePoint (0.92)	AddRandom (0.96)
	5	58	Random	0.38	Nary	SinglePoint (0.81)	SwapRandom (0.73)
2OSSHOM	1	62	Random	0.48	Random	SinglePoint (0.97)	AddRandom (0.98)
	2	60	Random	0.49	Random	SinglePoint (0.99)	AddRandom (0.96)
	3	52	Random	0.46	Random	SinglePoint (0.95)	AddRandom (0.94)
	4	64	Random	0.48	Random	SinglePoint (0.97)	AddRandom (0.92)
	5	66	Random	0.49	Random	SinglePoint (0.96)	AddRandom (0.95)
3O	1	56	Random	0.47	KTournament (6)	SinglePoint (0.93)	SwapRandom (1.00)
	2	50	Random	0.47	KTournament (2)	SinglePoint (0.98)	SwapRandom (0.98)
	3	60	Random	0.42	KTournament (2)	SinglePoint (0.97)	SwapRandom (0.98)
	4	58	Random	0.43	KTournament (4)	SinglePoint (0.98)	SwapRandom (0.98)
	5	52	Random	0.36	KTournament (4)	SinglePoint (0.94)	AddRandom (0.96)

Em seguida, cada um dos conjuntos das melhores configurações encontradas foram executados 10 vezes em cada sistema e para identificar a distribuição dos dados foi utilizado o teste de Shapiro-Wilk W, cujo resultado é apresentado na Tabela 6.5 juntamente com os gráficos de densidade e Q-Q da normal, Figuras 6.1 a 6.3.

Tabela 6.5: Shapiro-Wilk W - configuração de parâmetros

Formulação	***p-value***
2OWSD	3.531e-04
2OSSHOM	1.045e-07
3O	9.91e-06

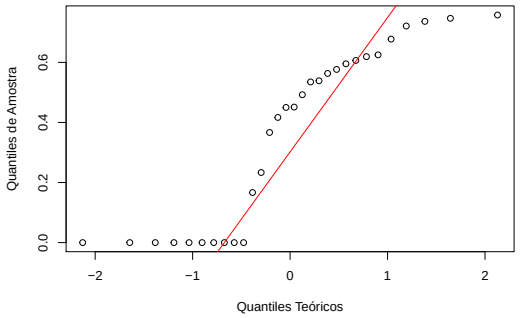


Figura 6.1: Q-Q da normal dos dados da formulação 2OWSD

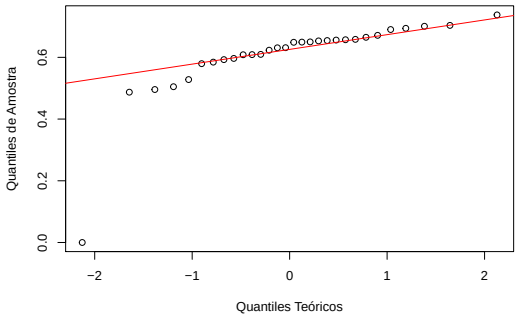


Figura 6.2: Q-Q da normal dos dados da formulação 2OSSHOM

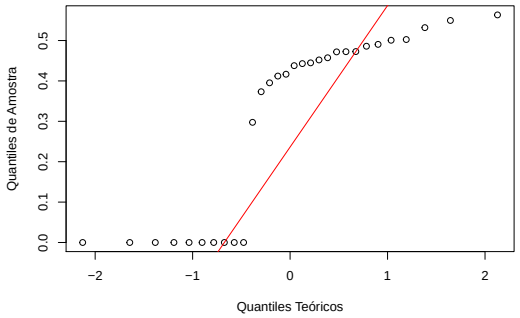


Figura 6.3: Q-Q da normal dos dados da formulação 3O

Uma vez identificada a anormalidade dos dados foi computada a média de HV de cada configuração, para cada sistema, em relação as 10 execuções independentes e aplicados os testes estatísticos de Friedman e Iman e Davenport para determinar a melhor configuração para cada tipo de formulação. O teste foi aplicado no conjunto de médias obtidas por cada configuração através de todos os sistemas, permitindo determinar a melhor configuração em geral. Na Tabela 6.6 são apresentadas as médias de HV e o desvio padrão imediatamente abaixo em parêntesis. Os melhores valores para cada sistema estão em negrito e o resultado do teste de Friedman é apresentado na Tabela 6.7.

De acordo com as Tabelas 6.6 e 6.7 observa-se que as melhores configurações encontradas para o NSGA-II obtiveram: (i) pequenos tamanhos de população; (ii) a predominância da estratégia *Random* em todas as formulações como estratégia para a inicialização da população; (iii) o tamanho de cada solução ficou entre 39% e 48% em relação à quantidade de FOMs existentes; (iv) não houve predominância de nenhum método de seleção; (v) a necessidade da utilização de um operador de recombinação; (vi) a predominância da estratégia *Random* nos operadores de mutação. As melhores configurações encontradas pelo algoritmo NSGA-II para as formulações 2OWSD, 2OSSHOM e 3O são apresentadas na Tabela 6.8.

Tabela 6.6: Média de Hipervolume (Desvio Padrão) das melhores configurações do NSGA-II

Sistema	Formulação	Configurações 1	2	3	4	5
bisect	2OWSD	0.451 (0.109)	0.595 (0.122)	**0.625 (0.17)**	0.563 (0.171)	0.607 (0.162)
	2OSSHOM	**0.610 (0.057)**	0.487 (0.078)	0.496 (0.034)	0.505 (0.057)	0.528 (0.051)
	3O	0.297 (0.0389)	**0.443 (0.077)**	0.395 (0.076)	0.417 (0.072)	0.373 (0.074)
bub	2OWSD	0.677 (0.145)	**0.758 (0.096)**	0.721 (0.717)	0.737 (0.132)	0.747 (0.14)
	2OSSHOM	0.650 (0.052)	**0.654 (0.071)**	0.631 (0.042)	0.623 (0.0507)	0.631 (0.047)
	3O	0.477 (0.048)	0.472 (0.043)	**0.49 (0.05)**	0.486 (0.058)	0.452 (0.085)
find	2OWSD	0.167 (0.157)	**0.45 (0.209)**	0.417 (0.196)	0.233 (0.086)	0.367 (0.153)
	2OSSHOM	0.596 (0.023)	**0.608 (0.027)**	0.580 (0.031)	0.584 (0.029)	0.593 (0.035)
	3O	0.55 (0.093)	**0.563 (0.095)**	0.532 (0.129)	0.501 (0.127)	0.502 (0.139)
fourballs	2OWSD	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)
	2OSSHOM	**0.658 (0.071)**	0.657 (0.074)	0.650 (0.064)	0.609 (0.043)	0.649 (0.061)
	3O	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)
mid	2OWSD	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)
	2OSSHOM	0.704 (0.171)	0.690 (0.061)	0.701 (0.095)	**0.738 (0.104)**	0.653 (0.250)
	3O	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)
triangle	2OWSD	0.539 (0.114)	0.535 (0.13)	**0.619 (0.067)**	0.576 (0.173)	0.492 (0.104)
	2OSSHOM	0.671 (0.071)	0.656 (0.114)	0.665 (0.096)	**0.694 (0.080)**	0.0 (0.0)
	3O	**0.472 (0.09)**	0.457 (0.104)	0.445 (0.075)	0.438 (0.118)	0.412 (0.087)
Média	2OWSD	0.306	0.390	**0.397**	0.352	0.369
	2OSSHOM	**0.648**	0.626	0.620	0.625	0.509
	3O	0.299	**0.323**	0.310	0.307	0.290

Tabela 6.7: Teste de Friedman e Iman e Davenport - configuração de parâmetros do NSGA-II

Formulação	*p-value* (Friedman)	*p-value* (Iman e Davenport)
2OWSD	0.3883	0.4112
2OSSHOM	0.1546	0.1459
3O	0.4244	0.4507

Tabela 6.8: Melhores configurações encontradas pelo *irace* para a configuração do NSGA-II

Formulação	Pop.	Init.	Hom Size	Selection	Crossover (Prob.)	Mutation (Prob.)
2OWSD	62	Random	0.39	Nary	SinglePoint (0.83)	SwapRandom (0.75)
2OSSHOM	62	Random	0.48	Random	SinglePoint (0.97)	AddRandom (0.98)
3O	50	Random	0.47	KTournament(2)	SinglePoint (0.98)	SwapRandom (0.98)

Em seguida as melhores configurações encontradas com o NSGA-II para as formulações 2OWSD, 2OSSHOM e 3O (Tabela 6.8) foram utilizadas no algoritmo SPEA2, o qual apenas necessita como parâmetro adicional o tamanho de arquivamento. Para isso foram estabelecidos quatro tamanhos diferentes de arquivamento em relação ao tamanho da população: i) 0: sem arquivamento; ii) 1, mesmo tamanho da população; iii) 1.5, 150% o tamanho da população; e iv) 2, o dobro do tamanho da população (200%). Cada uma dessas configurações de arquivamento foram denominadas de configurações 6, 7, 8 e 9 respectivamente. Essas configurações foram executados em cada sistema com 10 rodadas independentes. Posteriormente, com novos conjuntos de soluções encontradas pelo SPEA2 foi necessário determinar a nova PF_{true}, assim, os valores de HV foram recalculados para as melhores configurações do NSGA-II e computado o HV de cada configuração do SPEA2. Em seguida, foi realizada a comparação das configurações do SPEA2 em relação à melhor de cada NSGA-II para determinar o melhor algoritmo multi-objetivo para cada problema. Na Tabela 6.9 são apresentadas as médias de HV e o desvio padrão imediatamente abaixo em parêntesis. Os melhores valores para cada sistema estão em negrito

e em cinza os valores estatisticamente equivalentes ao melhor valor de acordo com o teste de Friedman apresentado na Tabela 6.10 e seus *post-hocs* cujo os resultados estão na Tabela 6.11.

Tabela 6.9: Média de Hipervolume (Desvio Padrão) das melhores configurações dos algoritmos multi-objetivos

Sistema	**Formulação**	**Configurações**				
		3	**6**	**7**	**8**	**9**
bisect	2OWSD	0.028 (0.008)	0.035 (0.003)	0.057 (0.002)	0.566 (0.006)	**0.572 (0.007)**
	2OSSHOM	0.367 (0.034)	**0.722 (0.050)**	0.583 (0.030)	0.577 (0.032)	0.562 (0.034)
	3O	0.020 (0.005)	0.029 (0.002)	0.051 (0.005)	0.283 (0.020)	**0.298** (0.020)
bub	2OWSD	0.042 (0.005)	0.080 (0.003)	0.021 (0.002)	0.496 (0.002)	**0.497 (9.68e-4)**
	2OSSHOM	0.491 (0.039)	**0.755 (0.052)**	0.754 (0.029)	0.753 (0.037)	0.748 (0.032)
	3O	0.032 (0.003)	0.065 (0.008)	0.064 (0.005)	**0.346 (0.021)**	0.332 (0.021)
find	2OWSD	0.005 (0.002)	0.006 (0.0)	0.000 (0.0)	0.578 (0.009)	**0.581 (0.009)**
	2OSSHOM	0.539 (0.020)	0.774 (0.051)	0.767 (0.040)	0.768 (0.036)	**0.798 (0.020)**
	3O	0.002 (3.97e-4)	0.004 (2.82e-4)	0.004 (3.63e-4)	**0.242 (0.010)**	0.235 (0.009)
fourballs	2OWSD	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	**0.603 (0.008)**	0.602 (0.012)
	2OSSHOM	0.470 (0.050)	**0.806 (0.049)**	0.768 (0.037)	0.805 (0.046)	0.774 (0.029)
	3O	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	**0.291 (0.010)**	0.283 (0.012)
mid	2OWSD	0.0 (0.0)	0.0 (0.0)	0.212 (0.342)	0.697 (0.012)	**0.700 (0.014)**
	2OSSHOM	0.387 (0.094)	0.837 (0.049)	0.841 (0.039)	0.854 (0.054)	**0.855 (0.049)**
	3O	0.0 (0.0)	0.0 (0.0)	0.259 (0.182)	0.354 (0.031)	**0.403 (0.028)**
triangle	2OWSD	0.006 (7.09e-4)	0.000 (0.0)	0.212 (0.008)	**0.588 (0.011)**	0.587 (0.007)
	2OSSHOM	0.460 (0.048)	0.789 (0.075)	**0.813 (0.049)**	0.807 (0.070)	0.786 (0.061)
	3O	0.04 (9.39e-4)	0.011 (0.001)	0.366 (0.025)	**0.378 (0.023)**	0.371 (0.028)
Média	2OWSD	0.014	0.020	0.147	0.588	**0.590**
	2OSSHOM	0.453	**0.780**	0.754	0.761	0.754
	3O	0.010	0.018	0.124	0.316	**0.320**

