WESLEY KLEWERTON GUEZ ASSUNÇÃO

UMA ABORDAGEM PARA INTEGRAÇÃO E TESTE DE MÓDULOS BASEADA EM AGRUPAMENTO E ALGORITMOS DE OTIMIZAÇÃO MULTIOBJETIVOS

CURITIBA

Abril de 2012

WESLEY KLEWERTON GUEZ ASSUNÇÃO

UMA ABORDAGEM PARA INTEGRAÇÃO E TESTE DE MÓDULOS BASEADA EM AGRUPAMENTO E ALGORITMOS DE OTIMIZAÇÃO MULTIOBJETIVOS

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

CURITIBA

Abril de 2012

Assunção, Wesley Klewerton Guez
Uma abordagem para integração e teste de módulos baseada em agrupamento e algoritmos de otimização multiobjetivos / Wesley Klewerton Guez Assunção. – Curitiba, 2012.
110 f. : il.; graf., tab.

Dissertação (mestrado) – Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática.
Orientadora: Silvia Regina Vergilio

1. Software -- Verificação. 2. Algoritmos genéticos. I. Vergilio, Silvia Regina. II. Título.

CDD 005.14

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

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa de Dissertação de Mestrado em Informática, do aluno Wesley Klewerton Guez Assunção, avaliamos o trabalho intitulado, *"Uma abordagem para integração e teste de módulos baseada em agrupamento e algoritmos de otimização multiobjetivos"*, cuja defesa foi realizada no dia 18 de abril de 2012, às 09:30 horas, no Departamento de Informática do Setor de Ciências Exatas da Universidade Federal do Paraná. Após a avaliação, decidimos pela aprovação do candidato.

Curitiba, 18 de abril de 2012.

Profa. Dra. Silvia Regina Vergilio
DINF/UFPR – Orientador

Profa. Dra. Itana Maria Gimenes
UEM – Membro Externo

Profa. Dra. Aurora Trindad Ramirez Pozo
DINF/UFPR – Membro Interno

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

CONTEÚDO

ARTIGOS PRODUZIDOS

1. COLANZI, T. E.; ASSUNÇÃO, W. K. G.; POZO, A. T. R.; VENDRAMIN, A. C. B. K.; PEREIRA, D. A. B.; PAULA FILHO, P. L. ; ZORZO, CARLOS ALBERTO. Application of Bio-inspired Metaheuristics in the Data Clustering Problem. CLEI Electronic Journal, v. 14, p. 5, 2011.

2. ASSUNÇÃO, W. K. G.; TRINDADE, D. F. G.; COLANZI, T. E.; VERGILIO, S. R. Evaluating test reuse of a software product line oriented strategy. In: 12th IEEE Latin-American Test Workshop (LATW), 2011, Porto de Galinhas - PE.

3. ASSUNÇÃO, W. K. G. ; COLANZI, T. E. ; POZO, A. T. R. ; VERGILIO, S. R. . Establishing Integration Test Orders of Classes with Several Coupling Measures. In: 2011 Genetic and Evolutionary Computation Conference (GECCO), 2011, Dublin, Ireland. Proceedings of the 13th annual conference on Genetic and evolutionary computation (GECCO'2011). New York, NY, USA : ACM, 2011. p. 1867-1874.

4. ASSUNÇÃO, W. K. G.; COLANZI, T. E.; VERGILIO, S. R.; POZO, A. T. R. Estabelecendo Sequências de Teste de Integração de Classes: Um Estudo Comparativo da Aplicação de Três Algoritmos Evolutivos Multiobjetivos. In: XXIX Simpósio Brasileira de Redes de Computadores e Sistemas Distrubuídos (SBRC) - XII Workshop de Testes e Tolerância a Falhas (WTF), 2011, Campo Grande - MT.

5. ASSUNÇÃO, W. K. G.; COLANZI, T. E.; POZO, A. T. R.; VERGILIO, S. R. Reduzindo o Custo do Teste de Integração com Algoritmos Evolutivos Multiobjetivos e Diferentes Medidas de Acoplamento. In: XXXI Congresso da Sociedade Brasileira de Computação - VIII Encontro Nacional de Inteligência Artificial (ENIA), 2011, Natal.

6. COLANZI, T. E.; ASSUNÇÃO, W. K. G.; VERGILIO, S. R.; POZO, A. T. R. Integration Test of Classes and Aspects with a Multi-Evolutionary and Coupling-Based

Approach. In: International Symposium on Search Based Software Engineering (SSBSE), 2011, Szeged.

7. COLANZI, T. E.; ASSUNÇÃO, W. K. G.; VERGILIO, S. R.; POZO, A. T. R. Generating Integration Test Orders for Aspect-Oriented Software with Multi-objective Algorithms. In: V Workshop Latino-Americano em Desenvolvimento de Software Orientado a Aspectos (LA-WASP), 2011, São Paulo - SP. Proceedings of the Congresso Brasileiro de Software: Teoria e Prática (CBSoft), 2011.

8. ASSUNÇÃO, W. K. G.; COLANZI, T. E.; POZO, A. T. R.; VERGILIO, S. R. Uma Avaliação do Uso de Diferentes Algoritmos Evolutivos Multiobjetivos para Integração de Classes e Aspectos. In: II Workshop de Engenharia de Software Baseada em Buscas (WESB), 2011, São Paulo - SP. Proceedings of the Congresso Brasileiro de Software: Teoria e Prática (CBSoft), 2011.

9. COLANZI, T. E.; ASSUNÇÃO, W. K. G.; VENDRAMIN, A. C. B. K.; PEREIRA, D. A. B.; POZO, A. T. R. Empirical Studies on Application of Genetic Algorithms and Ant Colony Optimization for Data Clustering. In: XXIX International Conference of the Chilean Computer Science Society (SCCC), 2010, Antofagasta. p. 1-10.

10. ASSUNÇÃO, W. K. G.; VERGILIO, S. R. Uma Abordagem para Integração e Teste de Módulos Baseada em Agrupamento e Algoritmos de Otimização Multiobjetivos. In: I Workshop de Teses e Dissertações do CBSoft (WTDSoft), 2011, São Paulo.

11. VERGILIO, S. R.; COLANZI, T. E.; POZO, A. T. R.; ASSUNÇÃO, W. K. G. Search Based Software Engineering: Review and Analysis of the Field in Brazil. In: 25th Brazilian Symposium on Software Engineering (SBES), 2011, São Paulo - SP. Proceedings of the 25th Brazilian Symposium on Software Engineering. Washington, DC : IEEE Computer Society, 2011. p. 50-55.

12. COLANZI, T. E.; ASSUNÇÃO, W. K. G.; TRINDADE, D. F. G.; ZORZO, C. A.; VERGILIO, S. R. Evaluating different strategies for testing software product lines.

Journal of Electronic Testing: Theory and Applications (JETTA). Submetido para publicação, 2011.

13. ASSUNÇÃO, W. K. G.; COLANZI, T. E.; VERGILIO, S. R.; POZO, A. T. R. Generating Integration Test Orders for Aspect Oriented Software with Multi-objective Algorithms. Revista de Informática Teórica e Aplicada (RITA). Submetido para publicação, 2012.

14. COLANZI, T. E.; VERGILIO, S. R.; ASSUNÇÃO, W. K. G.; POZO, A. T. R. Search Based Software Engineering: Review and Analysis of the Field in Brazil. Journal of Systems and Software (JSS) - Special Issue: Software Engineering in Brazil. Submetido para publicação, 2012.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

A Medida de acoplamento referente ao número de atributos

AG Algoritmo Genético

AJATO *AspectJ and Java Assessment Tool*

BCEL *Byte Code Engineering Library*

C Indicador de qualidade Cobertura

CAITO *Class and Aspect Integration an Test Order*

CITO *Class and Integration Test Order*

COS *Class and Order System*

ED Indicador de qualidade Distância Euclidiana da Solução Ideal

GD Indicador de qualidade Distância Geracional

GUIMOO *Graphical User Interface for Multi Objective Optimization*

IGD Distância Geracional Invertida

JMetal *Metaheuristic Algorithms in Java*

MECBA *Multi-Evolutionary and Coupling-Based Approach*

MECBA-Clu *Multi-Evolutionary and Coupling-Based Approach with Clusters*

MOABC *Multi-Objective Artificial Bee Colony*

MOEA *Multi-Objective Evolutionary Algorithm*

MOPSO *Multi-Objective Particle Swarm Optimization*

MTABU *Multi-Objective Tabu Search*

NSGA-II *Non-dominated Sorting Genetic Algorithm*

O Medida de acoplamento referente ao número de operações

OA Orientação a Aspectos

OO Orientação a Objetos

ORD *Object Relation Diagram*

P Medida de acoplamento referente ao número de tipos distintos de parâmetros

PACO *Pareto Ant Colony*

PAES *Pareto Archived Evolution Strategy*

$\mathbf{PF}_{approx}$ Aproximação da Fronteira de Pareto Real

$\mathbf{PF}_{known}$ Fronteira de Pareto Conhecida

$\mathbf{PF}_{true}$ Fronteira de Pareto Real

R Medida de acoplamento referente ao número de tipos distintos de retornos

SCC *Strongly Connected Component*

SPEA2 *Strength Pareto Evolutionary Algorithm*

RESUMO

Para encontrar defeitos de comunicação entre diferentes partes de um sistema é realizado o teste de integração, no qual cada módulo desenvolvido deve ser integrado e testado com os módulos já existentes. Entretanto, um módulo a ser integrado e testado, pode necessitar de recursos de um outro módulo ainda em desenvolvimento, levando a necessidade de se construir um *stub*. *Stubs* são simulações de recursos essenciais para o teste mas que ainda não estão disponíveis. O *stub* não faz parte do sistema, então a construção de *stubs* implica em custo adicional. Para minimizar a necessidade de *stubs* e consequentemente reduzir o custo do projeto, várias estratégias para integrar e testar módulos foram propostas. Porém, nenhuma dessas estratégias considera uma característica presente na maioria dos sistemas, que é a modularização. Dado este fato, este trabalho propõe uma estratégia que considera agrupamentos de módulos durante o estabelecimento de ordens para a integração e teste. Esta estratégia é implementada em uma abordagem chamada MECBA-Clu, uma abordagem baseada em algoritmos de otimização multiobjetivos e diferentes medidas de acoplamento para avaliar diversos fatores que influenciam o custo de construção de *stubs*. A abordagem MECBA-Clu é avaliada através da condução de um experimento com oito sistemas reais, quatro Orientados a Objetos e quatro Orientados a Aspectos, no qual os três diferentes algoritmos evolutivos multiobjetivos NSGA-II, SPEA2 e PAES foram aplicados. Os resultados apontam que o espaço de busca fica restrito a determinadas áreas em que as soluções podem ser encontradas. Além disso, de acordo com quatro indicadores de qualidade utilizados, observa-se que o algoritmo PAES obteve o melhor resultado, seguido pelo NSGA-II e por fim o SPEA2. Exemplos da utilização da abordagem também são apresentados.