Tabela 6.10: Teste de Friedman e Iman e Davenport - configuração de parâmetros

Formulação	***p-value* (Friedman)**	***p-value* (Iman e Davenport)**
2OWSD	0.0046	0.0003
2OSSHOM	0.0116	0.0028
3O	0.0003	3.435e-08

Tabela 6.11: *Post-hoc* - configuração de parâmetros

Formulação	**Configurações**					
		3	6	7	8	9
2OWSD	3	n/a	1.000	1.000	**0.049**	**0.026**
	6	1.000	n/a	1.000	0.067	**0.049**
	7	1.000	1.000	n/a	0.136	0.114
	8	**0.049**	0.067	0.136	n/a	1.000
	9	**0.026**	**0.049**	0.114	1.000	n/a
		1	6	7	8	9
2OSSHOM	1	n/a	**0.010**	**0.042**	**0.037**	0.070
	6	**0.010**	n/a	1.000	1.000	1.000
	7	**0.042**	1.000	n/a	1.000	1.000
	8	**0.037**	1.000	1.000	n/a	1.000
	9	0.070	1.000	1.000	1.000	n/a
		2	6	7	8	9
3O	2	n/a	0.723	0.483	**0.002**	**0.004**
	6	0.723	n/a	1.000	**0.028**	**0.041**
	7	0.483	1.000	n/a	0.114	0.136
	8	**0.002**	**0.028**	0.114	n/a	1.000
	9	**0.004**	**0.041**	0.136	1.000	n/a

De acordo com a Tabela 6.9 o algoritmo SPEA2 obteve os melhores resultados em relação ao NSGA-II em todos os sistemas avaliados para todas as formulações. Dentre as configurações do SPEA2 as melhores configurações foram a 9 (dobro de arquivamento) para as formulações 2OWSD e 3O, e a configuração 6 (sem arquivamento) para a formulação 2OSSHOM. As melhores configurações encontradas para a configuração dos algoritmos multi-objetivos são apresentadas na Tabela 6.12, bem como o melhor algoritmo multi-objetivo encontrado.

Tabela 6.12: Melhores configurações encontradas para a configuração dos algoritmos multi-objetivos

Form.	Alg.	Pop.	Init.	Hom Size	Selection	Crossover (Prob.)	Mutation (Prob.)	Arq.
2OWSD	SPEA2	62	Random	0.39	Nary	SinglePoint (0.83)	SwapRandom (0.75)	2
2OSSHOM	SPEA2	62	Random	0.48	Random	SinglePoint (0.97)	AddRandom (0.98)	0
3O	SPEA2	50	Random	0.47	KTournament(2)	SinglePoint (0.98)	SwapRandom (0.98)	2

Uma vez definida a melhor configuração para os algoritmos multi-objetivos foi estabelecido um conjunto de parâmetros para a configuração da abordagem HG4HOM que utiliza hiper-heurística e que estão sumarizados nas Tabelas 6.13 e 6.14, respectivamente para os métodos de seleção CF e FRR-MAB. O método de seleção *Random* não foi configurado, pois não necessita de configuração adicional. Para estabelecer uma comparação justa, os parâmetros em comum com a melhor configuração para os algoritmos multi-objetivos foram preservados. Nessas tabelas foram omitidos os valores de *Population Size*, *Hom Set Size* e *Selection Operator*, pois assumem os mesmos valores encontrados pelas melhores configurações para cada uma das funções de *fitness* (Tabela 6.12). Os valores para a configuração dos métodos de seleção se concentraram entre 0 e 1, com exceção do parâmetro para a *Sliding Window* do FRR-MAB, para o qual foi determinado um intervalo semelhante ao fornecido ao tamanho da população na configuração inicial dos algoritmos multi-objetivos. O algoritmo utilizado para a configuração dos parâmetros da hiper-heurística foi o NSGA-II, assim como ocorreu com a configuração dos algoritmos multi-objetivos.

Tabela 6.13: Parâmetros utilizados no *irace* para a configuração da hiper-heurística com o método de seleção CF

Nome	Tipo	Valores
Algorithm	o	NSGA-II
Fitness Evaluations	c	2000
Initialization	c	Random
Selection Method	c	CF
α	r	[0,1]
β	r	[0,1]
Budget	10.000	

Tabela 6.14: Parâmetros utilizados no *irace* para a configuração da hiper-heurística com o método de seleção FRR-MAB

Nome	Tipo	Valores
Algorithm	o	NSGA-II
Fitness Evaluations	c	2000
Initialization	c	Random
Selection Method	c	FRR-MAB
C	r	[0,1]
Decayed Factor	r	[0,1]
Sliding Window	i	[50,300]
Budget	10.000	

Nas Tabelas 6.15 e 6.16 são apresentados os conjuntos de configurações que o *irace* gerou de cada formulação para cada método de seleção. Na Tabela 6.16 os parâmetros *Decayed Factor* e *Sliding Window* são representados pelos acrônimos DF e SW, respectivamente.

Essas configurações selecionadas pelo *irace* foram executadas em cada sistema com 10 rodadas independentes. Posteriormente, foi gerada a nova PF_{true} com as novas fronteiras encontradas por essas configurações e foram computadas as médias de HV. Em seguida, foi

Tabela 6.15: Melhores configurações encontradas pelo *irace* para a configuração do método de seleção CF

Formulação		Configurações 1	2	3
2OWSD	α	0.08626	0.13443	0.13351
	β	0.86939	0.79642	0.79642
2OSSHOM	α	0.04093	0.518	0.04848
	β	0.79557	0.42819	0.42819
3O	α	0.68700	0.68380	0.68618
	β	0.25880	0.25971	0.26076

Tabela 6.16: Melhores configurações encontradas pelo *irace* para a configuração do método de seleção FRR-MAB

Formulação		Configurações 1	2	3
2OWSD	*C*	0.47382	0.47894	0.46707
	DF	0.95858	0.91096	0.90911
	SW	52	56	50
2OSSHOM	*C*	0.03538	0.00383	0.93638
	DF	0.15738	0.16032	0.61437
	SW	76	73	62
3O	*C*	0.18814	0.18320	0.18580
	DF	0.42964	0.41322	0.41932
	SW	68	64	65

realizada a comparação das configurações para determinar a melhor configuração de cada método de seleção em cada problema. Nas Tabelas 6.17 e 6.18 são apresentadas as médias de HV e o desvio padrão imediatamente abaixo em parêntesis. Os melhores valores para cada sistema estão em negrito e em cinza os valores estatisticamente equivalentes ao melhor valor de acordo com o teste de Friedman, cujos resultados são apresentados nas Tabelas 6.19 e 6.20, além do resultado do *post-hoc* apresentado na Tabela 6.21.

Tabela 6.17: Média de Hipervolume (Desvio Padrão) das melhores configurações do método de seleção CF

Sistema	Formulação	Configurações 1	2	3
`bisect`	2OWSD	0.216 (0.027)	0.218 (0.021)	**0.222 (0.023)**
	2OSSHOM	**0.234 (0.048)**	0.198 (0.037)	0.206 (0.043)
	3O	0.112 (0.019)	0.107 (0.017)	**0.117 (0.026)**
`bub`	2OWSD	**0.294 (0.024)**	0.285 (0.031)	0.286 (0.020)
	2OSSHOM	0.421 (0.054)	0.421 (0.056)	**0.434 (0.047)**
	3O	0.214 (0.023)	**0.218 (0.04)**	0.2 (0.028)
`find`	2OWSD	0.290 (0.024)	**0.300 (0.016)**	0.297 (0.029)
	2OSSHOM	**0.349 (0.044)**	0.346 (0.018)	0.343 (0.064)
	3O	0.106 (0.02)	**0.111 (0.02)**	0.109 (0.012)
`fourballs`	2OWSD	**0.296 (0.025)**	0.284 (0.030)	0.287 (0.026)
	2OSSHOM	0.327 (0.049)	**0.341 (0.042)**	0.316 (0.048)
	3O	**0.133 (0.013)**	**0.133 (0.02)**	0.121 (0.016)
`mid`	2OWSD	0.170 (0.024)	**0.175 (0.028)**	0.171 (0.032)
	2OSSHOM	0.235 (0.075)	0.228 (0.044)	**0.239 (0.056)**
	3O	**0.07 (0.02)**	0.068 (0.017)	0.064 (0.021)
`triangle`	2OWSD	**0.273 (0.030)**	0.267 (0.026)	0.260 (0.034)
	2OSSHOM	0.301 (0.044)	0.286 (0.062)	**0.314 (0.035)**
	3O	0.129 (0.015)	**0.137 (0.02)**	0.132 (0.014)
Média	2OWSD	**0.256**	0.255	0.254
	2OSSHOM	**0.311**	0.303	0.309
	3O	0.127	**0.129**	0.124

De acordo com a Tabela 6.17 a configuração 1 obteve os melhores resultados para as formulações 2OWSD e 2OSSHOM, enquanto a configuração 2 obteve o melhor resultado para a formulação 3O. Os valores obtidos por essas configurações não apresentaram equivalência estatística em relação às outras configurações, desse modo, essas configurações foram selecionadas para o método de seleção CF e são apresentadas na Tabela 6.22.

Em relação ao método de seleção FRR-MAB (Tabela 6.18), a configuração 3 obteve o melhor valor na formulação 2OWSD e a configuração 1 para as formulações 2OSSHOM e 3O.

Tabela 6.18: Média de Hipervolume (Desvio Padrão) das melhores configurações do método de seleção FRR-MAB

Sistema	Formulação	Configurações 1	2	3
bisect	2OWSD	0.219 (0.021)	0.216 (0.018)	**0.220 (0.034)**
	2OSSHOM	0.216 (0.033)	0.216 (0.034)	**0.227 (0.035)**
	3O	**0.108 (0.018)**	0.106 (0.021)	0.104 (0.018)
bub	2OWSD	0.283 (0.022)	0.276 (0.025)	**0.289 (0.029)**
	2OSSHOM	**0.407 (0.065)**	0.399 (0.058)	0.404 (0.034)
	3O	**0.21 (0.031)**	0.189 (0.015)	0.2 (0.027)
find	2OWSD	**0.298 (0.022)**	0.282 (0.024)	0.287 (0.013)
	2OSSHOM	**0.356 (0.035)**	0.340 (0.039)	0.327 (0.045)
	3O	0.103 (0.01)	0.105 (0.012)	**0.106 (0.015)**
fourballs	2OWSD	0.278 (0.039)	0.289 (0.020)	**0.290 (0.047)**
	2OSSHOM	**0.343 (0.049)**	0.321 (0.046)	0.314 (0.044)
	3O	**0.125 (0.007)**	0.115 (0.023)	0.118 (0.013)
mid	2OWSD	0.167 (0.027)	0.162 (0.023)	**0.182 (0.012)**
	2OSSHOM	0.238 (0.058)	0.203 (0.044)	**0.261 (0.065)**
	3O	0.065 (0.01)	0.065 (0.021)	**0.068 (0.018)**
triangle	2OWSD	0.260 (0.030)	0.238 (0.026)	**0.267 (0.031)**
	2OSSHOM	**0.294 (0.044)**	**0.294 (0.027)**	0.260 (0.051)
	3O	0.118 (0.02)	**0.134 (0.011)**	0.132 (0.015)
Média	2OWSD	0.251	0.244	**0.256**
	2OSSHOM	**0.309**	0.296	0.299
	3O	**0.122**	0.119	0.121

Tabela 6.19: Teste de Friedman e Iman e Davenport - configuração de parâmetros do método de seleção CF

Formulação	*p-value* (Friedman)	*p-value* (Iman e Davenport)
2OWSD	1	1
2OSSHOM	0.3114	0.3392
3O	0.3114	0.3392

Tabela 6.20: Teste de Friedman e Iman e Davenport - configuração de parâmetros do método de seleção FRR-MAB

Formulação	*p-value* (Friedman)	*p-value* (Iman e Davenport)
2OWSD	0.0155	0.002663
2OSSHOM	0.3114	0.3392
3O	0.8465	0.08686

Tabela 6.21: *Post-hoc* - configuração de parâmetros na formulação 2OWSD para o método de seleção FRR-MAB

	Configurações 1	2	3
1	n/a	0.149	0.149
2	0.149	n/a	**0.012**
3	0.149	**0.012**	n/a

Na formulação 2OWSD a configuração 3 obteve equivalência estatística com a configuração 1, porém foi selecionado a que obteve o maior valor de HV. As configurações selecionadas para o método de seleção FRR-MAB são apresentadas na Tabela 6.23.

Após as configurações dos algoritmos multi-objetivos e da hiper-heurística as melhores configurações (Tabelas 6.12, 6.22 e 6.23) foram executadas e seus resultados são apresentados na Seção 6.5.