Palavras-chave: *teste de integração, algoritmos evolutivos multiobjetivos, medidas de acoplamento, modularização.*

ABSTRACT

In the integration test phase, the modules are integrated and tested in order to find communication faults. However, a module to be integrated and tested may require resources from another module still under development. In such cases, a stub is required. Stubs are simulations of essential resources, which are not available yet for testing a module. The stub is not part of the system, then the stubbing construction involves additional cost. To minimize stubbing costs, several strategies have been proposed to integrate and test modules. However, no one of them considers a characteristic of most systems, the modularization. Therefore, this work proposes a strategy that considers clusters of modules during the establishment of integration and testing orders. This strategy is implemented in an approach, called MECBA-Clu, which considers multi-objective optimization algorithms and different coupling measures to evaluate several factors that influence the stubbing construction cost. The MECBA-Clu approach is evaluated through an experimental study with eight real systems, four Object-Oriented and four Aspect-Oriented ones, where three different multi-objective evolutionary algorithms, NSGA-II, SPEA2 and PAES, were applied. The results show that the search space is restricted to certain areas where the solutions can be found. In addition to this, by using four quality indicators, we can attest that the algorithm PAES has the best performance, followed by NSGA-II and finally SPEA2. Usage examples of the approach are also presented.

Keywords: *integration testing, multi-objective evolutionary algorithms, coupling measures, modularization.*

CAPÍTULO 1

INTRODUÇÃO

A Engenharia de Software é uma área que visa à especificação, desenvolvimento e manutenção de sistemas computacionais, com objetivo de garantir qualidade e produtividade [45]. Para o desenvolvimento de software de qualidade, o teste de software é considerado uma atividade fundamental. Como apresentado por Pressman [45], a atividade de teste é complexa e tem grande impacto no custo final de um projeto de software, portanto estratégias para reduzir este custo devem ser adotadas.

Geralmente, a atividade de teste é conduzida de maneira incremental, na qual primeiramente, realiza-se o teste dos módulos isoladamente e depois estes módulos são integrados, visando a revelar defeitos de comunicação entre as várias partes que compõem o software. A etapa de teste de integração pode ser conduzida de maneira diferente dependendo do contexto de desenvolvimento de software. No contexto de Orientação a Objetos (OO), os módulos a serem integrados são classes [29]. No contexto de Orientação a Aspectos (OA) os módulos são tanto classes quanto aspectos [37].

Recomenda-se que a atividade de teste seja conduzida paralelamente ao desenvolvimento, porém existem situações em que para se testar a comunicação entre dois módulos são necessários recursos de um terceiro módulo, mas o módulo que fornece o recurso requerido pode ainda estar em desenvolvimento. Portanto, para realizar o teste de integração, muitas vezes é necessário estabelecer uma ordem para a integração e teste dos módulos. Esta ordem é importante porque ela interfere na ordem na qual os módulos serão desenvolvidos, na qual serão elaborados casos de teste e na qual serão revelados defeitos de integração.

Para estabelecer a ordem para integrar e testar os módulos, em alguns casos é possível considerar as dependências entre os módulos mediante um grafo direcionado e, a partir do grafo, efetuar uma ordenação topológica. Porém, a maioria dos sistemas desenvolvi-

dos no contexto OO, e consequentemente OA, apresentam ciclos de dependências entre os módulos [41], o que dificulta o estabelecimento da sequência de módulos. Nestes casos, para integrar e testar um módulo por vez é preciso quebrar os ciclos, o que leva a necessidade de construir *stubs* associados à dependência quebrada. *Stubs* são pseudo-implementações de recursos ainda em desenvolvimento. Então, quando um módulo A, em teste, necessita de recursos de outro módulo B do qual A depende, mas que ainda está em desenvolvimento, um *stub* emulando o comportamento de B precisa ser construído. A construção dos *stubs* implica em custo adicional. Portanto, uma estratégia que visa a minimizar o custo de construção de *stubs* deve ser aplicada.

Contudo, decidir qual dependência deve ser quebrada, o que implica em decidir quais *stubs* devem ser construídos, não é tarefa trivial. Este problema ainda é agravado pelas restrições diferentes e conflitantes relativas ao desenvolvimento do sistema. Há diversos fatores que influenciam a construção de *stubs*: a complexidade dos *stubs*, questões contratuais e aspectos do desenvolvimento de software que devem ser considerados. Isto levou a pesquisadores proporem diferentes soluções.

Os primeiros trabalhos para tratar este problema apresentam soluções baseadas em algoritmos de grafos [10, 36, 50, 51, 53, 55]. A ideia desses trabalhos é quebrar os ciclos de dependências que levam a construção de um número reduzido de *stubs*. Porém, estes algoritmos não apresentam soluções satisfatórias, pois muitas vezes produzem somente ótimos locais, já que não analisam a consequência da quebra de determinado ciclo [8]. Existem situações em que quebrar dois ciclos pode exigir custo de construção de *stubs* menor do que quebrar somente um. Outra desvantagem destas abordagens é a dificuldade para serem adaptadas para considerar os vários fatores, já mencionados acima, envolvidos na construção de *stubs*

Para superar essas limitações, alguns autores propuseram o uso de Algoritmos Genéticos (AG) para solucionar o problema [8, 9, 24]. O uso de AG apresenta resultados promissores quando comparados às estratégias tradicionais baseadas em algoritmos de grafos. Estes algoritmos trabalham com uma função objetivo formada pela agregação de duas medidas que refletem a complexidade dos *stubs*: número de atributos e número de métodos.

Entretanto, para utilizar um AG com agregação de medidas é necessário definir pesos com o objetivo de priorizar determinada medida e ao mesmo tempo permitir uma exploração eficiente do espaço de busca, uma atividade que consome bastante esforço. Outra desvantagem do AG com agregação de medidas, é que apenas uma solução é encontrada em cada execução do algoritmo, mas pelas características do problema é possível verificar que várias soluções são possíveis. Essas situações motivaram a utilização de algoritmos multiobjetivos para solucionar o problema de integração e teste de módulos [3, 6, 5, 11, 17, 18, 59]. Algoritmos multiobjetivos são meta-heurísticas que aplicam o conceito de dominância de Pareto [43], no qual cada medida é considerada um objetivo a ser otimizado simultaneamente, sem a necessidade de ajuste de pesos.

Os algoritmos multiobjetivos apresentam resultados melhores que os AGs com agregação de medidas [3, 6, 5, 11, 17, 18, 59], além de oferecer um conjunto de boas soluções para o problema, visto que nem sempre é possível encontrar uma única solução ótima que satisfaça todas as restrições.

Dentre os trabalhos que utilizam algoritmos multiobjetivos, destaca-se uma abordagem chamada MECBA (*Multi-Evolutionary and Coupling-Based Approach* [17]) que propõe um conjunto de etapas genéricas que levam a solução do problema de integração e teste de módulos. Estas etapas são formuladas de maneira a permitir que a abordagem seja instanciada para diferentes contextos, como por exemplo, teste de programas OO e OA.

1.1 Motivação

Apesar de existirem vários trabalhos que propõem diferentes estratégias para integrar e testar módulos, nenhum deles considera uma característica presente no desenvolvimento de software: o agrupamento de módulos ou modularização [12, 62]. O agrupamento de módulos corresponde a várias classes ou aspectos, dependendo do contexto, que são organizadas em grupos e que geralmente devem ser desenvolvidas e testadas em conjunto por uma mesma equipe em um mesmo espaço de tempo.

Segundo alguns autores [12, 45, 52, 62], o agrupamento de módulos é amplamente utilizado para facilitar a organização, o desenvolvimento e a manutenção do software. Tais

grupos podem ser definidos com base em características da arquitetura do software, como por exemplo: componentes fortemente relacionadas [14, 45], por questões organizacionais e contratuais [12] ou devido ao desenvolvimento distribuído [46].

Esta limitação encontrada nos trabalhos relacionados, de não considerarem os agrupamentos de módulos, constitui a motivação para o presente trabalho, que propõe uma abordagem que considera os grupos de módulos durante o estabelecimento de ordens de integração e teste. Portanto, da mesma maneira que um software é desenvolvido e mantido por grupos de módulos, ele também deve ser integrado e testado considerando estes grupos.

1.2 Objetivos

Este trabalho tem por objetivo propor e implementar uma estratégia para estabelecer ordens de teste que considerem agrupamentos de módulos, no contexto de desenvolvimento de software OO e OA. Para permitir a implementação da estratégia, esta foi incorporada em uma extensão da abordagem MECBA [17], chamada de MECBA-Clu (*Multi-Evolutionary and Coupling-Based Approach with Clusters*).

Assim como a MECBA, a abordagem MECBA-Clu é baseada em algoritmos de otimização multiobjetivo e em um conjunto de métricas que formam um modelo de custo associado à construção dos *stubs*. Na MECBA-Clu os grupos de módulos que devem ser integrados e testados em conjunto são fornecidos pelo usuário e restrições com relação a estes grupos são implementadas pelos algoritmos de otimização multi-objetivo.

Para avaliar a abordagem proposta, um experimento foi conduzido com oito sistemas reais, quatro do contexto de OO e quatro do contexto de OA. Do experimento foram obtidos resultados relativos ao impacto no espaço de busca quando os grupos de módulos são considerados em comparação com a abordagem MECBA. Foi também realizada uma comparação entre três diferentes algoritmos evolutivos: NSGA-II, SPEA2 e PAES. Além disso, um exemplo de uso da abordagem é descrito utilizando resultados do exeprimento.

1.3 Organização do Trabalho

No Capítulo 2 são apresentados os conceitos fundamentais sobre a otimização multiobjetivo e descritos os algoritmos e indicadores de qualidade utilizados neste trabalho. No Capítulo 3 é contextualizado o teste de software, englobando seus objetivos, suas principais técnicas e fases. O Capítulo 4 é utilizado para fundamentar o problema de integração e teste de módulos, expondo as estratégias utilizadas para a sua solução. Dentre as estratégias, são apresentadas primeiramente as tradicionais baseadas em grafos e posteriormente as baseadas em meta-heurísticas. A abordagem proposta para tratar dos agrupamentos de módulos, chamada MECBA-Clu, é descrita no Capítulo 5, apresentando suas características, funcionamento e aspectos de implementação. A abordagem MECBA-Clu é avaliada no Capítulo 6, em um experimento aplicando três algoritmos evolutivos multiobjetivos para estabelecer ordens de integração e teste de oito sistemas reais. Por fim, os conclusões do trabalho são apresentadas no Capítulo 7, juntamente com as sugestões de trabalhos futuros.

CAPÍTULO 2

OTIMIZAÇÃO MULTIOBJETIVO

Problemas do mundo real, em geral, apresentam alto grau de complexidade para serem solucionados, devido à grande quantidade de variáveis que devem ser analisadas e restrições que precisam ser respeitadas. Métodos de otimização procuram solucionar matematicamente tais problemas a fim de encontrar soluções aceitáveis, uma vez que uma solução ótima é desconhecida ou custosa para ser alcançada.

Muitos casos permitem um nível de abstração que possibilita representar o escopo do problema através de um modelo que simplifique seu tratamento, reduzindo o espaço de busca e a influência de fatores externos ao objetivo principal do problema. Ao final obtém-se um modelo com objetivo único e bem definido que permite tratar um vetor n-dimensional $X = [x_1, x_2, ..., x_n]$ de variáveis de decisão, pertencentes ao universo de busca Ω, que são avaliadas mediante uma função objetivo $f(X)$, possibilitando avaliar o grau de aceitação da solução [16]. Estes problemas são chamados mono-objetivo.

Porém, existem problemas de natureza mais complexa, conhecidos como multiobjetivos, que exigem a otimização simultânea de vários interesses interdependentes e conflitantes, que diferentemente dos problemas citados acima envolvem a manipulação de várias funções objetivo. Ou seja, têm-se como objetivo otimizar uma solução avaliada por um conjunto de funções $F(X) = [f_1(X), f_2(X), ..., f_k(X)]$ para um problema que apresenta k interesses para serem otimizados simultaneamente [21].