Tabela 6.22: Melhores configurações encontradas para a configuração do método de seleção CF

Formulação	α	β
2OWSD	0.08626	0.86939
2OSSHOM	0.04093	0.79557
3O	0.68380	0.25971

Tabela 6.23: Melhores configurações encontradas para a configuração do método de seleção FRR-MAB

Formulação	C	DF	SW
2OWSD	0.46707	0.90911	50
2OSSHOM	0.03538	0.15738	76
3O	0.18814	0.42964	68

6.5 Resultados

Na literatura, os experimentos são repetidos de 30 a 50 vezes [54], dessa forma, os experimentos foram repetidos 30 vezes com 60.000 avaliações de *fitness* em cada rodada em cada sistema definido na Seção 6.2 para cada formulação da função de *fitness*. Os experimentos foram realizados com as melhores configurações encontradas durante a fase de configuração dos experimentos (Seção 6.4) para o melhor algoritmo multi-objetivo (Tabela 6.12), com as melhores configurações para os métodos de seleção CF e FRR-MAB (Tabelas 6.22 e 6.23), e com o método de seleção *Random*. Posteriormente, foi gerada uma PF_{true} com as fronteiras encontradas por esses experimentos e computada as médias de HV. Em seguida, foi realizado o teste de Kruskal-Wallis para determinar o melhor experimento para cada formulação em cada sistema. Não foi possível calcular o tempo de execução dos algoritmos, pois os experimentos foram executados em computadores diferentes.

Na Tabela 6.24 são apresentadas as médias de HV e o desvio padrão imediatamente ao lado em parêntesis, além do *p-value* obtido pelo Kruskal-Wallis na última coluna. Os melhores valores para cada sistema estão em negrito e em cinza os valores estatisticamente equivalentes ao melhor valor de acordo com os testes de *post-hoc* apresentados nas Tabelas 6.25 a 6.27.

Tabela 6.24: Média de Hipervolume (Desvio Padrão) das formulações

Sistema	Formulação	SPEA2	Random	CF	FRR-MAB	*p-value*
`bisect`	2OWSD	0.258 (0.007)	0.592 (0.046)	0.651 (0.040)	**0.664 (0.011)**	2.72e-21
	2OSSHOM	0.085 (0.004)	0.190 (0.008)	0.185 (0.010)	**0.540 (0.077)**	8.24e-22
	3O	0.066 (0.034)	0.071 (0.004)	0.243 (0.191)	**0.376 (0.199)**	2.06e-17
`bub`	2OWSD	0.241 (0.003)	0.496 (0.059)	**0.561 (0.026)**	0.555 (0.045)	1.92e-18
	2OSSHOM	0.260 (0.004)	0.447 (0.051)	**0.502 (0.034)**	0.501 (0.012)	6.29e-18
	3O	0.066 (0.003)	0.22 (0.062)	**0.313 (0.052)**	0.302 (0.059)	9.39e-19
`find`	2OWSD	0.235 (0.005)	0.490 (0.015)	0.552 (0.017)	**0.554 (0.006)**	1.92e-21
	2OSSHOM	0.251 (0.003)	0.0495 (0.013)	**0.524 (0.006)**	0.520 (0.007)	8.92e-22
	3O	0.066 (0.004)	0.289 (0.026)	**0.353 (0.009)**	0.35 (0.008)	1.05e-21
`fourballs`	2OWSD	0.245 (0.006)	0.543 (0.0)	0.574 (0.017)	**0.583 (0.011)**	2.17e-20
	2OSSHOM	0.254 (0.003)	0.517 (0.028)	**0.536 (0.007)**	0.532 (0.010)	3.58e-18
	3O	0.068 (0.002)	0.317 (0.021)	**0.35 (0.009)**	0.318 (0.024)	5.75e-21
`mid`	2OWSD	0.289 (0.004)	0.474 (0.164)	0.583 (0.107)	**0.605 (0.065)**	8.55e-14
	2OSSHOM	0.283 (0.004)	0.550 (0.111)	**0.595 (0.067)**	0.563 (0.083)	8.96e-16
	3O	0.092 (0.003)	0.147 (0.138)	**0.229 (0.171)**	0.171 (0.146)	0.030
`triangle`	2OWSD	0.252 (0.004)	0.522 (0.019)	0.544 (0.022)	**0.549 (0.006)**	3.05e-20
	2OSSHOM	0.263 (0.004)	0.505 (0.005)	**0.523 (0.008)**	0.505 (0.006)	2.08e-20
	3O	0.078 (0.003)	0.316 (0.009)	**0.347 (0.007)**	0.317 (0.01)	1.18e-21

De acordo com os resultados apresentados na Tabela 6.24 observa-se que os métodos de seleção CF e FRR-MAB obtiveram melhores resultados em relação aos outros algoritmos, exceto no sistema `mid` na formulação 3O onde todos os algoritmos foram estatisticamente equivalentes. O método de seleção CF obteve os melhores resultados ou estatisticamente equivalentes ao melhor em todos os sistemas com exceção do sistema `bisect` na formulação 2OSSHOM e `mid`

na formulação 2WSD. Por outro lado, o FRR-MAB também obteve os melhores resultados ou estatisticamente equivalentes ao melhor em todos os sistemas com exceção do sistema `mid` na formulação 2OSSHOM, no sistema `fourballs` na formulação 3O e no sistema `triangle` nas formulações 2OSSHOM e 3O. Desse modo, consta-se que o CF não obteve o melhor valor ou estatisticamente equivalente ao melhor valor em duas ocasiões, ao contrário do FRR-MAB que não obteve em quatro ocasiões. Assim, esses métodos de seleção são melhores do que uma abordagem que utiliza procedimentos aleatórios, como o algoritmo multi-objetivo (de acordo com a configuração de parâmetros) e o método de seleção *Random*. Desse modo, a utilização adequada das LLHs pode selecionar determinados mutantes para contribuir com a solução que estão presentes em algumas estratégias determinísticas e sendo que as LLHs podem ser mais facilmente selecionados do que através de uma estratégia estocástica.

Nas formulações com dois objetivos, observa-se que os valores de HV obtidos são próximos, porém ao adicionar um terceiro objetivo os valores de HV caem drasticamente, o que é esperado devido ao aumento na dificuldade da busca.

Os resultados do teste de *post-hoc* realizados são apresentados nas tabelas a seguir as quais apresentam os algoritmos que estão sendo comparados (Coluna 1), o valor de *p-value* obtido no *post-hoc* (Coluna 2), o valor de diferença observado (Coluna 3), a diferença crítica (Coluna 4), se houve diferença estatística (Coluna 5), o valor de *effect size* obtido (Coluna 6) e o tamanho da magnitude do *effect size* (Coluna 7). A diferença crítica representa o valor máximo quando comparado com a diferença observada para que a comparação apresente diferença estatística, ou seja, caso o valor de diferença observada seja superior ao valor de diferença crítica infere-se que há diferença estatística. A informação de que há ou não diferença estatística é apresentada na coluna 5 como TRUE e caso contrário FALSE.

Na Tabela 6.25 são apresentados os resultados obtidos pelo *post-hoc* na formulação 2OWSD. Pode-se observar a existência de diferença estatística insignificante ($\hat{A}_{12} < 0.56$) entre o CF e o FRR-MAB e uma diferença grande ($\hat{A}_{12} \geq 0.71$) deles para os outros algoritmos em todos os sistemas.

Em seguida, na Tabela 6.26 são apresentados os resultados obtidos pelo *post-hoc* na formulação 2OSSHOM. Nessa tabela observa-se que o CF e o FRR-MAB obtiveram uma diferença grande entre eles apenas nos sistemas `bisect`, `mid` e `triangle`. No sistema `bisect` é interessante destacar que o valor de *effect size* obtido entre o CF e o FRR-MAB apresenta-se baixo, porém ao realizar a comparação do FRR-MAB com o CF o valor de *effect size* altera-se para 1, havendo assim uma diferença estatística grande. É possível visualizar essa comparação através da coluna da diferença observada que é superior ao da diferença crítica e destacada através da coluna diferença.

Em relação aos outros algoritmos, o CF e o FRR-MAB obtiveram uma diferença grande em todos os sistemas, com exceção do sistema `bisect`, na qual o CF obteve piores resultados quando comparado ao FRR-MAB e assim sendo estatisticamente equivalente ao *Random*, além disso nos sistemas `mid` e `triangle` o FRR-MAB obteve resultados próximos ao *Random*, sendo ambos estatisticamente equivalentes.

Por fim, na Tabela 6.27 são apresentados os resultados obtidos pelo *post-hoc* na formulação 3O. O método de seleção CF obteve uma pequena diferença em relação ao FRR-MAB, exceto nos sistemas `fourballs` e `triangle` no qual a diferença é grande. Já no sistema `mid` todas as comparações indicam uma diferença inferior à grande, sendo a grande maioria com uma diferença insignificante.

Em seguida, foi analisado o conjunto de soluções obtidas por cada algoritmo. Na Tabela 6.28 é apresentado o número de soluções obtidas na PF_{true}, o número de soluções na PF_{know} obtido por cada algoritmo juntamente com o número de soluções da PF_{know} que estão presentes

Tabela 6.25: *Post-hoc* - Média de Hipervolume (Desvio Padrão) da formulação 2OWSD

Sistema	Comparação	*p-value*	Dif. Obs.	Dif. Crit.	Diferença	*Effect Size*	Magnitude
bisect	CF / FRR-MAB	0.851	8	23.695	FALSE	0.384	insignificante
	CF / Random	0.001	39.4	23.695	TRUE	0.964	grande
	CF / SPEA2	2.43e-04	70.47	23.695	TRUE	1	grande
	FRR-MAB / Random	3.90e-06	47.4	23.695	TRUE	1	grande
	FRR-MAB / SPEA2	2.22e-16	78.47	23.695	TRUE	1	grande
	Random / SPEA2	0.008	31.07	23.695	TRUE	1	grande
bub	CF / FRR-MAB	1	0.467	23.695	FALSE	0.459	insignificante
	CF / Random	0.003	33.367	23.695	TRUE	0.907	grande
	CF / SPEA2	1.78e-13	70.967	23.695	TRUE	1	grande
	FRR-MAB / Random	0.003	33.834	23.695	TRUE	0.84	grande
	FRR-MAB / SPEA2	1.18e-13	71.434	23.695	TRUE	1	grande
	Random / SPEA2	0.001	37.6	23.695	TRUE	1	grande
find	CF / FRR-MAB	0.997	2	23.695	FALSE	0.54	insignificante
	CF / Random	1.18e-05	45.4	23.695	TRUE	0.987	grande
	CF / SPEA2	2.33e-15	75.8	23.695	TRUE	1	grande
	FRR-MAB / Random	3.41e-05	43.4	23.695	TRUE	1	grande
	FRR-MAB / SPEA2	1.45e-14	73.8	23.695	TRUE	1	grande
	Random / SPEA2	0.01	30.4	23.695	TRUE	1	grande
fourballs	CF / FRR-MAB	0.448	14.63	23.695	FALSE	0.299	insignificante
	CF / Random	0.006	31.63	23.695	TRUE	0.89	grande
	CF / SPEA2	1.47e-11	65.67	23.695	TRUE	1	grande
	FRR-MAB / Random	7.36e-06	46.27	23.695	TRUE	0.976	grande
	FRR-MAB / SPEA2	< 2e-16	80.30	23.695	TRUE	1	grande
	Random / SPEA2	0.003	34.03	23.695	TRUE	1	grande
mid	CF / FRR-MAB	1	1.23	23.695	FALSE	0.523	insignificante
	CF / Random	5.78e-05	42.37	23.695	TRUE	0.861	grande
	CF / SPEA2	1.76e-08	56.07	23.695	TRUE	0.926	grande
	FRR-MAB / Random	3.08e-05	43.60	23.695	TRUE	0.908	grande
	FRR-MAB / SPEA2	7.56e-09	57.30	23.695	TRUE	0.967	grande
	Random / SPEA2	0.51	13.70	23.695	FALSE	0.667	médio
triangle	CF / FRR-MAB	1	0.17	23.695	FALSE	0.538	insignificante
	CF / Random	1.54e-04	40.37	23.695	TRUE	0.909	grande
	CF / SPEA2	2.08e-14	73.40	23.695	TRUE	1	grande
	FRR-MAB / Random	1.42e-04	40.53	23.695	TRUE	0.99	grande
	FRR-MAB / SPEA2	1.79e-14	73.57	23.695	TRUE	1	grande
	Random / SPEA2	0.004	33.03	23.695	TRUE	1	grande

na PF_{true} entre parênteses e o valor de ER ao lado. Os valores em negrito correspondem aos melhores valores de ER, os quais possuem mais soluções (PF_{know}) presentes na PF_{true}.

Ao analisar os valores de ER constata-se que o algoritmo SPEA2, sem a utilização do conceito de hiper-heurística, obteve, em sua maioria, os melhores valores de ER, ou seja, uma maior quantidade de soluções na PF_{true}. Dos dezoito casos observados o algoritmo SPEA2 obteve os melhores valores de ER em doze deles, seguido do método de seleção CF com três, *Random* com dois e o FRR-MAB com apenas um caso.