Ambos problemas, mono e multiobjetivo, podem apresentar restrições de desigualdade ou de igualdade, apresentadas respectivamente nas Equações 2.1 e 2.2. O número de restrições de igualdade p precisa obrigatoriamente ser menor que n número de variáveis de decisões, pois se $p \geq n$ o problema não apresenta liberdade suficiente para que suas soluções sejam otimizadas [16].

$$g_i(X) \leq 0 \quad i = 1, 2, ..., m \tag{2.1}$$

$$h_j(X) = 0 \quad j = 1, 2, ..., p \tag{2.2}$$

A otimização multiobjetivo apresenta grande complexidade para análise das soluções encontradas quando comparada a otimização mono-objetivo, devido às diferentes características da função objetivo.

Comparar duas soluções U e V encontradas durante uma otimização mono-objetivo de minimização envolve somente comparar a relação $f(U) < f(V)$, permitindo a fácil identificação de qual é a melhor solução. Já em uma minimização com dois objetivos conflitantes pode ocorrer um situação no qual $f_1(U) < f_1(V)$ e simultaneamente $f_2(V) < f_2(U)$, ou seja, a solução U é melhor para o primeiro objetivo (f_1) porém a solução V é melhor para o segundo objetivo (f_2), caso em que as duas soluções podem ser consideradas boas.

Diante dessa situação é possível afirmar que na otimização mono-objetivo existe a possibilidade de se definir uma solução, entre várias encontradas, como a melhor para solucionar determinado problema, já na otimização multiobjetivo existe um conjunto de soluções que apresentam qualidade equivalente de solucionar o problema, priorizando de maneira diferente os muitos objetivos.

Para encontrar esse conjunto de soluções candidatas a solucionar um problema multiobjetivo utiliza-se o conceito de dominância de Pareto [43], que permite comparar soluções considerando todos os objetivos do problema. Este conceito baseia-se na relação de dominância entre soluções, possibilitando afirmar que para um problema de minimização uma solução é melhor que outra, ou que U domina V representada pela notação $U \prec V$, quando são satisfeitas as condições da Equação 2.3.

$$\forall i \in \{1, 2, ..., k\}, \;\; f_i(U) \leq f_i(V) \quad \wedge \quad \exists i \in \{1, 2, ..., k\}, \;\; f_i(U) < f_i(V) \tag{2.3}$$

As duas condições da Equação 2.3 mostram que uma solução U domina V para um problema de minimização, mas que funciona analogamente para um problema de maximização, somente quando: (i) todos os valores de objetivos de U forem menores ou iguais aos valores de objetivos de V; e (ii) existir pelo menos um valor de objetivo em U que seja menor que seu correspondente valor em V.

Durante o processo de otimização multiobjetivo deseja-se encontrar o conjunto com todas as soluções não dominadas para um problema, este conjunto é chamado Pareto Ótimo. Porém, na maioria das vezes este conjunto é desconhecido ou impossível de ser encontrado, sendo este um problema NP-hard [32]. Diante disso, geralmente o conjunto de soluções encontradas durante o processo de otimização multiobjetivo é uma aproximação do conjunto Pareto Ótimo, representado por PF_{approx}.

Para solucionar problemas multiobjetivos, ou seja, obter o conjunto PF_{approx}, existem fundamentalmente três técnicas [16]: (i) priorizar somente aquele objetivo que é considerado como prioritário; (ii) utilizar agregação ponderada de objetivos através da definição de pesos para cada objetivo; e (iii) aplicar algoritmos de busca multiobjetivos que encontram conjuntos de soluções não dominadas. A primeira técnica, priorizar somente um objetivo, não garante que as soluções encontradas se aproximem do conjunto PF_{approx}, pois a solução é avaliada somente através de um dos objetivos, sendo que os outros objetivos podem estar distantes de possíveis pontos com melhores valores. A segunda técnica, agregação ponderada de objetivos, considera que cada solução será comparada com outras soluções segundo o resultado de uma função que agrega os objetivos, cada qual com seu peso. Esta técnica é melhor do que a primeira, entretanto o ajuste dos pesos deve ser feito a fim de permitir que o espaço de busca seja explorado de forma eficiente ao mesmo tempo que o objetivo desejado deve ser priorizado, o que não é uma atividade trivial. Tanto a primeira quanto a segunda técnicas, apesar de tratarem de dois ou mais objetivos, utilizam um único valor final para comparar as soluções encontradas, o que pode resultar em somente uma solução ser considerada como a melhor, o que não condiz com as características de um problema multiobjetivo. Portanto, a terceira técnica, utilizar algoritmos de busca multiobjetivos, visa a considerar os objetivos independentemente,

utilizando o conceito de dominância de Pareto para encontrar um conjunto de possíveis soluções com diferente balanceamento entre os objetivos.

Algoritmos de busca são técnicas de otimização que permitem selecionar uma melhor solução dentro de um conjunto de possíveis soluções. Estes algoritmos são divididos em dois grupos principais [19, 58]. O primeiro grupo inclui técnicas clássicas do campo da pesquisa operacional, tais como o algoritmo *branch and bound* e a programação linear. As técnicas clássicas são em geral determinísticas. O segundo grupo inclui as meta-heurísticas, utilizadas principalmente para resolver problemas que não podem ser representados por equações matemáticas. Este grupo inclui Otimização por Nuvem de Partículas, Otimização por Colônia de Formigas, Algoritmos Evolutivos, e muitos outros.

Dentre os algoritmos de busca utilizados para solucionar problemas de otimização multiobjetivo, os Algoritmos Evolutivos têm se mostrado promissores [16] e estão sendo amplamente utilizados na área de Engenharia de Software [19, 28, 58]. Diante disso, estes algoritmos foram utilizados no presente trabalho. Uma introdução breve de Algoritmos Evolutivos é apresentada na seção a seguir e posteriormente os Algoritmos Evolutivos Multiobjetivos são descritos.

2.1 Algoritmos Evolutivos

Algoritmos Evolutivos são programas de computador que se baseiam na genética e na evolução biológica de seres vivos para encontrar soluções de problemas de otimização [26], apresentando-se como uma alternativa viável para otimização de problemas complexos.

Estes algoritmos trabalham com o conceito de população de indivíduos, no qual cada indivíduo representa uma solução candidata para determinado problema. Esta população de indivíduos é utilizada para gerar novas soluções que vão compor uma nova população. Esta nova população constitui uma nova geração de indivíduos.

A criação de novos indivíduos é efetuada a partir da aplicação dos operadores genéticos de seleção, cruzamento e mutação. O operador de seleção escolhe os melhores indivíduos da população para gerarem descendentes que comporão uma nova geração de indivíduos. O operador de cruzamento é responsável por gerar filhos a partir de dois pais, escolhidos

pelo operador de seleção, combinando as características genéticas das soluções. O operador de mutação faz modificações aleatórias em um indivíduo, escolhido pelo operador de seleção, a fim de obter uma diversidade de soluções dentro da população.

Os operadores de seleção, cruzamento e mutação são aplicados até gerarem um número determinado de indivíduos, conforme parametrizado. Dentre os indivíduos da população atual, e os indivíduos gerados pelos operadores, é adotada um regra para definir quais os indivíduos sobreviverão para a próxima geração. Para preservar os melhores indivíduos durante o processo evolutivo, é possível aplicar um método que copia as melhores soluções para a próxima geração, este método é chamado de elitismo.

Para verificar o quão boa é uma solução para resolver um problema, utiliza-se uma função de aptidão (*fitness*) para determinar quais indivíduos sobrevivem para uma nova geração, descartando-se as piores soluções. De geração em geração segue o processo evolutivo até que determinado critério de parada seja satisfeito, como por exemplo, um número máximo de avaliações da função de aptidão ou um número máximo de gerações.

2.2 Algoritmos Evolutivos Multiobjetivos

Os algoritmos evolutivos são aplicados com sucesso em diversos problemas mono-objetivos, levando pesquisadores a criarem adaptações destes algoritmos para lidar com problemas multiobjetivos, o que originou uma nova categoria de meta-heurísticas chamadas Algoritmos Evolutivos Multiobjetivos, do inglês MOEAs (*Multi-objective Evolutionary Algorithms* [16]). Além dos Algoritmos Evolutivos, outros tipos de meta-heurísticas também foram adaptadas para lidar com problemas multiobjetivos, tais como: nuvem de partículas e evolução diferencial [23].

A seguir, três Algoritmos Evolutivos Multiobjetivos são descritos: *Non-dominated Sorting Genetic Algorithm* [20], *Strength Pareto Evolutionary Algorithm* [63], e *Pareto Archived Evolution Strategy* [33]. Estes algoritmos foram escolhidos para o desenvolvimento do presente trabalho. Eles são largamente utilizados na literatura e representam diferentes estratégias de evolução para lidar com os problemas de otimização multiobjetivo.

2.2.1 *Non-dominated Sorting Genetic Algorithm*

O *Non-dominated Sorting Genetic Algorithm* (NSGA-II) [20] é um algoritmo de otimização multiobjetivo baseado em algoritmos evolutivos e como característica principal apresenta forte estratégia de elitismo. O pseudocódigo do NSGA-II é apresentado no Algoritmo 2.1.

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

Algoritmo 2.1: Pseudocódigo do NSGA-II (Adaptado de [16])

A entrada do algoritmo NSGA-II, conforme observado no pseudocódigo, é composta de: tamanho da população "N'", número de gerações "g" e as funções a serem otimizadas "$f_k(X)$", onde k corresponde ao número de objetivos a serem otimizados.

Em cada geração o algoritmo NSGA-II ordena os indivíduos das populações de pais e filhos de acordo com a dominância entre as soluções, formando diversas fronteiras (linhas 10 e 11 do Algoritmo 2.1). A primeira fronteira é composta pelas soluções não dominadas de toda a população, a segunda é composta pelas soluções que passam a ser não dominadas após retiradas as soluções da primeira fronteira, a terceira fronteira é composta por soluções que passam a ser não dominadas após retiradas as soluções da primeira e segunda

fronteiras, e assim sucessivamente até todas as soluções estarem classificadas em alguma fronteira.

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

Ambas ordenações, de fronteiras e de distância de multidão, são usadas pelo operador de seleção (linhas 6 e 16 do Algoritmo 2.1) e para determinar os indivíduos que sobrevivem para a próxima geração. O NSGA-II utiliza a seleção por torneio, selecionando soluções de fronteiras com maior dominância e em caso de empate na dominação é utilizado como critério de desempate a distância de multidão.

A criação dos novos indivíduos é efetuada mediante a aplicação dos operadores de cruzamento e mutação (linhas 7 e 17 do Algoritmo 2.1).

O processo de ordenação em fronteiras, ordenação pela distância de multidão, e o elitismo são ilustrados na Figura 2.1, onde P_t é a população dos pais; Q_t é a população dos filhos; F_1, F_2 e F_3 são fronteiras de soluções já ordenadas da união de P_t e Q_t; e P_{t+1} representa o conjunto de soluções que serão usadas na próxima geração.