Por outro lado, dos casos em que o SPEA2 não obteve os melhores valores de ER este obteve os piores valores ou próximo dos piores valores de ER. Além disso, o método de seleção *Random* foi o único algoritmo que obteve o valor de ER igual a 1 (maior valor possível), fato observado na formulação 3O para o sistema `bisect`. Nessa situação o método de seleção *Random* gerou a maior quantidade de soluções para esse caso, porém não obteve nenhuma solução na PF_{true}, o que sugere que a utilização de uma abordagem aleatória não explora o espaço de busca adequadamente em relação ao contexto analisado considerando as formulações propostas. Entretanto, ao comparar o método de seleção *Random* com os outros métodos de seleção em relação aos valores de ER este, em sua maioria, obteve melhores valores que os outros métodos de seleção, dessa forma, demonstrando que a utilização de uma abordagem aleatória mostra-se adequada em relação a diversidade de soluções para o contexto analisado considerando as formulações propostas.

Tabela 6.26: *Post-hoc* - Média de Hipervolume (Desvio Padrão) da formulação 2OSSHOM

Sistema	Comparação	*p-value*	Dif. Obs.	Dif. Crit.	Diferença	*Effect Size*	Magnitude
bisect	CF / FRR-MAB	1.42e-06	49.13	23.695	TRUE	0	insignificante
	CF / Random	0.838	8.27	23.695	FALSE	0.362	insignificante
	CF / SPEA2	1.21e-04	40.87	23.695	TRUE	1	grande
	FRR-MAB / Random	1.21e-04	40.87	23.695	TRUE	1	grande
	FRR-MAB / SPEA2	< 2e-16	90.00	23.695	TRUE	1	grande
	Random / SPEA2	1.43e-06	49.13	23.695	TRUE	1	grande
bub	CF / FRR-MAB	0.840	8.23	23.695	FALSE	0.631	pequeno
	CF / Random	0.002	34.67	23.695	TRUE	0.846	grande
	CF / SPEA2	9.21e-15	74.30	23.695	TRUE	1	grande
	FRR-MAB / Random	0.034	26.43	23.695	TRUE	0.833	grande
	FRR-MAB / SPEA2	1.06e-11	66.07	23.695	TRUE	1	grande
	Random / SPEA2	2.18e-04	39.63	23.695	TRUE	1	grande
find	CF / FRR-MAB	0.80	9.00	23.695	FALSE	0.650	médio
	CF / Random	1.30e-06	49.30	23.695	TRUE	0.999	grande
	CF / SPEA2	1.11e-16	79.43	23.695	TRUE	1	grande
	FRR-MAB / Random	1.59e-04	40.30	23.695	TRUE	0.997	grande
	FRR-MAB / SPEA2	2.81e-13	70.43	23.695	TRUE	1	grande
	Random / SPEA2	0.010	30.13	23.695	TRUE	1	grande
fourballs	CF / FRR-MAB	0.631	11.80	23.695	FALSE	0.618	pequeno
	CF / Random	8.16e-04	36.70	23.695	TRUE	0.921	grande
	CF / SPEA2	1.67e-15	76.17	23.695	TRUE	1	grande
	FRR-MAB / Random	0.053	24.90	23.695	TRUE	0.763	grande
	FRR-MAB / SPEA2	4.10e-11	64.37	23.695	TRUE	1	grande
	Random / SPEA2	2.36e-04	39.47	23.695	TRUE	1	grande
mid	CF / FRR-MAB	5.35e-04	37.67	23.695	TRUE	0.819	grande
	CF / Random	8.04e-04	36.73	23.695	TRUE	0.874	grande
	CF / SPEA2	8.88e-16	76.80	23.695	TRUE	0.967	grande
	FRR-MAB / Random	1	0.93	23.695	FALSE	0.49	insignificante
	FRR-MAB / SPEA2	2.75e-04	39.13	23.695	TRUE	0.933	grande
	Random / SPEA2	1.78e-04	40.07	23.695	TRUE	0.9	grande
triangle	CF / FRR-MAB	7.55e-05	41.83	23.695	TRUE	0.953	grande
	CF / Random	1.70e-04	40.17	23.695	TRUE	0.958	grande
	CF / SPEA2	< 2e-16	87.33	23.695	TRUE	1	grande
	FRR-MAB / Random	0.999	1.67	23.695	FALSE	0.47	insignificante
	FRR-MAB / SPEA2	1.12e-05	45.50	23.695	TRUE	1	grande
	Random / SPEA2	4.45e-06	47.17	23.695	TRUE	1	grande

Por fim, observa-se que o número de soluções presentes na PF_{true} diminui quando é adicionado mais um objetivo (formulação 3O), isso é esperado, uma vez que a dificuldade no processo de busca aumentou.

Posteriormente, na segunda parte do estudo, foram utilizadas as estratégias tradicionais de geração de HOMs e o melhor algoritmo encontrado na primeira parte do estudo, nesse caso, o algoritmo SPEA2 com o método de seleção CF o qual será denominado apenas como algoritmo CF e assim representando a abordagem HG4HOM.

Cada estratégia tradicional produz apenas um conjunto de HOMs para cada sistema devido a sua natureza estocástica, com exceção da estratégia *RandomMix* a qual foi executada 30 vezes e produziu 30 conjuntos para cada sistema. Esses conjuntos foram submetidos ao conjunto de teste T e cada conjunto gerou uma matriz informando por qual caso de teste cada mutante era morto. Esse procedimento também foi aplicado no algoritmo CF em relação aos resultados obtidos na primeira parte do estudo, porém para uma análise mais detalhada foi considerada cada solução produzida como um conjunto de HOMs em cada execução independente realizada para cada formulação em cada sistema. Dessa maneira, foram avaliadas as soluções produzidas em relação às 30 execuções independentes realizadas as quais produziram um total de 11.160 matrizes para a formulação 2OWSD, 11.160 matrizes para a formulação 2OSSHOM e 9 mil matrizes para a formulação 3O, resultando num total de 31.320 matrizes analisadas.

Tabela 6.27: *Post-hoc* - Média de Hipervolume (Desvio Padrão) da formulação 3O

Sistema	Comparação	*p-value*	Dif. Obs.	Dif. Crit.	Diferença	*Effect Size*	Magnitude
bisect	CF / FRR-MAB	0.366	16.00	23.695	FALSE	0.316	insignificante
	CF / Random	6.53e-08	54.10	23.695	TRUE	0.952	grande
	CF / SPEA2	< 2e-16	40.03	23.695	TRUE	0.883	grande
	FRR-MAB / Random	3.74e-13	70.10	23.695	TRUE	1	grande
	FRR-MAB / SPEA2	1.80e-08	56.03	23.695	TRUE	1	grande
	Random / SPEA2	0.484	14.07	23.695	FALSE	0.3	insignificante
bub	CF / FRR-MAB	0.955	5.13	23.695	FALSE	0.576	pequeno
	CF / Random	6.66e-04	37.17	23.695	TRUE	0.894	grande
	CF / SPEA2	1.11e-14	74.10	23.695	TRUE	1	grande
	FRR-MAB / Random	0.005	32.03	23.695	TRUE	0.874	grande
	FRR-MAB / SPEA2	9.79e-13	68.97	23.695	TRUE	1	grande
	Random / SPEA2	7.38e-04	36.93	23.695	TRUE	1	grande
find	CF / FRR-MAB	0.946	5.46	23.695	FALSE	0.591	pequeno
	CF / Random	3.23e-06	47.73	23.695	TRUE	1	grande
	CF / SPEA2	3.33e-16	77.73	23.695	TRUE	1	grande
	FRR-MAB / Random	6.08e-05	42.27	23.695	TRUE	1	grande
	FRR-MAB / SPEA2	5.70e-14	72.27	23.695	TRUE	1	grande
	Random / SPEA2	0.011	30.00	23.695	TRUE	1	grande
fourballs	CF / FRR-MAB	1.07e-04	41.13	23.695	TRUE	0.983	grande
	CF / Random	1.95e-05	44.47	23.695	TRUE	0.968	grande
	CF / SPEA2	< 2e-16	88.53	23.695	TRUE	1	grande
	FRR-MAB / Random	0.987	3.33	23.695	FALSE	0.563	pequeno
	FRR-MAB / SPEA2	3.90e-06	47.40	23.695	TRUE	1	grande
	Random / SPEA2	2.41e-05	44.07	23.695	TRUE	1	grande
mid	CF / FRR-MAB	0.45	14.56	23.695	FALSE	0.62	pequeno
	CF / Random	0.13	21.30	23.695	FALSE	0.671	médio
	CF / SPEA2	1	0.93	23.695	FALSE	0.5	insignificante
	FRR-MAB / Random	0.91	6.73	23.695	FALSE	0.559	insignificante
	FRR-MAB / SPEA2	0.39	15.50	23.695	FALSE	0.367	insignificante
	Random / SPEA2	0.11	22.23	23.695	FALSE	0.311	insignificante
triangle	CF / FRR-MAB	3.53e-05	43.33	23.695	TRUE	1	grande
	CF / Random	5.90e-06	46.67	23.695	TRUE	1	grande
	CF / SPEA2	< 2e-16	90.00	23.695	TRUE	1	grande
	FRR-MAB / Random	0.99	3.33	23.695	FALSE	0.555	insignificante
	FRR-MAB / SPEA2	5.90e-06	46.67	23.695	TRUE	1	grande
	Random / SPEA2	3.53e-05	43.33	23.695	TRUE	1	grande

Tabela 6.28: Fronteira de Pareto das formulações

Sistema	Formulação	PF_{true}	SPEA2		Random		CF		FRR-MAB	
			PF_{know}	ER	PF_{know}	ER	PF_{know}	ER	PF_{know}	ER
bisect	2OWSD	289	**182 (179)**	**0.02**	133 (50)	0.62	146 (43)	0.71	189 (53)	0.72
	2OSSHOM	257	**58 (58)**	**0**	134 (132)	0.01	136 (134)	0.01	212 (184)	0.13
	3O	262	120 (52)	0.57	412 (0)	1	216 (144)	0.33	**225 (184)**	**0.18**
bub	2OWSD	325	**165 (165)**	**0**	204 (119)	0.42	228 (102)	0.55	227 (70)	0.69
	2OSSHOM	322	**173 (171)**	**0.01**	237 (133)	0.44	247 (81)	0.67	227 (97)	0.57
	3O	293	**150 (148)**	**0.01**	215 (112)	0.48	200 (45)	0.78	198 (83)	0.58
find	2OWSD	342	**237 (237)**	**0**	156 (32)	0.79	155 (82)	0.47	235 (29)	0.88
	2OSSHOM	319	**209 (201)**	**0.04**	167 (83)	0.50	236 (46)	0.81	245 (37)	0.85
	3O	258	**160 (142)**	**0.11**	137 (74)	0.46	213 (36)	0.83	224 (24)	0.89
fourballs	2OWSD	341	**233 (230)**	**0.01**	180 (78)	0.57	251 (32)	0.87	164 (33)	0.80
	2OSSHOM	330	**204 (198)**	**0.03**	186 (94)	0.49	217 (52)	0.76	225 (26)	0.88
	3O	235	**154 (113)**	**0.27**	135 (67)	0.50	171 (45)	0.74	141 (29)	0.79
mid	2OWSD	315	165 (35)	0.79	**282 (255)**	**0.1**	196 (117)	0.40	166 (45)	0.73
	2OSSHOM	253	153 (1)	0.99	**183 (151)**	**0.17**	154 (75)	0.51	225 (88)	0.61
	3O	294	98 (2)	0.98	261 (209)	0.2	**237 (198)**	**0.16**	253 (204)	0.19
triangle	2OWSD	293	**170 (160)**	**0.06**	219 (33)	0.85	201 (103)	0.49	222 (23)	0.90
	2OSSHOM	240	178 (65)	0.63	239 (20)	0.92	**195 (140)**	**0.28**	224 (27)	0.88
	3O	204	152 (27)	0.82	181 (34)	0.81	**154 (126)**	**0.18**	214 (21)	0.90

Além disso, foi avaliado o conjunto de casos de teste adequado aos FOMs, denominado de estratégia Original, para o qual foi adotado procedimento semelhante ao descrito anteriormente.

As matrizes geradas pelas estratégias tradicionais e Original, além do algoritmo CF, foram submetidas a um algoritmo de otimização [34] para obter o conjunto adequado e reduzido de casos de teste. Nesse etapa para cada matriz o algoritmo foi executado 10 vezes, gerando 10 conjuntos de casos de teste adequados, ao final foi determinado uma média das 10 execuções em relação ao tamanho do conjunto de casos de teste adequado $|T_s|$ e o escore de mutação MS_s em relação aos FOMs (conjunto MG da Tabela 6.2).

Na Tabela 6.29 são apresentados os resultados obtidos em relação à análise das matrizes. A primeira coluna apresenta a abordagem adotada (estratégia ou o algoritmo CF), a segunda coluna determina a métrica avaliada (escore de mutação ou conjunto de casos de teste adequado), a terceira coluna representa o valor médio em relação aos valores obtidos para cada sistema analisado (colunas quatro a nove). Os melhores valores estão destacados em negrito: maior escore de mutação e menor conjunto de casos de teste; e em cinza os valores estatisticamente equivalentes de acordo com o teste de Friedman.

Tabela 6.29: Média de escore de mutação e tamanho do conjunto de casos de teste adequado em relação aos FOMs

Abordagem	**Métrica**	**Média**	**Sistema** `bisect`	`bub`	`find`	`fourballs`	`mid`	`triangle`
Original	MS_s	100%	100%	100%	100%	100%	100%	100%
	$\|T_s\|$	75.42	10.5	28.9	14.5	92.7	60.6	245.3
DiffOp	MS_s	**95%**	93%	95%	**98%**	89%	**97%**	**96%**
	$\|T_s\|$	29.5	3.1	2.7	6.5	8.1	15.8	140.9
Each-Choice	MS_s	92%	91%	96%	96%	88%	89%	93%
	$\|T_s\|$	19.1	3	3.3	5	6.7	**6.5**	90.1
First2Last	MS_s	94%	**95%**	**97%**	91%	87%	**97%**	94%
	$\|T_s\|$	**18.6**	3	6.2	**1.5**	**4.1**	12.6	**84.4**
RandomMix	MS_s	93%	93%	88%	95%	**90%**	**97%**	**96%**
	$\|T_s\|$	28.7	2.6	3.4	2.6	14	21.2	128.6
CF (2OWSD)	MS_s	88%	90%	83%	92%	81%	87%	93%
	$\|T_s\|$	24.6	3	1.7	3.6	5.9	12.2	121
CF (2OSSHOM)	MS_s	86%	83%	80%	92%	81%	89%	92%
	$\|T_s\|$	24.4	**1.9**	**1.6**	3.6	5.8	12.6	120.7
CF (3O)	MS_s	84%	79%	84%	93%	83%	71%	92%
	$\|T_s\|$	23	2.5	2	3.8	6.2	7.9	114.9

Em relação aos resultados apresentados na Tabela 6.29 constata-se que o algoritmo CF obteve o menor conjunto de casos de teste em dois casos, a estratégia *First2Last* obteve em três casos e a estratégia *Each-Choice* em um caso. Em relação ao escore de mutação a estratégia *DiffOp* obteve o maior valor ou igual ao maior valor em três casos, *First2Last* em três casos e *RandomMix* em três casos, sendo que *DiffOp*, *First2Last* e *RandomMix* obtiveram o mesmo escore de mutação no sistema `mid` e as estratégias *DiffOp* e *RandomMix* no sistema `triangle`. De acordo com as médias gerais obtidas pelas estratégias e pelo algoritmo CF observa-se que a estratégia *DiffOp* obteve a melhor média para o escore de mutação e a estratégia *First2Last* obteve a melhor média de tamanho do conjunto de casos de teste.

Ao avaliar o desempenho do algoritmo CF em relação às estratégias observa-se que, em alguns casos, esse obteve menores conjuntos de casos de teste, bem como valores de escore de mutação próximos dos valores obtidos pelas estratégias tradicionais. Pode-se citar os casos dos sistemas `bisect` e `bub` nos quais a abordagem para a formulação 2OSSHOM obteve os

menores conjuntos de casos de teste em relação às estratégias tradicionais. Na comparação do desempenho das formulações observa-se que entre as formulações 2OWSD e 2OSSHOM, a formulação 2OWSD obteve em média maior escore de mutação e a formulação 2OSSHOM conseguiu em média um menor conjunto de casos de teste. Entretanto, em relação à formulação 3O essa obteve em média um menor conjunto de casos de teste em relação às outras formulações.

Ao ser aplicado o teste estatístico nos valores apresentados na Tabela 6.29 obtiveram-se os valores de *p-value* 8.394e-04 para o teste de Friedman e 1.576e-05 para o teste de Iman e Davenport em relação aos valores de escore de mutação, dessa forma, necessitando-se aplicar o teste *post-hoc* o qual é apresentado na Tabela 6.30. Em relação aos valores do tamanho dos conjuntos de casos de teste, o teste de Friedman obteve o valor de *p-value* 0.06399 e o teste de Iman e Davenport o valor de 0.04611, assim, não necessitando o teste *post-hoc* e os valores equivalentes sendo destacados na Tabela 6.29 em cinza. Dessa forma, constata-se que o algoritmo CF obteve resultado equivalente às estratégias em relação ao tamanho do conjunto de casos de teste.

Tabela 6.30: *Post-hoc* - Média de escore de mutação e tamanho do conjunto de casos de teste adequado em relação aos FOMs

	DiffOp	Each-Choice	First2Last	RandomMix	CF (2OWSD)	CF (2OSSHOM)	CF (3O)
DiffOp	n/a	1.000	1.000	1.000	**0.049**	**0.028**	**0.040**
Each-Choice	1.000	n/a	1.000	1.000	0.356	0.293	0.316
First2Last	1.000	1.000	n/a	1.000	0.293	0.178	0.192
RandomMix	1.000	1.000	1.000	n/a	0.064	**0.049**	0.062
CF (2OWSD)	**0.049**	0.356	0.293	0.064	n/a	1.000	1.000
CF (2OSSHOM)	**0.028**	0.293	0.178	**0.049**	1.000	n/a	1.000
CF (3O)	**0.040**	0.316	0.192	0.062	1.000	1.000	n/a

Na Tabela 6.30 observa-se que as estratégias tradicionais obtiveram equivalência estatística em relação ao escore de mutação, o qual é destacado em cinza na Tabela 6.29. Nessa análise, o algoritmo CF obteve diferença estatística apenas em relação a estratégia *DiffOp* que obteve o melhor escore de mutação e em relação a estratégia *RandomMix* na formulação 2OSSHOM. Dessa forma, o algoritmo CF obteve valores com equivalência estatística em relação às estrategias *Each-Choice* e *First2Last*, e com a estratégia *RandomMix* nas formulações 2OWSD e 3O.

Através dos valores apresentados nas Tabelas 6.29 e 6.30 observa-se que a abordagem proposta, para as três formulações descritas na Tabela 6.1, conseguiu reduzir o número de casos de teste necessários, sendo equivalente estatisticamente às estratégias tradicionais. Entretanto, houve a diminuição do escore de mutação havendo significância estatística em relação às estratégias tradicionais.

Em seguida, foram gerados valores de *fitness* em relação às formulações propostas para as soluções geradas pelas estratégias tradicionais. Posteriormente, foram determinadas as melhores soluções em cada sistema pelas estratégias e pelo algoritmo CF para cada formulação as soluções que obtiveram o menor valor de ED encontrado em relação à PF_{true}. Para a situação de haver muitas soluções que possuam o valor de ED igual ao menor valor de ED encontrado, foi considerada a primeira solução igual ao menor valor de ED. As soluções encontradas e seus valores de ED são apresentados na Tabela 6.31. Cada solução encontrada possui os valores dos objetivos encontrados na seguinte ordem para as formulações: (i) 2OWSD (primeiro objetivo, segundo objetivo); (ii) 2OSSHOM (primeiro objetivo, terceiro objetivo); e (iii) 3O (primeiro objetivo, segundo objetivo, terceiro objetivo).

Tabela 6.31: *fitness* das soluções encontradas com o menor valor de ED

Abordagem	Formulação	Métrica	Sistema					
			bisect	bub	find	fourballs	mid	triangle
DiffOp	2OWSD	ED	1.09	1.08	0.39	1.11	0.51	2.06
		Fitness	(0.44, 0.0)	(0.42, 0.0)	(0.40, 0.0)	(0.48, 2.0)	(0.51, 1.0)	(0.49, 0.0)
	2OSSHOM	ED	1.09	1.08	1.08	0.48	0.50	2.06
		Fitness	(0.44, 0.0)	(0.42, 0.0)	(0.40, 0.0)	(0.48, 2.0)	(0.51, 1.0)	(0.49, 0.0)
	3O	ED	2.86	1.47	1.47	0.48	0.51	1.50
		Fitness	(0.44, 0.0, 0.0)	(0.42, 0.0, 0.0)	(0.40, 0.0, 0.0)	(0.48, 2.0, 2.0)	(0.51, 1.0, 1.0)	(0.49, 0.0, 0.0)
Each-Choice	2OWSD	ED	0.21	0.24	1.02	0.23	1.01	0.15
		Fitness	(0.22, 2.0)	(0.26, 4.0)	(0.19, 2.0)	(0.24, 1.0)	(0.14, 0.0)	(0.15, 3.0)
	2OSSHOM	ED	0.21	0.24	0.18	1.03	1	1.01
		Fitness	(0.22, 2.0)	(0.26, 4.0)	(0.19, 2.0)	(0.24, 1.0)	(0.14, 0.0)	(0.15, 3.0)
	3O	ED	0.24	0.21	1.43	1.43	1.42	0.15
		Fitness	(0.26, 4.0, 4.0)	(0.22, 2.0, 2.0)	(0.19, 2.0, 2.0)	(0.24, 1.0, 1.0)	(0.14, 0.0, 0.0)	(0.15, 3.0, 3.0)
First2Last	2OWSD	ED	0.42	0.39	0.43	1.08	0.46	0.44
		Fitness	(0.43, 1.0)	(0.40, 1.0)	(0.43, 1.0)	(0.42, 2.0)	(0.47, 2.0)	(0.45, 3.0)
	2OSSHOM	ED	0.42	0.39	0.43	0.41	0.46	1.09
		Fitness	(0.43, 1.0)	(0.40, 1.0)	(0.43, 1.0)	(0.42, 2.0)	(0.47, 2.0)	(0.45, 3.0)
	3O	ED	1.48	0.39	0.43	0.41	0.46	0.44
		Fitness	(0.43, 1.0, 1.0)	(0.40, 1.0, 1.0)	(0.43, 1.0, 1.0)	(0.42, 2.0, 2.0)	(0.47, 2.0, 2.0)	(0.45, 3.0, 3.0)
RandomMix	2OWSD	ED	0.41	0.38	0.40	0.42	0.43	0.44
		Fitness	(0.43, 3.0)	(0.42, 8.0)	(0.41, 3.0)	(0.43, 4.0)	(0.44, 3.0)	(0.45, 7.0)
	2OSSHOM	ED	0.41	0.38	0.40	0.42	0.43	0.44
		Fitness	(0.43, 3.0)	(0.42, 8.0)	(0.41, 3.0)	(0.43, 4.0)	(0.44, 3.0)	(0.45, 7.0)
	3O	ED	0.41	0.38	0.40	0.42	0.43	0.44
		Fitness	(0.43, 3.0, 3.0)	(0.42, 8.0, 8.0)	(0.41, 3.0, 3.0)	(0.43, 4.0, 4.0)	(0.44, 3.0, 3.0)	(0.45, 7.0, 7.0)
CF	2OWSD	ED	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
		Fitness	(0.06, 12.0)	(0.03, 6.0)	(0.01, 4.0)	(0.06, 17.0)	(0.02, 6.0)	(4.36, 1790.0)
	2OSSHOM	ED	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
		Fitness	(0.21, 42.0)	(0.08, 13.0)	(0.05, 14.0)	(0.02, 7.0)	(0.03, 11.0)	(0.05, 32.0)
	3O	ED	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
		Fitness	(0.23, 46.0, 46.0)	(0.05, 8.0, 8.0)	(0.05, 16.0, 16.0)	(0.02, 5.0, 5.0)	(0.02, 7.0, 7.0)	(0.79, 363.0, 363.0)

Na Tabela 6.31 observa-se que a abordagem obteve os melhores valores de ED, sendo que, quanto menor o valor de ED melhor é a solução encontrada. Dessa maneira, a abordagem encontrou soluções que estão presentes na PF_{true} em relação às formulações, ao contrário das estratégias que obtiveram soluções mais distantes da fronteira. Assim, os resultados quando comparados com a Tabela 6.29 indicam que apesar do alto escore de mutação e baixo número de casos de teste necessários pelas estratégias essas obtiveram soluções dominadas pela abordagem em relação às formulações.