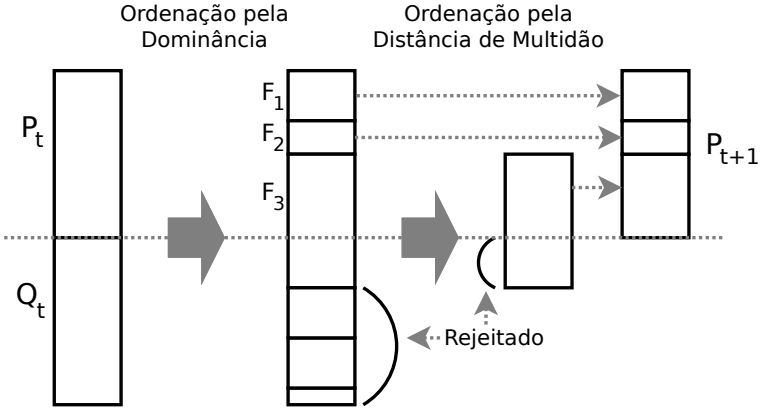


Figura 2.1: Diagrama de Funcionamento do Elitismo no NSGA-II (Adaptada de [20])

O algoritmo NSGA-II é um dos mais tradicionais algoritmos multiobjetivos baseados em algoritmos evolutivos, e é largamente utilizado em comparações com outros algoritmos [16, 18].

2.2.2 *Strength Pareto Evolutionary Algorithm*

Outro algoritmo multiobjetivo baseado em algoritmo evolutivo é chamado *Strength Pareto Evolutionary Algorithm* (SPEA2) [63]. Este algoritmo apresenta como principal diferença em relação ao algoritmo NSGA-II a forma de cálculo do *fitness* e a utilização de um arquivo externo, separado da população regular, que é utilizado para armazenar as soluções não dominadas encontradas durante o processo evolutivo. O pseudocódigo do SPEA2 é apresentado no Algoritmo 2.2.

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

Algoritmo 2.2: Pseudocódigo do SPEA2 (Adaptado de [16])

Os parâmetros de entrada, Algoritmo 2.2, são similares aos do algoritmos NSGA-II com adição do parâmetro correspondente ao tamanho do arquivo externo "$\overline{N}$".

Em cada geração do SPEA2 é calculado para todas as soluções um valor chamado de *strength* que é utilizado para definir o *fitness* da solução. O valor de *strength* de uma solução i corresponde ao número de indivíduos, pertencentes ao arquivo externo e a população regular, que dominam a solução i.

O *fitness* de uma solução i é calculado pela soma de todos os valores de *strength* das soluções dominadas por i, tanto do arquivo externo quanto da população regular (linha 4 do Algoritmo 2.2). Valor de *fitness* igual a 0 indica que um indivíduo não é dominado por nenhuma outra solução, por outro lado, valores altos de *fitness* representam soluções dominadas por vários outros indivíduos.

Como o arquivo externo tem um tamanho fixo determinado por parâmetro, então durante o preenchimento do arquivo externo duas situações podem ocorrer: o arquivo pode estar com mais soluções não dominadas que seu limite, então um operador de eliminação de soluções é efetuado (Algoritmo 2.2, linha 7), calculando-se a distância das soluções para seus vizinhos e removendo-se as mais próximas. Por outro lado, caso o número de soluções não dominadas seja menor que o tamanho do arquivo, então este é preenchido com soluções dominadas (Algoritmo 2.2, linha 9) . Somente os indivíduos que compõem o arquivo externo sobrevivem para uma próxima geração.

Para compor uma nova população, criação dos novos indivíduos é efetuada mediante a aplicação dos operadores de cruzamento e mutação em indivíduos selecionados da população e no arquivo externo (linha 12 do Algoritmo 2.2).

2.2.3 *Pareto Archived Evolution Strategy*

No processo evolutivo do algoritmo *Pareto Archived Evolution Strategy* (PAES) [33] o conceito de população é diferente das estratégias tradicionais de algoritmos evolutivos, pois apenas uma solução é mantida em cada geração. A estratégia para gerar novos indivíduos consiste em utilizar somente o operador de mutação, como pode ser observado na linha 3 do pseudocódigo apresentado no Algoritmo 2.3. Uma vez que o algoritmo trabalha com apenas uma solução por geração não existe possibilidade de utilizar o operador de cruzamento. Assim como no SPEA2, existe um arquivo externo de soluções que é populado com as soluções não dominadas encontradas durante o processo evolutivo.

Entrada: $f_k(X)$
1 **repita**
2 Inicializar população com um único pai C e adicionar para o arquivo $\mathbb{A}$
3 Mutar C para produzir um filho C' e avaliar seu *fitness*
4 **se** $C \prec C'$ **então**
5 Descartar C'
6 **senão se** $C \succ C'$ **então**
7 Substituir C por C', e adicionar C' para $\mathbb{A}$
8 **senão se** $\exists_{c'' \in \mathbb{A}}(C'' \prec C')$ **então**
9 Descartar C'
10 **senão**
11 Testar $(C, C', \mathbb{A})$ para determinar qual será a solução que continuará no processo evolutivo, possibilitando adicionar C' para $\mathbb{A}$
12 **fim**
13 **até** *atingir critério de parada* ;

Algoritmo 2.3: Pseudocódigo do PAES (Adaptado de [16])

A cada geração, o algoritmo PAES cria uma nova solução filho que é comparada com a solução pai, se a solução filho é dominada pela solução pai, a solução filho é descartada (Algoritmo 2.3, linhas 4 e 5), se a solução filho domina a solução pai, o filho toma o lugar do pai e o filho é acrescentado ao arquivo externo (linhas 6 e 7), se a solução filho for dominada por alguma solução do arquivo, o filho é descartado (linhas 8 e 9), e caso nenhuma das soluções (pai, filho e do arquivo) for dominante, a escolha da solução que vai permanecer no processo evolutivo é feita considerando a diversidade entre as soluções (linhas 10 e 11). É importante destacar que o pseudocódigo do algoritmo PAES apresentado no Algoritmo 2.3 representa uma otimização de minimização.

Caso o tamanho do arquivo externo seja excedido, é aplicada uma estratégia de diversidade sobre este conjunto de soluções, eliminando soluções similares, mantendo a exploração de um espaço de busca maior.

2.3 Indicadores de Qualidade

No contexto deste trabalho, indicadores de qualidade são medidas utilizadas para avaliar a qualidade das soluções resultantes do processo de otimização de problemas multiobjetivos, além de permitir a comparação do comportamento entre diferentes meta-heurísticas aplicadas em um mesmo contexto.

Para utilizar os indicadores de qualidade descritos a seguir, são necessários determinados conjuntos de soluções. Estes conjuntos são: PF_{approx}, PF_{known} e PF_{true}, que são definidos a seguir:

- PF_{approx}: este conjunto corresponde a uma aproximação do conjunto de Pareto Ótimo, que representa o conjunto de possíveis soluções não dominadas encontradas para determinado problema. Em cada execução de um MOEA, um conjunto PF_{approx} é obtido;

- PF_{known}: este conjunto corresponde às melhores soluções encontradas por determinado MOEA para um problema. Em geral um MOEA é executado várias vezes para que seu comportamento seja observado, e como em cada execução é obtido um conjunto PF_{approx}, o conjunto PF_{known} é obtido através da união de todos os PF_{approx} gerados por determinado MOEA, eliminam-se as soluções dominadas e repetidas;

- PF_{true}: este conjunto corresponde às melhores soluções conhecidas e obtidas computacionalmente para determinado problema. Como na maioria dos trabalhos que tratam de otimização multiobjetivo mais de um MOEA são utilizados, o conjunto PF_{true} pode ser obtido através da união de todos os PF_{known} de cada MOEA, eliminam-se as soluções dominadas e repetidas [65].

Nas seções a seguir são apresentados os indicadores considerados relevantes ao escopo deste trabalho, devido a serem amplamente utilizados e estarem implementados na maioria das ferramentas para análises de algoritmos multiobjetivos [23, 42].

2.3.1 Distância Geracional e Distância Geracional Invertida

O indicador de Distância Geracional (GD), traduzido do inglês *Generational Distance*, [56, 57] é usado para calcular a distância do conjunto PF_{approx} em relação ao conjunto PF_{true}. É uma medida de erro pelo qual verifica-se o quão distantes estão as soluções obtidas no conjunto PF_{approx}, de suas correspondentes mais próximas no conjunto PF_{true}.

Distância Geracional Invertida (IGD), do inglês *Inverted Generational Distance* [48] é um indicador baseado no indicador GD, porém com objetivo de calcular a distância do

conjunto PF_{true} em relação ao conjunto PF_{approx}, ou seja, observa-se o inverso de GD. No indicador IGD, para cada solução do conjunto PF_{true}, calcula-se a distância em relação a sua correspondente mais próxima do conjunto PF_{approx}.

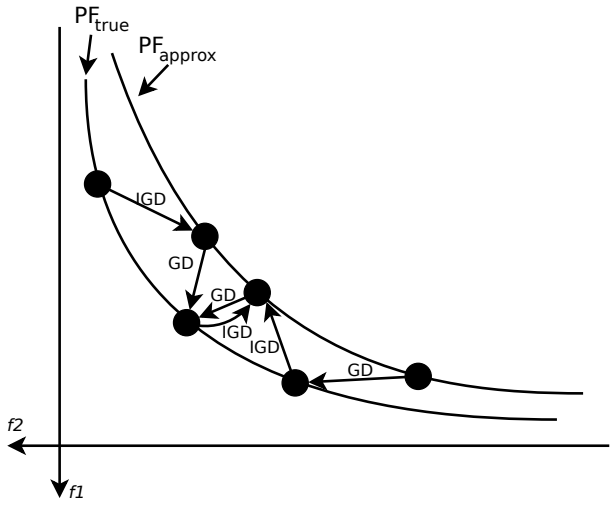


Figura 2.2: Exemplo dos Indicadores GD e IGD

A Figura 2.2 apresenta um exemplo dos indicadores GD e IGD. O resultado de cada um dos indicadores corresponde à soma de todas as medidas obtidas para cada solução do conjunto analisado.

Enquanto o indicador GD possibilita analisar o quão próximo está um conjunto PF_{approx} do conjunto PF_{true}, o indicador IGD traz a idéia do quão distante está um conjunto PF_{approx} do conjunto PF_{true} [61]. Apesar de ser possível serem utilizados individualmente, estes dois indicadores são complementares para uma análise mais completa e confiável.

Para estes dois indicadores deseja-se obter valores próximos a 0, portanto quanto menor melhor. O valor 0 indica que todas as soluções do conjunto PF_{approx} estão sobre soluções do conjunto PF_{true} para GD, ou que o conjunto PF_{approx} contém todas as soluções do conjunto PF_{true} para o indicador IGD.

2.3.2 Cobertura

O indicador de Cobertura (C), traduzido do inglês *Coverage* [33, 64], é usado para medir a dominância de Pareto entre dois conjuntos. Comparando-se C(PF_a, PF_b) obtém-se um valor entre 0 e 1 referente a quanto o conjunto PF_b é dominado pelo conjunto PF_a.

Similarmente analisa-se C(PF_b, PF_a) para obter o quanto o conjunto PF_a é dominado pelo conjunto PF_a.

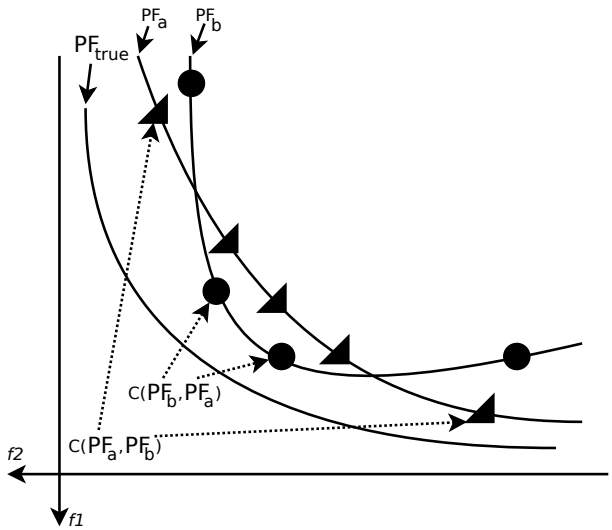


Figura 2.3: Exemplo do Indicador C

A Figura 2.3 apresenta um exemplo do indicador C, para um problema de minimização com dois objetivos, no qual analisam-se dois conjuntos PF_a e PF_b de cinco e quatro elementos, respectivamente. No primeiro caso analisa-se C(PF_a, PF_b) que retorna o valor 0,5 pois observa-se que o conjunto PF_a domina dois dos quatro elementos do conjunto PF_b, no segundo caso analisa-se C(PF_b, PF_a) que retorna o valor 0,6 pois o conjunto PF_b domina três dos cinco elementos do conjunto PF_a.