Ao analisar o tamanho das soluções (primeiro objetivo), as soluções geradas pela abordagem, em sua maioria, possuem os menores tamanhos encontrados em relação às estratégias. Além disso, mesmo possuindo soluções com tamanhos inferiores aos das estratégias, a abordagem obteve soluções que encontraram mais HOMs difíceis de serem mortos e capazes de substituírem seus FOMs constituintes. Interessante destacar que a estratégia *DiffOp* que obteve maior escore de mutação (Tabela 6.29) foi pior do que as outras estratégias em relação a encontrar HOMs que atendessem ao segundo e terceiro objetivos. Entre as estratégias tradicionais a estratégia *Each-Choice* obteve os melhores *trade-offs* entre os objetivos propostos o que corrobora com os resultados obtidos no trabalho de Prado Lima et *al.* [111], onde os autores também identificaram que a estratégia *Each-Choice* obteve os melhores resultados em relação a média de escore de mutação por mutantes e média de escore de mutação por casos de teste dentre as estratégias tradicionais de geração de HOMs.

Na Tabela 6.32 são apresentadas as médias dos valores de ED da Tabela 6.31. Ao avaliar os valores apresentados observa-se que a estratégia *RandomMix* obteve os menores valores de ED, isso é esperado, uma vez que essa estratégia por ser estocástica foi executada 30 vezes em cada sistema, tendo assim uma melhor oportunidade na exploração de HOMs. Entretanto, a estratégia *DiffOp* que, como mencionado anteriormente, obteve o melhor escore de mutação, nessa avaliação obteve as piores médias de ED.

Por fim, observa-se que, nesse contexto, a aplicação do conceito de hiper-heurística pode auxiliar o testador na redução do esforço na determinação da melhor estratégia, além de obter

Tabela 6.32: Média dos valores de ED de cada formulação

Abordagem	Média		
	2OWSD	2OSSHOM	3O
DiffOp	1.04	1.05	1.38
Each-Choice	0.48	0.61	0.81
First2Last	0.54	0.53	0.60
RandomMix	0.41	0.41	0.41
CF	**0.0**	**0.0**	**0.0**

resultados similares em relação ao escore de mutação e números de casos de teste, bem como encontrar soluções que sejam capazes de substituírem seus FOMs constituintes e encontrem mutantes mais difíceis de serem mortos que podem estar associados a diferentes defeitos não revelados pelos FOMs.

6.6 Respondendo às Questões de Pesquisa

Nesta seção são apresentadas respostas às questões de pesquisas propostas na Seção 6.2 visando a responder a seguinte questão de pesquisa geral: "Como são os resultados produzidos pela abordagem HG4HOM quando comparados com os algoritmos multi-objetivos e estratégias tradicionais de geração de HOMs?"

QP1: Qual algoritmo multi-objetivo é o melhor em relação aos objetivos propostos? Como apresentado na Tabela 6.12, o algoritmo SPEA2 destacou-se pelos seus resultados em todas as formulações propostas tendo uma diferença significante de valores encontrados em relação ao NSGA-II. Do ponto de vista do testador ele pode optar por utilizar o SPEA2 com ou sem arquivamento, sendo que a utilização do arquivamento com o dobro de tamanho em relação a população obteve os melhores resultados na maioria dos casos como pode ser observado na Tabela 6.9.

QP2: Quais são os resultados obtidos pelos métodos de seleção? Os métodos de seleção CF e FRR-MAB obtiveram os valores mais expressivos, com diferença estatística, em relação ao método de seleção *Random* (Tabela 6.24). Observa-se que os melhores resultados encontrados pelos métodos de seleção CF e FRR-MAB obtiveram um *effect size* com uma magnitude grande quando comparados ao método de seleção *Random* na maioria dos casos (Tabelas 6.25, 6.26 e 6.27). Desse modo, a utilização adequada das LLHs pode selecionar determinados mutantes para contribuir com a solução que estão presentes em algumas estratégias determinísticas. As LLHs podem ser mais facilmente selecionadas do que através de uma estratégia estocástica. Entre os métodos de seleção CF e FRR-MAB, o método de seleção CF obteve melhores resultados na maioria dos sistemas para cada formulação. Do ponto de vista do testador ele pode optar entre os métodos de seleção CF e FRR-MAB que obtiveram equivalência estatística em quase todos os sistemas, porém caso o testador opte por obter soluções mais adequadas em relação às formulações propostas ele pode optar pelo método de seleção CF que obteve, um melhor desempenho, como descrito anteriormente, e além disso obteve na maioria dos casos os melhores valores de ER (Tabela 6.28). Tendo em vista o tempo gasto para a configuração dos parâmetros o testador pode optar pelo método aleatório, porém esse método não possui um desempenho superior aos outros métodos de seleção em relação às formulações.

QP3: Como são os resultados do melhor algoritmo multi-objetivo quando comparados com a HG4HOM? A abordagem HG4HOM utiliza algoritmo multi-objetivo com hiper-heurística, desse modo, como constatado nas questões anteriores o melhor método de seleção encontrado é o CF com o algoritmo SPEA2. Quando comparados os resultados em relação ao melhor algoritmo multi-objetivo, SPEA2, os valores obtidos de HV em todos os sistemas para todas as formulações, Tabela 6.24, possuem uma diferença estatística com magnitude grande ($\hat{A}_{12} \geq 0.71$), muitas vezes com o valor de *effect size* igual a 1 (Tabelas 6.25, 6.26 e 6.27). Entretanto, ao avaliar os valores de ER obtidos (Tabela 6.28) o algoritmo SPEA2 sem o conceito de hiper-heurística obteve a maioria dos melhores valores, ou seja, isso indica que esse algoritmo obteve mais soluções que não eram dominadas por outras embora essas soluções não sejam melhores em relação aos objetivos de cada formulação.

QP4: Como são os resultados da abordagem HG4HOM quando comparados às estratégias tradicionais de geração de HOMs em termos de escore de mutação e redução no número de casos de teste? De acordo com os resultados apresentados na Tabela 6.29 a abordagem HG4HOM não obteve em média os melhores valores, porém obteve escore de mutação próximo ao das estratégias tradicionais e número de casos de teste com equivalência estatística para o melhor valor.

QP5: Qual o desempenho da abordagem HG4HOM quando comparada às estratégias tradicionais de geração de HOMs em relação aos objetivos propostos? Ao analisar as soluções observa-se que a abordagem HG4HOM obteve melhores valores de ED em relação às estratégias tradicionais (Tabela 6.31), bem como obteve soluções que apresentaram melhores valores para o primeiro, segundo e terceiro objetivos. Dessa maneira, a utilização da abordagem propiciou encontrar soluções que contém HOMs que possam revelar defeitos e que possam substituir seus FOMs constituintes. Entretanto, a não determinação dos HOMs equivalentes e a não geração de novos casos de teste que matam os HOMs encontrados prejudicou o desempenho da abordagem em relação ao escore de mutação. Dessa maneira, observa-se que no contexto em que a abordagem foi aplicada a sua utilização pode auxiliar o testador na redução do esforço na determinação da melhor estratégia, além de obter resultados similares em relação ao escore de mutação e números de casos de teste, bem como encontrar soluções que sejam capazes de substituírem seus FOMs constituintes e encontrem mutantes mais difíceis de serem mortos.

6.7 Ameaças à Validade

No presente trabalho alguns pontos foram detectados que podem ser considerados como ameaças à validade dos resultados e são descritos a seguir.

- Os experimentos foram conduzidos com apenas seis sistemas. É necessário executar os experimentos com um conjunto mais amplo de sistemas maiores ou reais os quais poderiam aumentar a acurácia das análises e testes estatísticos realizados;
- Os resultados dependem fortemente dos operadores e das estratégias utilizadas, sendo possível que os resultados melhorem com um conjunto mais amplo desses operadores;
- Não foram determinados os HOMs equivalentes gerados pela abordagem e nem criados novos casos de teste para que esses mutantes vivos fossem mortos, desse modo, melhorando os resultados obtidos. Isso levou a diminuição do escore de mutação dos casos de teste que matam os HOMs gerados. Ainda há uma limitação relacionada à avaliação das

soluções quando o objetivo da busca é encontrar mutantes mais difíceis de serem mortos. Geralmente tais mutantes não são mortos pelos casos de teste disponíveis e para matá-los novos casos de teste precisam ser gerados;

- A configuração dos experimentos e a avaliação dos métodos de seleção foram conduzidos apenas com um dos algoritmos multi-objetivos. A utilização de um amplo conjunto de algoritmos auxilia para um melhor ajuste de parâmetros e uma melhor exploração do espaço de busca para encontrar soluções melhores de acordo com as várias características dos algoritmos.

6.8 Considerações Finais

Este capítulo apresentou os resultados da avaliação experimental conduzida em seis sistemas com dois algoritmos multi-objetivos e três métodos de seleção. Os algoritmos foram configurados utilizando o *irace*, posteriormente as melhores configurações foram executadas 10 vezes. Foi aplicado o teste estatístico de Friedman e seu *post-hoc* para determinar o melhor algoritmo multi-objetivo e os melhores métodos de seleção. Os experimentos foram executados 30 vezes em cada sistema para cada formulação com as melhores configurações encontradas. A comparação dos resultados foi realizada com base no indicador de qualidade Hipervolume e utilizado o teste estatístico de Kruskal-Wallis, seu *post-hoc* e *effect size* no auxílio das avaliações.

De acordo com os experimentos realizados, os resultados mostraram que o algoritmo SPEA2 obteve melhores soluções para as formulações quando comparado com o NSGA-II. Na avaliação dos métodos de seleção juntamente com o algoritmo SPEA2, os métodos de seleção CF e FRR-MAB obtiveram os melhores resultados, superiores ao algoritmo SPEA2 sem o conceito de hiper-heurística. Dentre os métodos de seleção o CF apresentou os melhores resultados em geral.

Em comparação com as estratégias tradicionais a abordagem obteve resultados próximos em relação ao escore de mutação e valor equivalente ao melhor em relação ao tamanho do conjunto de casos de teste adequado. Considerando os valores de ED obtidos pela abordagem e pelas estratégias em relação às formulações propostas, a abordagem obteve os melhores resultados. Desse modo, a abordagem auxilia o testador na redução do esforço em escolher uma estratégia de geração de HOMs específica, bem como a encontrar determinados tipos de HOMs, produzindo resultados similares ao das estratégias tradicionais.

Capítulo 7

Conclusão

O presente trabalhou introduziu uma abordagem denominada *Hyper-Heuristic for Generation of Higher Order Mutants* (HG4HOM) que utiliza algoritmos multi-objetivos com o conceito de hiper-heurística na geração de conjuntos de HOMs mais adequados a um domínio de um problema a ser investigado, afim de manter a eficácia dos teste. A abordagem visa a reduzir os esforços do testador na implementação de uma solução de busca, configuração de algoritmos e utilizar as principais vantagens das diferentes estratégias tradicionais de geração de HOMs (não baseados em busca) existentes.

Para isso, a abordagem inclui uma representação para o problema que permite que uma solução represente um conjunto de HOMs adequados a serem utilizados no teste de mutação. Busca-se para esses conjuntos os mutantes que possam revelar defeitos ainda não revelados, reduzir o número de mutantes ou substituir seus FOMs constituintes sem perder a eficácia do teste com relação ao escore de mutação. A determinação do conjunto adequado é realizado por meio de operadores de busca que implementam as estratégias tradicionais de geração de HOMs.

Para implementar a abordagem proposta foram utilizados os algoritmos NSGA-II [24] e SPEA2 [131] e os métodos de seleção *Choice Function*, adaptado por Maashi et *al.* [79], o *Fitness-Rate-Rank based Multi-Armed Bandit* (FRR-MAB) e a seleção aleatória (*Random*). Na avaliação da qualidade da aplicação das heurísticas de baixo nível foi utilizado o conceito de dominância [48]. As estratégias de geração de HOMs utilizadas foram [89, 110]: *FirstToLast*, *RandomMix*, *Different-Operators* e *Each-Choice*. A configuração da abordagem foi realizada utilizando *irace* [74].

Experimentos foram conduzidos avaliando a abordagem em relação à utilização dos algoritmos convencionais. Além disso, foi realizada a comparação para a determinação do melhor método de seleção e o comportamento da abordagem em relação às estratégias tradicionais. Esses experimentos foram realizados em relação a seis sistemas já utilizados na literatura visando os objetivos relacionados ao tamanho de um conjunto de HOMs que possuam preferencialmente os HOMs que sejam capazes de substituírem seus FOMs constituintes e que fossem capazes de revelarem defeitos não revelados pelos FOMs.

Resultados do estudo empírico conduzido mostraram que a aplicação do conceito de hiper-heurística, além de auxiliar o testador na redução do esforço para a determinação da melhor estratégia, obtém resultados similares às estratégias tradicionais em relação ao escore de mutação e número de casos de teste. Além disso, foi possível encontrar tipos de mutantes que atendem aos objetivos descritos anteriormente. Dentre os algoritmos analisados, o SPEA2 destacou-se pelos valores obtidos em relação ao Hipervolume. Em relação aos métodos de seleção, o método *Choice-Function* obteve o melhor desempenho.