O valor retornado pelo indicador C, entre [0,1], corresponde a porcentagem de dominância de um conjunto em relação ao outro, pois se este valor for 0 indica que as soluções do primeiro conjunto não dominam nenhum elemento do segundo conjunto, por outro lado 1 indica que todos os elementos do segundo conjunto são dominados pelos elementos do primeiro conjunto. Portanto no exemplo da Figura 2.3 o conjunto PF_a domina 50% do conjunto PF_b e PF_b domina 60% PF_a.

2.3.3 Distância Euclidiana da Solução Ideal

Um último indicador considerado relevante para análise dos resultados consiste em encontrar as soluções mais próximas de uma solução definida como ideal. Para aplicar este indicador, inicialmente determinam-se os melhores valores para cada um dos objetivos

entre todas as soluções do conjunto PF_{true}, formando uma solução considerada como a solução ideal para o problema. A partir deste ponto, calcula-se a distância euclidiana (ED) em relação a todas as soluções do conjunto PF_{approx}.

O objetivo deste indicador é encontrar a solução que mais se aproxima dos objetivos ótimos, e consequentemente identificar o algoritmo que mais se aproximou da melhor solução possível para um determinado problema. A Figura 2.4 mostra um exemplo ilustrativo do cálculo do indicador ED para um problema de minimização com dois objetivos.

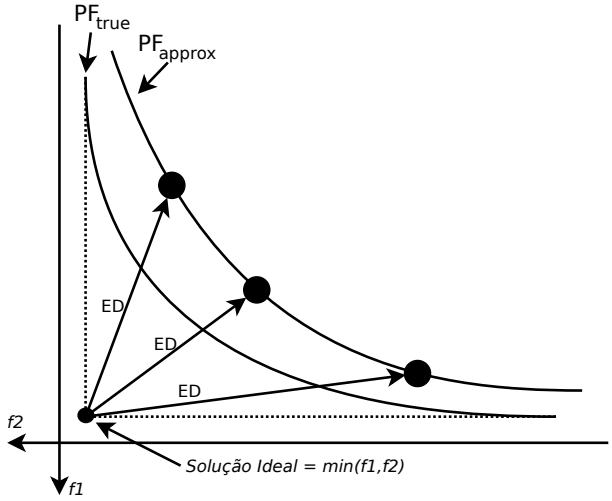


Figura 2.4: Exemplo do Indicador ED

Este indicador tem como base a técnica de classificação *Compromise Programming* [15]. Esta técnica é utilizada em otimização multiobjetivo como apoio para o tomador de decisão selecionar uma solução entre as várias encontradas.

2.4 Considerações Finais

Problemas multiobjetivos apresentam grande complexidade para serem solucionados devido às variáveis (objetivos) conflitantes do espaço de busca e às restrições associadas ao problema. Devido às suas características, existe a necessidade de novos conceitos para avaliar e comparar a qualidade das soluções encontradas.

Para solucionar este tipo de problema, algoritmos de busca (meta-heurísticas) foram adaptados para otimização multiobjetivo e são amplamente utilizados. Em Engenharia de Software esses algoritmos estão sendo utilizados para apoiar atividades em que interesses

interdependentes devem ser considerados simultaneamente [28].

Neste trabalho eles são utilizados para resolver o problema de integração e teste de módulos, considerando grupos de módulos que devido a restrição de desenvolvimento precisam ser testados em conjunto, tanto no contexto de OO, quanto no contexto de OA. Nos próximos capítulos são fornecidos os principais fundamentos da atividade de teste nestes contextos (Captítulo 3) e também sobre o problema sendo analisado e as estratégias existentes para a sua solução (Captítulo 4).

CAPÍTULO 3

TESTE DE SOFTWARE

O desenvolvimento de software é uma atividade complexa que demanda conhecimento de tecnologias, processos e domínio do contexto no qual o produto final será utilizado.

A construção de software é uma atividade desenvolvida por seres humanos, o que leva esta tarefa a estar totalmente passível de erros, uma vez que o resultado final está estritamente atrelado a habilidade, interpretação e execução das pessoas envolvidas no processo [22].

O teste de software pode ser considerado uma atividade de fundamental importância, principalmente para aumentar a confiabilidade de um produto e reduzir os problemas introduzidos durante o seu desenvolvimento.

Apesar das várias mudanças que ocorrem em relação a linguagens e paradigmas de programação, metodologias de desenvolvimento, surgimento de novos tipos de hardware, dentre outros acontecimentos, o objetivo do teste de software sempre continua o mesmo: encontrar defeitos [22].

O teste de software pode ser analisado segundo duas dimensões [49]: a primeira dimensão é chamada de Técnicas de Teste e relacionada a como enxergar o produto de software a fim de derivar requisitos de teste que possam cobrir o maior conjunto de possibilidades factíveis; a segunda dimensão refere-se ao tamanho e complexidade do elemento em teste, partindo-se das unidades mais simples até o sistema como um todo, essa dimensão é chamada de Fases do Teste.

Visando a uma contextualização geral da atividade de teste de software, a seguir são apresentadas as duas dimensões do teste: Técnicas de Teste e Fases do Teste.

3.1 Técnicas de Teste

Em um cenário ideal, um software deveria ser testado executando-se todos os elementos específicos do domínio de entrada, porém diante da grande possibilidade de combinações entre as possíveis entradas para um software, por mais simples que sejam, torna-se impossível a realização desta tarefa, pois o custo e o tempo seriam muito elevados [22].

Diante dessa impossibilidade, é importante selecionar um subconjunto representativo do domínio de entrada com as principais possibilidades de entrada e que seja bom o suficiente para encontrar a maior quantidade de erros.

Com objetivo de estabelecer passos bem definidos para selecionar o melhor subconjunto de requisitos de teste, que cumpram com sucesso sua função, foram estabelecidas diferentes técnicas de teste que fornecem diretrizes para projetar tais requisitos [49]. As técnicas de teste servem de guia para os testadores selecionarem os requisitos de teste de um software [45].

Dentro de cada técnica existem diferentes critérios que apoiam a derivação dos requisitos de teste. Critérios são regras utilizadas para efetivamente gerar os requisitos de teste, mas critérios não serão abordados neste trabalho uma vez que apresentam um conteúdo extenso e complexo e não fazem parte do objetivo principal desta proposta.

3.1.1 Técnica Funcional

A geração de requisitos de testes na técnica funcional, também chamada teste caixa-preta, não leva em consideração aspectos da implementação do software. O objetivo é determinar um conjunto de entradas que melhor represente o domínio do sistema, juntamente com as saídas relacionadas.

O foco desta técnica está em identificar defeitos de desempenho, interface, comportamento indesejado, falta de validações, defeito na inicialização e finalização de processos, dentre outros [45]. Utiliza-se, em geral, a especificação do software para derivar os requisitos de teste.

A vantagem desta técnica é que pode ser aplicada em todas as fases de teste, independente do paradigma de programação utilizado no desenvolvimento [22]. A desvantagem é ser muito dependente da experiência do testador, pois exige grande conhecimento do domínio de entrada. Outro agravante é que existem situações onde a especificação do software é descritiva e não formal [49].

Os principais critérios de teste desta técnica são: Particionamento em Classes de Equivalência, Análise do Valor Limite, Teste Funcional Sistemático, Grafo Causa-Efeito e *Error-Guessing* [22].

3.1.2 Técnica Estrutural

Em muitos casos, utilizar somente entradas e saídas não é suficiente para garantir que toda estrutura interna de um software foi testada exaustivamente, o que só é possível mediante a análise da implementação do sistema.

A técnica estrutural, ou teste de caixa branca, preocupa-se em gerar requisitos de teste observando-se a lógica do programa. Os requisitos de teste são estabelecidos com base na implementação dos artefatos em teste, verificando-se sua estrutura interna. É aconselhável utilizar a técnica estrutural como complementar da técnica funcional, porque ambas revelam diferentes tipos de defeitos [45].

Os critérios da técnica estrutural [38] incluem: critérios baseados em fluxo de controle, critérios baseados em fluxo de dados, e critérios baseados na complexidade.

3.1.3 Técnica Baseada em Defeitos

Esta técnica deriva requisitos de teste baseados em erros típicos, os mais comuns encontrados no domínio do software em desenvolvimento. O principal critério utilizado nesta técnica é chamado Análise de Mutantes [22].

No critério Análise de Mutantes são propositalmente introduzidos no programa defeitos originados por erros comumente cometidos por programadores, durante o desenvolvimento do software, gerando-se cópias do programa original [22], chamadas mutantes. O objetivo de introduzir tais defeitos é gerar dados de testes que quando submetidos ao programa

original e ao programa mutante apresentem comportamentos diferentes. Caso o dado de teste não apresente comportamentos diferentes, ele não é suficientemente completo para o teste, então novos dados devem ser gerados.

3.2 Fases do Teste

Segundo Pressman [45], o teste de software inicia-se a partir de componentes pequenos e segue em direção a elementos maiores, fazendo a integração das partes à medida que o escopo aumenta. Inicialmente isolam-se as menores unidades significantes a serem testadas (módulos), a fim de encontrar defeitos específicos daquele ponto. Na sequência integram-se as partes já testadas para encontrar defeitos de comunicação entre as unidades. Por fim, a atividade de teste é executada para o sistema como um todo, no ambiente operacional.

Cada um destes passos é chamado Fase do Teste e em geral as fases são divididas em: (i) Teste Unitário; (ii) Teste de Integração; e (iii) Teste de Sistema. Essa divisão tem como objetivo guiar o testador a criar melhores dados de testes, abordando diferentes características do software em cada fase, aumentando a possibilidade de encontrar defeitos [49].

As técnicas de teste apresentadas na seção anterior podem ser aplicadas em cada uma das fases do teste, porém com escopo diferenciado, mas o tamanho do artefato a ser testado influencia a utilização de determinada técnica.

3.2.1 Teste Unitário

O teste unitário, ou de unidade, leva em consideração identificar defeitos na menor unidade funcional do software, o módulo. Uma unidade, ou módulo, é uma parte do software que apresenta uma estrutura bem definida e que pode ser testada independentemente de outras partes, possibilitando que várias unidades possam ser testadas em paralelo.

O objetivo desta fase de teste é isolar a unidade e verificar defeitos relacionados a algoritmos incorretos ou mal implementados, estruturas de dados incorretas, ou simples erros de programação [22]. A técnica estrutural é a mais indicada nesta fase.

À medida que uma unidade é implementada ela já pode ser testada, pelo próprio desenvolvedor, independente da finalização completa do sistema ou de outras unidades.

Em alguns casos, devido ao isolamento da unidade, existe a necessidade de simular recursos externos, o que é feito por meio de *drivers* que representam a chamada ou invocação da unidade em teste e *stubs* que fornecem os recursos requeridos pela unidade, geralmente fornecidos por unidades chamadas ou invocadas.

Segundo Pressman [45], a necessidade de *drivers* e *stubs* durante a atividade de teste não é tarefa trivial e aumenta o custo do projeto.

3.2.2 Teste de Integração

A medida em que as unidades individuais que compõem o software vão sendo testadas, outra fase do teste pode ser iniciada, para verificar como se comportam as unidades quando trabalham em conjunto, uma vez que o funcionamento das várias unidades de um software, juntas entre si, é que estabelecem sua estrutura. No teste de integração o objetivo principal é encontrar defeitos relativos à comunicação entre as unidades do software.

Considerando-se os procedimentos como as principais unidades de um software, nesta fase do teste verifica-se o funcionamento correto do envio de parâmetros e os retornos entre os procedimentos; caso a unidade em teste sejam classes, testa-se o relacionamento entre os métodos e atributos compartilhados entre duas classes acopladas.

3.2.3 Teste de Sistema

A partir do momento que se tem o sistema finalizado, com suas partes integradas e testadas, inicia-se o teste de sistema.

O objetivo é encontrar defeitos relacionados ao funcionamento no ambiente operacional do software quando comparado às especificações iniciais. Requisitos não funcionais como desempenho, robustez, segurança e disponibilidade devem ser explorados [22].

3.3 Fases do Teste de Software Orientado a Objetos

Comparado ao paradigma procedimental, no contexto de OO o método pode ser assumido como a menor unidade a ser testada [60]. Porém um método não pode ser testado isolado da classe a qual pertence, pois um método só existe pelo objeto instanciado de uma classe. Diante desta situação a classe é vista como o *driver* do método [22].

Outros autores assumem que a classe é a menor unidade funcional [7, 40, 44], porém como por definição uma classe engloba métodos e atributos, então para garantir que a classe seja testada o melhor possível é necessário verificar o funcionamento conjunto entre as partes, o que é característico do teste de integração.

Com objetivo de englobar os diferentes componentes da classe como menor unidade funcional, Harrold e Rothermel [29] dividem o teste unitário de software OO em:

- **Intra-método**: cada método de uma classe é testado individualmente;
- **Inter-método**: testa a interação entre os métodos de uma classe que funcionam em conjunto para realizar determinada tarefa;
- **Intra-classe**: são testadas diferentes sequências de chamadas entre todos os métodos públicos de uma classe.

Para verificar erros na integração entre as classes de um software, os mesmos autores [29] propõem o teste **inter-classe** que, como no teste intra-classe, envolve a execução de várias sequências de chamadas de métodos públicos, porém, que não necessariamente precisam estar na mesma classe.

3.4 Fases do Teste de Software Orientado a Aspectos

Os defeitos comumente encontrados em software OO são herdados para o contexto de OA, o que permite que para a atividade de teste as fases já conhecidas e amplamente utilizadas em OO possam ser aplicadas neste contexto. Entretanto, diante das características específicas do contexto de OA, para tratar os novos tipos de defeitos [2, 13], novas fases devem ser introduzidas ou adaptadas.

Para se testar as diferentes abstrações, como por exemplo, aspectos, classes, métodos e adendos, Lemos [37] propôs uma classificação para as diferentes fases do teste de software OA baseada na proposta de Sommerville [52] e na abordagem de Harrold e Rothermel [29]. Esta classificação é formada por:

1. **Teste de Unidade**: considerando-se método e adendo como as menores unidades funcionais, o teste de unidade pode ser dividido em dois tipos:

   - **Intra-método**: o mesmo para o teste de métodos exposto no contexto OO [29], que tem como objetivo testar cada método isoladamente;
   - **Intra-adendo**: similar ao item anterior mas com objetivo de testar cada adendo individualmente;

2. **Teste de Módulo**: esta fase visa a testar o conjunto de unidades interdependentes que se comunicam por mensagens ou por interações com adendos. Considerando a diferença entre classe e aspectos, os tipos são:

   - **Inter-método**: visa a testar cada método público juntamente com outros métodos da mesma classe que são chamados direta ou indiretamente [29];
   - **Adendo-método**: consiste em testar cada adendo juntamente com outros métodos chamados por ele direta ou indiretamente;
   - **Método-adendo**: testa cada método público juntamente com os adendos que o afetam direta ou indiretamente;
   - **Inter-adendo**: visa a testar cada adendo juntamente com outros adendos que o afetam direta ou indiretamente;
   - **Inter-método-adendo**: consiste em testar cada método público juntamente com os adendos que o afetam direta e indiretamente, e com métodos chamados direta ou indiretamente. Esse tipo de teste engloba os quatro primeiros tipos de teste apresentados acima;

- **Intra-classe**: são testadas diferentes sequências de chamadas entre todos os métodos públicos de uma classe [29], considerando ou não a interação com aspectos.
- **Inter-classe**: assim como o anterior, o teste é executado através de diferentes sequências de chamadas entre todos os métodos públicos, porém considerando classes diferentes que interagem entre si [29], considerando ou não a interação com aspectos.

3.5 Considerações Finais

Um fator que contribui para o aumento do custo na atividade de teste é a necessidade de construção de *stubs*, que durante o teste de integração ocorre com grande frequência, pois muitas vezes um recurso requerido ainda está em desenvolvimento. Uma estratégia para reduzir essa necessidade de *stubs*, e consequentemente a complexidade e o custo do teste de software, consiste em estabelecer uma sequência para a integração e teste dos módulos, que podem ser classes ou aspectos. Esta sequência deve implicar em um custo mínimo, assunto abordado no próximo capítulo.

CAPÍTULO 4

DETERMINAÇÃO DE SEQUÊNCIAS DE INTEGRAÇÃO E TESTE DE MÓDULOS

A minimização do custo do teste de integração de módulos[1] por meio da redução do custo de construção de *stubs* é o foco principal deste trabalho, portanto neste capítulo é abordado o estado da arte em que se encontra a pesquisa nesta área.

A complexidade elevada e o grande esforço necessário para a integração e teste de módulos foi inicialmente identificada em sistemas OO, devido à características deste contexto. No contexto OA estas características também são encontradas, visto que a OA é uma evolução da OO. Em ambos contextos, trabalhos foram realizados a fim de solucionar o problema.

Diante do exposto, a seguir é apresentado o problema existente durante a integração e teste de módulos, seguido pela apresentação das principais estratégias utilizadas e trabalhos encontrados, em ambos contextos. Por fim, a abordagem MECBA, utilizada como base para este trabalho, é descrita.

4.1 O Problema de Ordenação de Módulos

Durante o teste de integração, alguns módulos precisam de recursos de outros módulos que ainda estão em desenvolvimento, o que leva a necessidade de construção de *stubs*, implicando em aumento de complexidade e custo do teste. *Stubs* são pseudo-implementações que simulam recursos que não estão disponíveis para executar o teste e que são indispensáveis para o funcionamento do módulo em teste.

Com o objetivo de reduzir o custo do teste de integração, através da minimização do custo de construção de *stubs*, procura-se estabelecer uma sequência de desenvolvimento e teste de módulos de um software. Na Engenharia de Software esta tarefa é chamada de

[1]No decorrer deste trabalho, módulo será utilizado para se referir a uma classe e/ou um aspecto.

Ordenação e Teste de Integração de Módulos, referenciada pela abreviação CITO (*Class and Integration Test Order*) [1] para o contexto OO; e pela abreviação CAITO (*Class and Aspect Integration an Test Order*) [24] para o contexto OA.

Diante do grande número de dependências entre as classes nos sistemas OO, encontrar uma sequência associada a um custo mínimo (número de *stubs*), não é tarefa trivial. Para sistemas OA, o problema apresenta maior dificuldade, uma vez que novas formas de dependências são introduzidas devido a utilização de aspectos.

Na literatura encontram-se vários trabalhos que propuseram diferentes estratégias para solucionar o problema CITO, muitos deles baseados em grafos direcionados, chamados *Object Relation Diagrams* (ORDs) [35, 36]. Estes grafos são abstrações da estrutura de classes de um software, onde os vértices representam as classes e as arestas correspondem aos relacionamentos entre elas. Um exemplo de ORD é apresentado na Figura 4.1 no qual é interessante observar que cada relacionamento é rotulado pelo seu tipo.

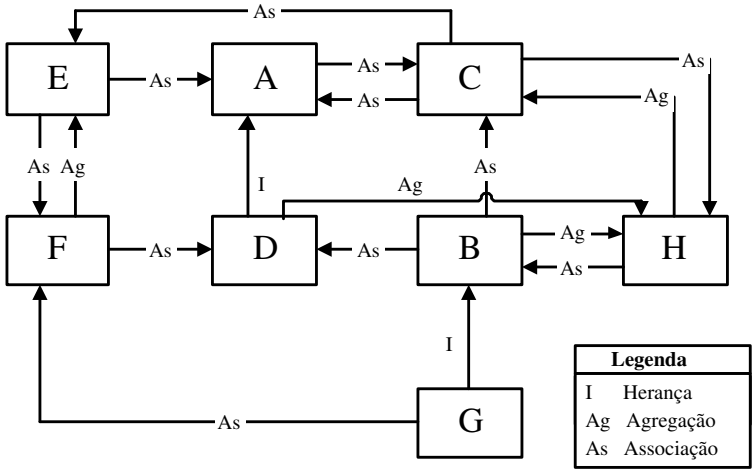


Figura 4.1: Exemplo de ORD (Extraída de [49])

Inicialmente as soluções propostas para o problema CAITO (contexto de OA) foram baseadas em adaptações de estratégias existentes no contexto OO. Ré et al. [50, 51] propuseram um ORD estendido que considera as dependências entre aspectos e classes. Um exemplo de ORD estendido é apresentado na Figura 4.2.

No ORD estendido podem ser observados os novos tipos de relacionamentos:

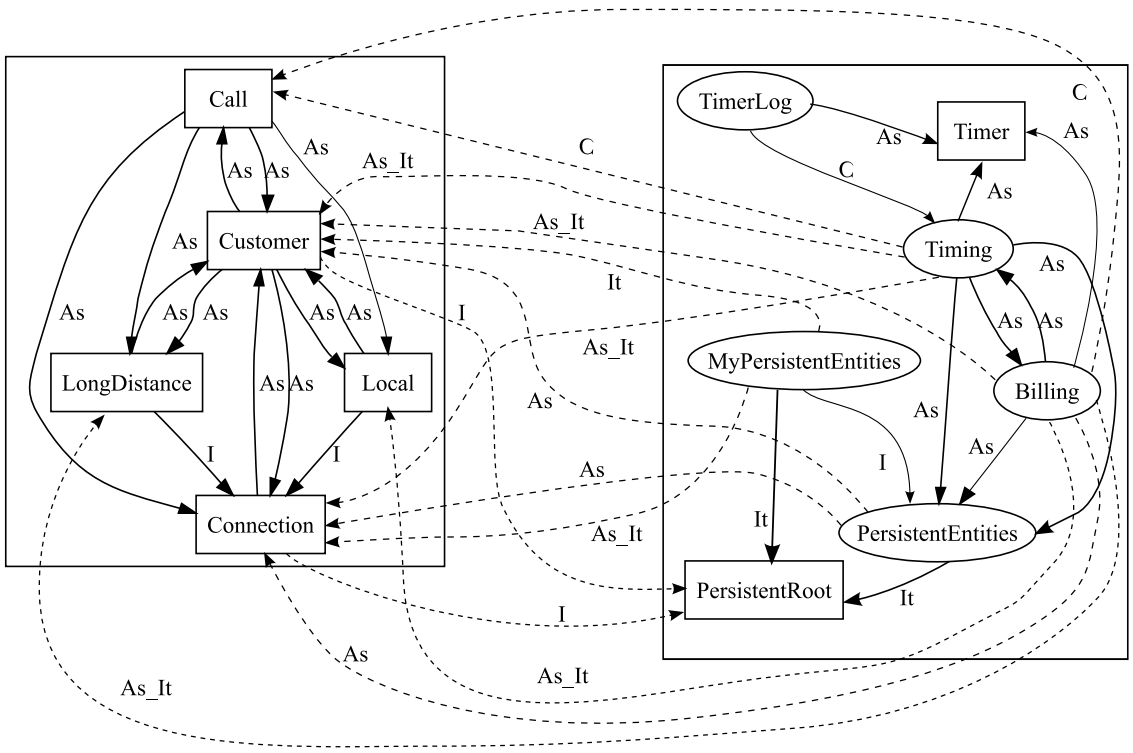


Figura 4.2: Exemplo de ORD Estendido (Extraída de [51])