Portanto, a abordagem proposta neste trabalho alcançou os objetivos propostos de auxiliar na seleção automática de estratégias de geração de HOMs, e obteve resultados similares aos das estratégias tradicionais com relação ao número de casos de teste e ao escore de mutação, livrando assim o testador da tarefa de ter que analisar e realizar experimentos para escolher a melhor estratégia.

7.1 Trabalhos Futuros

O presente trabalho proporciona alguns aspectos que podem ser estudados e avaliados futuramente, sendo descritos a seguir:

- A utilização de outros algoritmos multi-objetivos na realização dos experimentos, por exemplo, PAES, IBEA, εMOEA, εNSGA-II, NSGA-III, MOCell e MOEA/D-DRA;
- O uso de Evolução Gramatical na geração de novos algoritmos para determinar características favoráveis ao contexto de geração HOMs;
- A utilização de sistemas maiores ou reais. O número de HOMs a serem estudados aumenta de acordo com o tamanho do sistema a ser analisado, dessa maneira, com o uso sistemas maiores ou próximos dos sistemas reais pode-se aumentar a acurácia dos resultados. Além disso, fornecendo indicadores de utilização da abordagem HG4HOM no meio industrial;
- Adição de novos objetivos para avaliar o comportamento da abordagem, por exemplo, a adição da preferência do usuário na busca por determinadas soluções.
- A utilização de uma restrição para haver diferentes ordens de HOMs em uma solução;
- Utilização do procedimento de mutação através da manipulação do *bytecode* ao invés do código fonte original forneceria agilidade do processo de mutação além de se aproximar mais em relação ao uso industrial;
- A utilização de HOMs com ordens maiores poderia revelar mais defeitos e gerar mutantes mais difíceis de serem mortos. Desse modo, havendo a necessidade de criação de novos casos de teste e determinar os mutantes equivalentes, além de avaliar o comportamento da abordagem com outras ordens de HOMs e determinar o quanto os resultados obtidos variam em relação a ordem dada;
- Criação de novos operadores de mutação para a abordagem, com o uso de novas estratégias de geração de HOMs;
- Alterar a representação de uma solução para outros modelos, por exemplo, uma solução composta por apenas um HOM e uma solução composta por HOMs de diferentes ordens;
- Utilizar um procedimento de arquivamento das melhores soluções (soluções não dominadas) encontradas durante o processo de busca semelhante ao processo de arquivamento utilizado pelo SPEA2. Desse modo, avaliar se essa característica influência no processo de busca e provê melhores resultados;
- Utilizar uma abordagem baseada em co-evolução que permita encontrar os mutantes mutantes mais difíceis de serem mortos e ao mesmo tempo gerar casos de teste capazes de matá-los. Asim, procurando evitar possíveis mutantes equivalentes.

Referências Bibliográficas

[1] O. A. Adekanmbi, O. O. Olugbara, and J. Adeyemo. A Comparative study of State-of-the-Art Evolutionary Multi-Objective Algorithms for Optimal Crop-mix planning. *International Journal of Agricultural Science and Technology (IJAST)*, 2, 2014.

[2] P. Ammann and J. Offutt. Using formal methods to derive test frames in category-partition testing. In *Proc. Ninth Annual Conf. Computer Assurance (COMPASS'94), Gaithersburg MD*, pages 69–80. IEEE Computer Society Press, 1994.

[3] S. Anand, E. K. Burke, T. Y. Chen, J. Clark, M. B. Cohen, W. Grieskamp, M. Harman, M. J. Harrold, and P. McMinn. An orchestrated survey of methodologies for automated software test case generation. *Journal of Systems and Software*, 86(8):1978–2001, 2013.

[4] W. K. G. Assunção, T. E. Colanzi, S. R. Vergilio, and A. T. R. Pozo. A multi-objective optimization approach for the integration and test order problem. *Information Sciences*, 267:119–139, 2014.

[5] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

[6] J. Bader, K. Deb, and E. Zitzler. Faster hypervolume-based search using Monte Carlo sampling. In *Multiple Criteria Decision Making for Sustainable Energy and Transportation Systems*, volume 634 of *Lecture Notes in Economics and Mathematical Systems*, pages 313–326. Springer Berlin Heidelberg, 2010.

[7] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *Genetic programming: an introduction*, volume 1. Morgan Kaufmann San Francisco, 1998.

[8] M. P. Basgalupp, R. C. Barros, T. S. da Silva, and A. C. de Carvalho. Software effort prediction: a hyper-heuristic decision-tree based approach. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 1109–1116. ACM, 2013.

[9] B. Bergmann and G. Hommel. Improvements of general multiple test procedures for redundant systems of hypotheses. In *Multiple Hypothesenprüfung / Multiple Hypotheses Testing*, pages 100–115. Springer, 1988.

[10] L. C. T. Bezerra, M. López-Ibáñez, and T. Stützle. *Automatic Design of Evolutionary Algorithms for Multi-Objective Combinatorial Optimization*, pages 508–517. Springer International Publishing, 2014.

[11] L. C. T. Bezerra, M. López-Ibáñez, and T. Stützle. Automatic Component-Wise Design of Multiobjective Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation*, 20(3):403–417, 2016.

[12] M. Birattari, T. Stützle, L. Paquete, K. Varrentrapp, et al. A Racing Algorithm for Configuring Metaheuristics. In *Proceedings of the Genetic and Evolutionary Computation Conference*.

[13] T. A. Budd. Mutation analysis: Ideas, examples, problems and prospects. *Computer Program Testing*, 8:29–l48, 1981.

[14] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013.

[15] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward. A classification of hyper-heuristic approaches. In *Handbook of metaheuristics*, pages 449–468. Springer, 2010.

[16] B. Calvo and G. Santafe. scmamp: Statistical Comparison of Multiple Algorithms in Multiple Problems. *The R Journal*, Accepted for publication, 2015.

[17] V. R. De Carvalho, S. R. Vergilio, and A. Pozo. Uma hiper-heurística de seleção de meta-heurísticas para estabelecer sequências de módulos para o teste de software. In *29th Brazilian Workshop on Search-Based Software Engineering*, 2015.

[18] K. Chakhlevitch and P. Cowling. Hyperheuristics: recent developments. In *Adaptive and multilevel metaheuristics*, pages 3–29. Springer, 2008.

[19] P. Chevalley and P. Thévenod-Fosse. A mutation analysis tool for Java programs. *International journal on software tools for technology transfer*, 5(1):90–103, 2003.

[20] C. C. Coello, G. B. Lamont, and D. A. Van Veldhuizen. *Evolutionary algorithms for solving multi-objective problems*. Springer Science & Business Media, 2007.

[21] W. J. Conover. Practical Nonparametric Statistics, 3rd ed., 1999.

[22] P. Cowling, G. Kendall, and E. Soubeiga. A hyperheuristic approach to scheduling a sales summit. In *Practice and Theory of Automated Timetabling III*, pages 176–190. Springer, 2001.

[23] K. Deb, M. Mohan, and S. Mishra. A Fast Multi-objective Evolutionary Algorithm for Finding Well-Spread Pareto-Optimal Solutions. In *KanGAL Report No. 2003002, Indian Institute of Technology*, Kanpur, India, 2003.

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

[25] M. E. Delamaro, J. C. Maldonado, and M. Jino. *Introdução ao teste de software*. Elsevier, Rio de Janeiro, RJ, Brasil, 2007.

[26] R. A. DeMillo, R. J. Lipton, and F. G. Sayward. Hints on test data selection: Help for the practicing programmer. *Computer*, (4):34–41, 1978.

[27] R. A. DeMillo and A. J. Offutt. Constraint-based automatic test data generation. *IEEE Transactions on Software Engineering*, 17(9), 1991.

[28] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7:1–30, 2006.

[29] A. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, 1999.

[30] A. E. Eiben and S. K. Smit. *Evolutionary Algorithm Parameters and Methods to Tune Them*, pages 15–36. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[31] A. E. Eiben and J. E. Smith. *Introduction to evolutionary computing*, volume 53. Springer, Berlim, 2003.

[32] A.E. Eiben and S.K. Smit. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1):19–31, 2011.

[33] S. Eldh, S. Punnekkat, H. Hansson, and P. Jönsson. Component testing is not enough-a study of software faults in Telecom middleware. In *Testing of Software and Communicating Systems*, pages 74–89. Springer, 2007.

[34] E. L. Féderle, G. Guizzo, T. E. Colanzi, S. R. Vergilio, and E. Spinosa. Seleção de produto baseada em algoritmos multiobjetivos para o teste de mutação de variabilidades. In *IV Workshop de Engenharia de Software Baseada em Busca*, 2013.

[35] T. N. Ferreira, J. N. Kuk, A. T. R. Pozo, and S. R. Vergilio. Product selection based on upper confidence bound MOEA/D-DRA for testing software product lines. In *Proceedings of the 18th IEEE Congress on Evolutionary Computation (CEC '16)*, Vancouver, Canada, July 2016.

[36] T. N. Ferreira, J. A. Prado Lima, A. Strickler, J. N. Kuk, S. R. Vergilio, and A. Pozo. Hyper-heuristic Based Product Selection for Software Product Line Testing. *IEEE Computational Intelligence Magazine*, 2017.

[37] A. Fialho. *Adaptive operator selection for optimization*. PhD thesis, Université Paris Sud-Paris XI, 2010.

[38] A. Fialho, L. Da Costa, M. Schoenauer, and M. Sebag. Dynamic multi-armed bandits and extreme value-based rewards for adaptive operator selection in evolutionary algorithms. In *Learning and Intelligent Optimization*, pages 176–190. Springer, 2009.

[39] M. Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the american statistical association*, 32(200):675–701, 1937.

[40] M. Friedman. A comparison of alternative tests of significance for the problem of m rankings. *The Annals of Mathematical Statistics*, 11(1):86–92, 1940.

[41] S. Garcia and F. Herrera. An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. *Machine Learning Research*, 9:2677–2694, 2008.

[42] P. Garrido and M. C. Riff. DVRP: a hard dynamic combinatorial optimisation problem tackled by an evolutionary hyper-heuristic. *Journal of Heuristics*, 16(6):795–834, 2010.

[43] Patrick Giraudoux. *pgirmess: Data Analysis in Ecology*, 2016. R package version 1.6.4.

[44] R. A. Gonçalves, C. P. Almeida, and A. Pozo. Upper confidence bound (UCB) algorithms for adaptive operator selection in MOEA/D. In *Evolutionary Multi-Criterion Optimization*, pages 411–425. Springer, 2015.

[45] Y. Gong, W. Chen, Z. Zhan, J. Zhang, Y. Li, and Q. Zhang. Distributed evolutionary algorithms and their models: A survey of the state-of-the-art. *Applied Soft Computing*, 2015.

[46] H. J. Greenberg. *Mathematical programming glossary*. Greenberg, 2004.

[47] R. Grissom and J. Kim. Effect Sizes for Research. *A Broad Practical Approach*, 2005.

[48] G. Guizzo, G. M. Fritsche, S. R. Vergilio, and A. T. R. Pozo. A hyper-heuristic for the multi-objective integration and test order problem. In *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference*, pages 1343–1350. ACM, 2015.

[49] M. Harman, E. Burke, J. Clark, X. Yao, et al. Dynamic adaptive search based software engineering. In *ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–8. IEEE, 2012.

[50] M. Harman, Y. Jia, and W. B. Langdon. A manifesto for higher order mutation testing. In *Third International Conference on Software Testing, Verification, and Validation Workshops (ICSTW)*, pages 80–89. IEEE, 2010.

[51] M. Harman, Y. Jia, and W. B. Langdon. Strong higher order mutation-based test data generation. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pages 212–222. ACM, 2011.

[52] M. Harman, Y. Jia, P. R. Mateo, and M. Polo. Angels and monsters: An empirical investigation of potential test effectiveness and efficiency improvement from strongly subsuming higher order mutation. In *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*, pages 397–408. ACM, 2014.

[53] M. Harman, S. A. Mansouri, and Y. Zhang. Search-based software engineering: Trends, techniques and applications. *ACM Computing Surveys (CSUR)*, 45(1):11, 2012.

[54] M. Harman, P. McMinn, J. T. de Souza, and S. Yoo. *Search Based Software Engineering: Techniques, Taxonomy, Tutorial*, pages 1–59. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[55] S. Hartmann. A competitive genetic algorithm for resource-constrained project scheduling. *Naval Research Logistics (NRL)*, 45(7):733–750, 1998.

[56] H. H. Hoos and T. Stützle. *Stochastic local search: Foundations & applications*. Elsevier, 2004.

[57] W. E. Howden. Weak mutation testing and completeness of test sets. *IEEE Transactions on Software Engineering*, (4):371–379, 1982.

[58] S. Hussain. Mutation clustering. *Ms. Th., Kings College London, Strand, London*, 2008.

[59] R. L. Iman and J. M. Davenport. Approximations of the critical region of the friedman statistic. *Communications in Statistics-Theory and Methods*, 9(6):571–595, 1980.