- **Associação Transversal ou *Crosscutting Association* (C)**: representa a associação gerada pelo ponto de corte de um aspecto com um método de uma classe ou adendo de outro aspecto. Na Figura 4.2 este relacionamento é observado entre o aspecto `Billing` e a classe `Call`;

- **Dependência (U)**: é gerada pela relação entre adendos e pontos de cortes, e entre pontos de corte;

- **Associação de Dependência ou *Association Dependency* (As)**: ocorre entre objetos interceptados por um ponto de corte. Esse tipo de associação é apresentado na Figura 4.2 pelo relacionamento entre `Timing` e `Customer`;

- **Dependência de Declaração Intertipo ou *Intertype Declaration Dependency* (It)**: ocorre quando existe uma declaração intertipo em um aspecto para uma classe. Por exemplo, um aspecto `Aa` declara que a classe `A` estende `B`. No ORD estendido de exemplo, este tipo de relacionamento é apresentado entre `Billing` e `Local`, e entre `MyPersistentEntities`, `PersistentRoot` e `Connection`;

- **Dependência de Herança ou *Inheritance Dependency* (I)**: representa o relacionamento de herança entre aspectos, classes, ou entre classes e aspectos, como pode ser observado entre `PersistentEntities` e `MyPersistentEntities`, na Figura 4.2.

Tanto no ORD quando no ORD estendido, quando não existem ciclos de dependências, uma solução que não necessite de *stubs* pode ser encontrada mediante uma simples ordenação topológica inversa do grafo, considerando as dependências entre os vértices. Porém, Melton e Tempero mostram em um estudo [41] que a maior parte dos programas desenvolvidos em Java[2], e consequentemente desenvolvidos em AspectJ[3], apresentam ciclos de dependência, o que aumenta a complexidade para encontrar uma solução para o problema.

Esta complexidade existente devido aos ciclos de dependência motivou vários estudos que propuseram diferentes abordagens para quebrar os ciclos de dependência, a fim de obter uma ordenação de classes que reduz o custo de construção de *stubs*. Estas abordagens podem ser classificadas em duas categorias: (i) abordagens tradicionais; e (ii) abordagens baseadas em meta-heurísticas. Os trabalhos de ambas as categorias são descritos a seguir.

4.2 Estratégias Tradicionais

Kung et al. [35], além de proporem o ORD, foram os pioneiros em propor a identificação e quebra dos componentes fortemente conectados[4], chamados de SCCs (*Strongly Connected Components*). Em seu trabalho (Kung et al. [35]), os SCCs são identificados no ORD e as associações presentes nos SCCs são removidas até que não existam mais ciclos no ORD. Após a quebra dos ciclos, através de uma ordenação topológica é estabelecida a ordem de integração e teste dos módulos.

Tai e Daniels [53] propuseram uma abordagem de quebra de ciclos baseada na atribuição de dois níveis de dependência entre as classes: maior e menor. O nível maior

[2]Java é uma linguagens de programação amplamente utilizada no contexto OO.

[3]AspectJ é a principal linguagem de programação utilizada no contexto OA, sendo baseada na linguagem de programação Java.

[4]Sendo $p = (u,v)$ um par de vértices em um grafo G, G é chamado fortemente conectado se para todo p, existe um caminho de u para v, e de v para u.

é atribuído considerando-se as dependências de herança e agregação, já o nível menor é atribuído considerando-se as associações. Para escolher qual ciclo será quebrado, é atribuído um peso para cada aresta do SCC com base na relação de dependências de entrada e saída, e as arestas com maior pesos são removidas.

Le Traon et al. [55] aplicaram o algoritmo de Tarjan [54] para a identificação dos SCCs e, para cada SCC, utilizaram um conceito de dependência chamado de *Frond Dependency* para quebrar os ciclos. *Frond Dependency* refere-se a uma aresta que vai de um vértice X para um vértice Y, sendo que este vértice Y é antecessor direto ou indireto de X. Para decidir qual dependência vai ser quebrada, são atribuídos pesos através da soma das dependências de entrada e de saída para determinada classe do SCC. O processo é aplicado recursivamente para os SCCs que têm mais de um vértice. Quando dois vértices têm o mesmo peso, a decisão pela quebra é arbitrária.

Briand e Labiche [10] também usaram o algoritmo de Tarjan para identificar os ciclos e definir pesos para cada aresta, entretanto nesta estratégia o cálculo do peso é mais representativo. Para calcular os pesos foi considerado o número de ciclos em que cada aresta está envolvida, e a aresta com maior peso, consequentemente envolvida em mais ciclos, é eliminada. Este processo é repetido até não existirem mais SCCs. Uma vantagem desta abordagem é que não são quebradas dependências de herança e agregação, pois estas dependências são mais complexas de serem simuladas.

Esta mesma estratégia de Briand e Labiche [10] foi estendida para o contexto OA. No trabalho de Ré et al. [50] o ORD estendido (Figura 4.2) foi utilizado como base para aplicação do algoritmo de Tarjan e cálculo dos pesos. Assim como Briand e Labiche [10], as arestas com maior peso são selecionadas para serem removidas. Além de não quebrar herança e agregação, também não foram quebradas associações de declaração inter-tipo.

Este mesmo ORD estendido também foi utilizado para avaliar quatro diferentes estratégias de integração e teste de classes e aspectos [51]. Estas estratégias são: (i) *Combined*, que considera que classes e os aspectos devem ser integrados e testados em paralelo; (ii) *Incremental+*, integram-se primeiramente todas as classes e posteriormente considera-se os aspectos, como proposto por Ceccato et al. [13]; (iii) *Reverse*, que aplica a

sequência reversa da encontrada pela estratégia *Combined*; e (iv) *Random*, que determina uma sequência aleatória de integração. O resultado do estudo [51] apontou as estratégias *Combined* e *Incremental+* como as melhores, exigindo menor número de *stubs*. Porém os autores afirmam que esta estratégia *Combined* assemelha-se mais à realidade do ciclo de desenvolvimento de software, já que classes e aspectos são desenvolvidos conjuntamente.

Abdurazik and Offutt [1] adotaram uma maneira diferente de decidir pela quebra dos ciclos. Um peso foi atribuído tanto para a aresta quanto para o vértice, com base em uma análise quantitativa de nove medidas de acoplamento. As medidas de acoplamento envolvem o uso do número de parâmetros, o número de tipos de retorno, o número de variáveis e o número de métodos, necessárias na construção do *stub*. Para quebrar os ciclos, considera-se o custo em simular uma classe caso ela seja removida, portanto é mais viável construir um *stub* de classe envolvida em vários SCC, pois este *stub* pode ser utilizado várias vezes, reduzindo o custo e facilitando a integração e teste. A avaliação desta abordagem [1] demostrou melhores resultados que as trabalhos anteriores.

Kraft et al. [34] implementaram uma abordagem chamada COS (*Class and Order System*), que utiliza o ORD e seis tipos de dependência para rotular as arestas. As dependências são: herança, composição, associação, dependência, polimorfismo e *owenedElement*. Através do tipo da dependência e da tentativa e erro, são atribuídos os pesos para as arestas. Nesta abordagem, as arestas com menor peso são removidas para quebrar os ciclos.

Mao et al. [39] propuseram uma extensão do ORD e consideram três heurísticas para remover as arestas dos SCCs. A heurísticas são: peso cíclico, fatores de direção das arestas e intensidade da associação. Através da utilização de um algoritmo chamado AICTO, primeiramente todos os ciclos com dois vértices são removidos, depois outros tipos de ciclos são quebrados. Para decidir qual aresta do ciclo vai ser removida, são considerados os maiores pesos com base nas três heurísticas.

Jaroenpibooki et al. [31, 30] ao invés de utilizarem os tipos de relacionamento entre as classes, utilizaram técnicas de "fatiamento" de classes (*class slicing technique*). Neste trabalho os *stubs* são considerados como partes isoladas das classes. O procedimento para

estabelecer uma ordem compreende em: (i) identificar e ordenar os SCCs de acordo com seu tipo; e posteriormente (ii) fatiar as classes dos SCCs.

Todas as estratégias tradicionais utilizam grafos para representar as dependências, e com base em tais representações são aplicados algoritmos baseados em grafos para encontrar e quebrar os ciclos. Entretanto algumas desvantagens foram apontadas por Briand et al. [9] para estas estratégias. É geralmente difícil adaptá-las para considerarem fatores específicos relacionados a construção de *stubs*, tais como: o número de atributos necessários, o número de chamadas de métodos, etc. Além disso, a maioria das abordagens visam somente à minimização do número de ciclos que devem ser quebrados, entretanto, os autores [9] apontam que existem situações em que quebrar dois ciclos pode exigir um custo de construção de *stubs* menor do que quebrar somente um. Portanto, a maioria das estratégias geram soluções que são sub-ótimas.

4.3 Estratégias Baseadas em Meta-heurísticas

Como apresentado na seção anterior, Briand et al. [9] apontam que as estratégias tradicionais dificultam a consideração de fatores que influenciam a construção dos *stubs*. A partir disto, alguns autores propuseram estratégias que aplicam meta-heurísticas, que facilitam a utilização de diferentes formas para calcular o custo da construção de *stubs*.

As estratégias baseadas em meta-heurísticas, são divididas em dois tipos de algoritmos utilizados: (i) algoritmos com função de agregação que usam média ponderada dos objetivos (geralmente medidas de acoplamento) [6, 8, 24]; e (ii) algoritmos multiobjetivos [3, 4, 6, 5, 11, 17, 18] que utilizam conceitos de dominância de Pareto (ver Capítulo 2). A seguir estes dois tipos de estratégias são apresentadas.

4.3.1 Algoritmos com Função de Agregação

Briand et al. [9] foram os primeiros autores a aplicar meta-heurísticas para estabelecer ordens de integração e teste. Eles aplicaram um AG usando como função objetivo uma função de agregação formada por duas medidas de acoplamento: número de atributos e

número de métodos, necessários para a construção de *stubs*, com diferentes pesos para cada medida. Em outro trabalho [8] os mesmos autores realizaram um experimento com um conjunto de sistemas reais utilizando quatro diferentes funções objetivo, formadas por: (i) número de dependências; (ii) número de acoplamentos de métodos; (iii) número de acoplamentos de atributos; e (iv) agregação das medidas de acoplamento através de uma média geométrica. Dentre as funções objetivo utilizadas, a função com agregação ponderada de acoplamento de atributos e acoplamento de métodos apresentou as melhores soluções utilizando pesos de 0,5 para cada medida.