[60] Y. Jia, M. Cohen, and M. Petke. Learning combinatorial interaction test generation strategies using hyperheuristic search. International Conference on Software Engineering (ICSE), 2015.

[61] Y. Jia and M. Harman. Constructing subtle faults using higher order mutation testing. In *Eighth IEEE International Working Conference on Source Code Analysis and Manipulation*, pages 249–258. IEEE, 2008.

[62] Y. Jia and M. Harman. Milu: A customizable, runtime-optimized higher order mutation testing tool for the full C language. In *Practice and Research Techniques, 2008. TAIC PART'08. Testing: Academic & Industrial Conference*, pages 94–98. IEEE, 2008.

[63] Y. Jia and M. Harman. Higher order mutation testing. *Information and Software Technology*, 51(10):1379–1393, 2009.

[64] Y. Jia and M. Harman. An analysis and survey of the development of mutation testing. *IEEE Transactions on Software Engineering*, 37(5):649–678, 2011.

[65] M. Kintis, M. Papadakis, and N. Malevris. Isolating first order equivalent mutants via second order mutation. In *IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST)*, pages 701–710. IEEE, 2012.

[66] J. B. Kollat and P. M. Reed. The value of online adaptive search: a performance comparison of NSGAII, ε-NSGAII and εMOEA. In *Evolutionary Multi-Criterion Optimization (EMO): Third International Conference, Guanajuato, Mexico*, pages 386–398. Springer, 2005.

[67] N. Krasnogor and S. Gustafson. A study on the use of "self-generation" in memetic algorithms. *Natural Computing*, 3(1):53–76, 2004.

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

[69] A. C. Kumari, K. Srinivas, and M. P. Gupta. Software module clustering using a hyper-heuristic based multi-objective genetic algorithm. In *IEEE 3rd International, Advance Computing Conference (IACC), 2013*, pages 813–818. IEEE, 2013.

[70] T. L. Lai and H. Robbins. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4–22, 1985.

[71] W. B. Langdon, M. Harman, and Y. Jia. Efficient multi-objective higher order mutation testing with genetic programming. *Journal of systems and Software*, 83(12):2416–2430, 2010.

[72] K. Li, A. Fialho, S. Kwong, and Q. Zhang. Adaptive operator selection with bandits for a multiobjective evolutionary algorithm based on decomposition. *Evolutionary Computation, IEEE Transactions on*, 18(1):114–130, 2014.

[73] R. Lingampally, A. Gupta, and P. Jalote. A multipurpose code coverage tool for java. In *40th Annual Hawaii International Conference on System Sciences (HICSS)*, pages 261b–261b. IEEE, 2007.

[74] M. López-Ibánez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[75] M. López-Ibánez and T. Stützle. The Automatic Design of Multiobjective Ant Colony Optimization Algorithms. *IEEE Transactions on Evolutionary Computation*, 16(6):861–875, 2012.

[76] S. Luke. *Essentials of Metaheuristics*. Lulu, second edition, 2013. http://cs.gmu.edu/~sean/book/metaheuristics/, acessado em: 22/08/2015.

[77] Y. Ma, J. Offutt, and Y. Kwon. MuJava: an automated class mutation system. *Software Testing, Verification and Reliability*, 15(2):97–133, 2005.

[78] Y. S. Ma and S. W. Kim. Mutation testing cost reduction by clustering overlapped mutants. *Journal of Systems and Software*, 2016.

[79] M. Maashi, E. Özcan, and G. Kendall. A multi-objective hyper-heuristic based on choice function. *Expert Systems with Applications*, 41(9):4475–4493, 2014.

[80] L. Madeyski, W. Orzeszyna, R. Torkar, and M. Józala. Overcoming the equivalent mutant problem: A systematic literature review and a comparative experiment of second order mutation. *Software Engineering, IEEE Transactions on*, 40(1):23–42, 2014.

[81] L. Madeyski and N. Radyk. Judy–a mutation testing tool for Java. *IET software*, 4(1):32–42, 2010.

[82] J. C. Maldonado, A. R. C. Rocha, and K. C. Weber. Qualidade de software: teoria e prática. *São Paulo*, 2001.

[83] J. C. Maldonado, A. M. R. Vincenzi, and M. E. Delamaro. Proteum/IM 2.0: An integrated mutation testing environment. In *Mutation 2000 Symposium, San Jose, CA: Kluwer Academic Publishers*, pages 91–101. Springer, 2000.

[84] T. Mariani, G. Guizzo, S. R. Vergilio, and A. T. R. Pozo. Grammatical Evolution for the Multi-Objective Integration and Test Order Problem. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO'16, pages 1069–1076, New York, NY, USA, 2016. ACM.

[85] O. Maron and A. W. Moore. The Racing Algorithm: Model Selection for Lazy Learners. *Artificial Intelligence Review*, 11(1):193–225, 1997.

[86] P. R. Mateo and M. P. Usaola. Bacterio: Java mutation testing tool: A framework to evaluate quality of tests cases. In *IEEE International Conference on Software Maintenance (ICSM), 2012 28th*, pages 646–649. IEEE, 2012.

[87] P. R. Mateo and M. P. Usaola. Mutant execution cost reduction: Through music (mutant schema improved with extra code). In *IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST), 2012*, pages 664–672. IEEE, 2012.

[88] P. R. Mateo and M. P. Usaola. Parallel mutation testing. *Software Testing, Verification and Reliability*, 23(4):315–350, 2013.

[89] R. P. Mateo, P. U. Macario, F. Aleman, and J. Luis. Validating second-order mutation at system level. *IEEE Transactions on Software Engineering*, 39(4):570–587, 2013.

[90] J. Maturana, A. Fialho, F. Saubion, M. Schoenauer, and M. Sebag. Extreme compass and dynamic multi-armed bandits for adaptive operator selection. In *IEEE Congress on Evolutionary Computation, 2009. CEC'09*, pages 365–372. IEEE, 2009.

[91] P. McMinn. Search-based software test data generation: A survey. *Software Testing Verification and Reliability*, 14(2):105–156, 2004.

[92] G. J. Myers, C. Sandler, and T. Badgett. *The art of software testing*. John Wiley & Sons, 2011.

[93] A. J. Nebro, J. J. Durillo, and M. Vergne. Redesigning the jMetal Multi-Objective Optimization Framework. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO Companion '15, pages 1093–1100, New York, NY, USA, 2015. ACM.

[94] P. B. Nemenyi. *Distribution-free multiple comparisons*. PhD thesis, Princeton University, 1963.

[95] Q. V. Nguyen and L. Madeyski. Searching for strongly subsuming higher order mutants by applying multi-objective optimization algorithm. In *Advanced Computational Methods for Knowledge Engineering*, pages 391–402. Springer, 2015.

[96] Q. V. Nguyen and L. Madeyski. Empirical evaluation of multiobjective optimization algorithms searching for higher order mutants. *Cybernetics and Systems*, 47(1-2):48–68, 2016.

[97] W. Nie. *Cost Evaluation and Portfolio Management Optimization for Biopharmaceutical Product Development*. PhD thesis, UCL (University College London), 2015.

[98] G. Ochoa. Search methodologies in real-world software engineering. In *Proceedings of the 15th annual conference companion on Genetic and evolutionary computation*, pages 1085–1088. ACM, 2013.

[99] A. J. Offutt. Investigations of the software testing coupling effect. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 1(1):5–20, 1992.

[100] A. J. Offutt, G. Rothermel, and C. Zapf. An experimental evaluation of selective mutation. In *Proceedings of the 15th international conference on Software Engineering*, pages 100–107. IEEE Computer Society Press, 1993.

[101] E. Omar, S. Ghosh, and D. Whitley. Constructing subtle higher order mutants for Java and AspectJ programs. In *IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*, pages 340–349. IEEE, 2013.

[102] E. Omar, S. Ghosh, and D. Whitley. Comparing search techniques for finding subtle higher order mutants. In *Proceedings of the 2014 conference on Genetic and evolutionary computation*, pages 1271–1278. ACM, 2014.

[103] E. Omar, S. Ghosh, and D. Whitley. HOMAJ: A tool for higher order mutation testing in AspectJ and Java. In *IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 165–170. IEEE, 2014.

[104] E. Omar, S. Ghosh, and D. Whitley. Subtle higher order mutants. *Information and Software Technology*, 81:3–18, 2017.

[105] M. Papadakis and N. Malevris. An empirical evaluation of the first and second order mutation testing strategies. In *third International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 2010*, pages 90–99. IEEE, 2010.

[106] M. Papadakis, N. Malevris, and M. Kallia. Towards automating the generation of mutation tests. In *Proceedings of the 5th Workshop on Automation of Software Test*, AST, pages 111–118, 2010.

[107] V. Pareto. Manuel D'économie Politique, Paris. *Ams Press*, 1927.

[108] D.H. Phan and J. Suzuki. R2-IBEA: R2 indicator based evolutionary algorithm for multiobjective optimization. In *IEEE Congress on Evolutionary Computation*, pages 1836–1845, June 2013.

[109] T. Pohlert. *The Pairwise Multiple Comparison of Mean Ranks Package (PMCMR)*, 2014. R package.

[110] M. Polo, M. Piattini, and I. García-Rodríguez. Decreasing the cost of mutation testing with second-order mutants. *Software Testing, Verification and Reliability*, 19(2):111–131, 2009.

[111] J. A. Prado Lima, G. Guizzo, S. R. Vergilio, A. P. C. Silva, H. L. Jakubovski Filho, and H. V. Ehrenfried. Evaluating Different Strategies for Reduction of Mutation Testing Costs. In *Proceedings of the 1st Brazilian Symposium on Systematic and Automated Software Testing*, SAST, pages 4:1–4:10, New York, NY, USA, 2016. ACM.

[112] R. S. Pressman. *Engenharia de software*. McGraw Hill Brasil, 2011.

[113] R. Purushothaman and D. E. Perry. Toward understanding the rhetoric of small source code changes. *IEEE Transactions on Software Engineering*, 31(6):511–526, 2005.

[114] J. Radatz, A. Geraci, and F. Katki. IEEE standard glossary of software engineering terminology. *IEEE Std*, 610121990(121990):3, 1990.

[115] O. Räihä. A survey on search-based software design. *Computer Science Review*, 4(4):203–249, 2010.

[116] S. Rapps and E. J. Weyuker. Selecting software test data using data flow information. *IEEE Transactions on Software Engineering*, (4):367–375, 1985.

[117] H. A. Richard, R. A. Demillo, B. Hathaway, W. Hsu, W. Hsu, E. W. Krauser, R. J. Martin, A. P. Mathur, and E. H. Spafford. Design of mutant operators for the C programming language. Technical report, Software Engineering Research Center, Department of Computer Science, Purdue University, Indiana, 1989.

[118] N. R. Sabar, M. Ayob, G. Kendall, and R. Qu. Grammatical evolution hyper-heuristic for combinatorial optimization problems. *IEEE Transactions on Evolutionary Computation*, 17(6):840–861, 2013.

[119] D. Schuler and A. Zeller. Javalanche: efficient mutation testing for Java. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 297–298. ACM, 2009.

[120] S. S. Shapiro and M. B. Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3/4):591–611, 1965.

[121] A. Strickler, J. A. Prado Lima, S. R. Vergilio, and A. T. R. Pozo. Deriving products for variability test of Feature Models with a hyper-heuristic approach. *Applied Soft Computing*, 2016.

[122] Team, R. C. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2016.

[123] H. C. Thode. *Normality Tests*, pages 999–1000. Springer, Berlin, Heidelberg, 2011.

[124] R. H. Untch, A. J. Offutt, and M. J. Harrold. Mutation analysis using mutant schemata. In *ACM SIGSOFT Software Engineering Notes*, volume 18, pages 139–148. ACM, 1993.

[125] D. A. Van Veldhuizen. Multiobjective evolutionary algorithms: classifications, analyses, and new innovations. Technical report, DTIC Document, 1999.

[126] A. Vargha and H. D. Delaney. A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics*, 25(2):101–132, January 2000.

[127] L. While, L. Bradstreet, and L. Barone. A fast way of calculating exact hypervolumes. *IEEE Transactions on Evolutionary Computation*, 16(1):86 –95, February 2012.

[128] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[129] M. R. Woodward and K. Halewood. From weak to strong, dead or alive? an analysis of some mutation testing issues. In *Proceedings of the Second Workshop on Software Testing, Verification, and Analysis, 1988*, pages 152–158. IEEE, 1988.

[130] E. Zitzler. *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*. PhD thesis, ETH Zurich, Switzerland, 1999.

[131] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: improving the strength pareto evolutionary algorithm. Technical report, Department of Electrical Engineering, Swiss Federal Institute of Technology, Zurich, Switzerland, 2001.

[132] E. Zitzler and L. Thiele. Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, 1999.

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