No trabalho de Galvan et al. [24] os autores também aplicaram um AG que utiliza agregação das medidas baseadas no número de atributos e número de métodos, entretanto tratando do problema no contexto OA. Neste estudo [24] o AG, utilizando os mesmos pesos que Briand et al. [8], obteve melhor resultado que as estratégias tradicionais [50]. Este trabalho é considerado o primeiro estudo de utilização de meta-heurísticas para solução do problema CAITO.

Apesar das estratégias que aplicam meta-heurísticas com função de agregação de medidas apresentarem melhores resultados que as estratégias tradicionais, determinar pesos que permitam encontrar soluções com custo mínimo, ao mesmo tempo que exploram eficientemente o espaço de busca, não é tarefa trivial, como já discutido no Capítulo 2. Também observa-se que o problema de integração e teste de módulos tem características de problemas multiobjetivos, pois é necessário otimizar diferentes interesses simultaneamente, ou seja, o problema é de fato multiobjetivo [11, 18]. Portanto, algoritmos multiobjetivos foram aplicados nos problemas CITO/CAITO, conforme descrito na seção a seguir.

4.3.2 Algoritmos Multiobjetivos

Com relação ao uso de algoritmos multiobjetivos, o trabalho realizado por Cabral et al. [11], foi um dos primeiros trabalhos a utilizar os conceitos de Pareto [43] para o problema de integração e teste de módulos. Os autores aplicaram ao problema o algoritmo *Pareto Ant Colony* (PACO) que é baseado em otimização por colônia de formigas. Foram

considerados como objetivos as medidas de acoplamento de atributos e acoplamento de métodos, portanto dois objetivos. Esta estratégia apresentou melhores resultados que a estratégia que utiliza agregação de funções utilizada por Briand et al [8].

No trabalho de Vergilio et al. [59] os autores trataram do problema CITO aplicando três tipos diferentes de algoritmos multiobjetivos: o NSGA-II que é um MOEA, já apresentado no Capítulo 2, o algoritmo PACO, que também foi utilizado por Cabral et al. [11], e o *Multi-objective Tabu Search* (MTabu) que é baseado em busca tabu. Foram utilizados os dois objetivos tradicionais, assim como por Cabral et al. [11]. Dentre os três algoritmos utilizados, o algoritmo evolutivo NSGA-II apresentou os melhores resultados.

A partir dos bons resultados do MOEA NSGA-II, Assunção et al. [5] aplicaram três MOEAs para solucionar o problema CITO, também considerando dois objetivos. Os algoritmos utilizados foram: NSGA-II, SPEA2 e PAES. Os resultados apontaram o PAES como o MOEA que apresenta melhor comportamento para o problema. Outro diferencial deste trabalho [5] foi a utilização de sistemas maiores e mais complexos que os utilizados nos estudos citados anteriormente [8, 9, 11], caracterizando uma aplicação mais prática.

Um estudo semelhante foi executado por Colanzi et al. [18], entretanto no contexto OA. Os MOEAs NSGA-II e SPEA2 foram aplicados ao problema CAITO considerando as duas medidas tradicionais, porém estendendo o cálculo das medidas para considerar o custo de construir *stubs* também para os aspectos. Os resultados apontaram o NSGA-II como melhor algoritmo. Em uma extensão deste trabalho, Assunção et al. [4] aplicaram o MOEA PAES para ser comparado com o NSGA-II e SPEA2. O PAES obteve os melhores resultados, similarmente ao observado no contexto OO [5].

Assunção et al. [6] compararam a estratégia multiobjetivo, utilizando os MOEAs NSGA-II e SPEA2, com a: (i) estratégia tradicional utilizando o algoritmo de Tarjan [10]; e a (ii) estratégia utilizando um AG com função de agregação [8]. As duas medidas tradicionais foram utilizadas como objetivos à serem minimizados. A estratégia multiobjetivo foi considerada a melhor dentre as três estratégias, pois obteve a maior quantidade de soluções na Fronteira de Pareto (ver Capítulo 2, Seção 2.3), além de não necessitar de ajuste de pesos.

Em outro trabalho Assunção et al. [3] consideraram outras características mais específicas de acoplamento entre classes. Eles introduziram novas medidas de complexidade de *stubs*, além das medidas tradicionais, (i) número de métodos e (ii) número de atributos, os autores utilizaram (iii) número de tipos distintos de retornos, e (iv) número de tipos distintos de parâmetros. No experimento foram utilizados os MOEAs NSGA-II e SPEA2. Para avaliar os resultados foram usados os indicadores de qualidade GD, IGD, C, e ED (descritos no Capítulo 2, Seção 2.3). Os resultados apontaram que o SPEA2 obteve um convergência levemente melhor que o NSGA-II, mas ambos encontraram boas ordens de integração e teste.

Como tentativa de generalizar a resolução do problema de integração e teste de módulos, no trabalho de Colanzi et al. [17] foi proposta uma abordagem que define um conjunto de quatro etapas genéricas para estabelecer ordens de integração e teste de módulos. Esta abordagem constitui a principal referência para o presente trabalho, e portanto na seção a seguir ela será descrita em detalhes.

4.3.3 A Abordagem MECBA

A abordagem MECBA (*Multi-Evolutionary and Coupling-Based Approach*) proposta no trabalho de Colanzi et al. [17] tem como objetivo apresentar etapas genéricas para tratar do problema de integração e teste de módulos. A Figura 4.3 ilustra a abordagem.

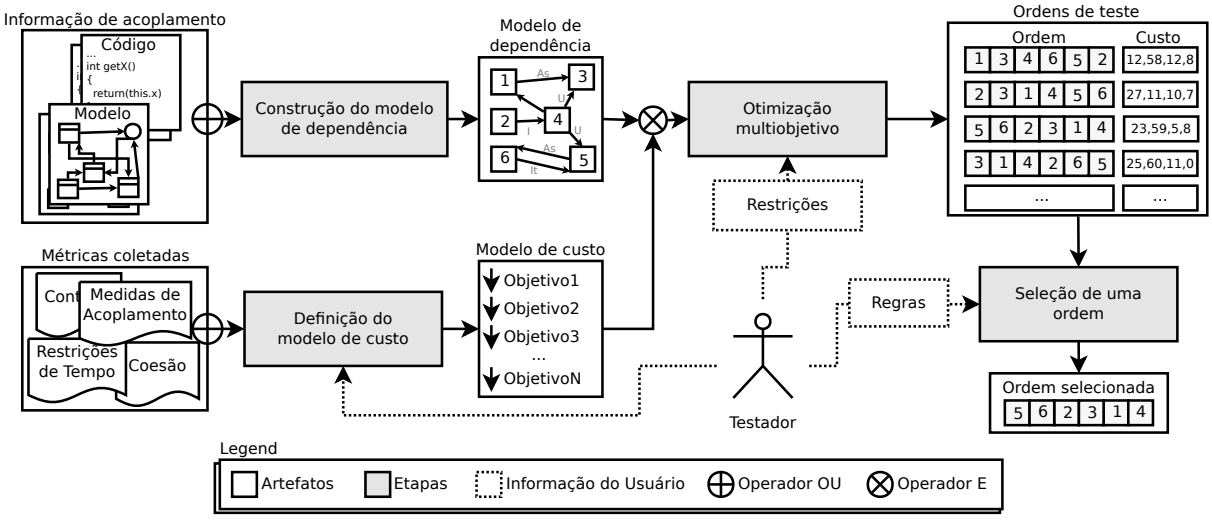


Figura 4.3: Etapas Detalhadas da Abordagem MECBA

As etapas da abordagem são descritas a seguir:

1. **Construção do modelo de dependência**: produz uma representação das dependências a serem consideradas. Um exemplo de modelo de dependência a ser utilizado no contexto OO é o ORD [35, 36]. No contexto OA, pode ser utilizado o ORD estendido [50].

2. **Definição do modelo de custo**: define quais medidas são utilizadas pelos algoritmos de otimização para calcular o custo envolvido na construção de *stubs*, ou seja, as medidas utilizadas pela função de avaliação. Sugere-se o uso de medidas de acoplamento tais como [3]: (i) número de atributos; (ii) número de métodos; (iii) número distintos de retornos; e (iv) número distintos de parâmetros.

3. **Otimização multiobjetivo**: nesta etapa qualquer algoritmo de otimização multiobjetivo pode ser utilizado, mas devido aos resultados apresenttados na literatura, sugere-se a utilização dos MOEAs. Esses algoritmos recebem como entrada os artefatos produzidos nas etapas 1 e 2 e executam uma busca através de um processo evolutivo, respeitando restrições impostas pelo testador. Como resultado produzem um conjunto de boas soluções para o problema, com uma grande diversidade de custos associados às diferentes medidas adotadas no modelo de custo.

4. **Seleção de uma Ordem**: como a etapa anterior produz um conjunto de diferentes soluções, nesta etapa o testador analisa a medida de custo que deseja priorizar ou a restrição relativa a atividade de teste para decidir qual das soluções utilizar.

Para avaliar a abordagem MECBA, Colanzi et al. instanciaram a abordagem para o contexto OA [17]. As características do experimento e avaliação são apresentadas na seção a seguir. Algumas destas características também serão utilizadas no experimento e avaliação da abordagem propostas no presente trabalho.

4.3.3.1 Aspectos de Implementação e Avaliação

Para avaliar a abordagem MECBA, Colanzi et al. conduziram um experimento para tratar do problema CAITO [17]. Foram utilizadas dois MOEAs, as mesmas quatro medidas de acoplamento utilizadas por Assunção et al. [3] e quatro sistemas reais, desenvolvidos em OA. Os aspectos de implementação utilizados no experimento são apresentados a seguir:

- **MOEAs**: foram utilizados os MOEAs NSGA-II e SPEA2, já apresentados no Capítulo 2. Como base para a implementação foi utilizado o *framework* JMetal [23], escrito na linguagem Java;

- **Representação do Problema**: por se tratar de um problema de permutação de módulos que formam uma ordem em que estes devem ser desenvolvidos e testados, o problema foi representado por um vetor de números inteiros onde cada posição do vetor corresponde a um número identificador de cada módulo. O tamanho do vetor é igual ao número de módulos que compõem o sistema em teste;

- **Entradas dos Algoritmos**: os dados de entrada do algoritmo são formados por cinco matrizes lidas de um arquivo de texto. Uma matriz corresponde às dependências entre os módulos e as outras quatro são relativas as medidas utilizadas para o cálculo de *fitness*. As matrizes são: (i) matriz de dependência entre os módulos; (ii) matriz de acoplamento de atributos; (iii) matriz de acoplamento de operações; (iv) matriz de tipos distintos de retornos; e (v) matriz de tipos distintos de parâmetro;

- **Restrições**: as restrições consideradas durante o processamento do algoritmo foram: não quebrar as dependências de herança e declaração intertipos. Durante a execução dos MOEAs estas restrições foram verificadas no momento de geração da população inicial e durante a aplicação da mutação e do cruzamento. A estratégia de tratamento das restrições envolve uma varredura do início ao fim do cromossomo, verificando se já apareceram as dependências para os módulos que são dependentes de outros módulos. Caso uma restrição de precedência de módulos seja quebrada,

este módulo é colocado no fim do cromossomo e todos os módulos posteriores têm a sua posição decrementada em uma posição;

- **Função de Avaliação**: para calcular o *fitness* de cada solução, foram utilizadas medidas de acoplamento que são entradas dos algoritmos. Cada medida é calculada como um objetivo e o seu valor foi obtido pela somatória dos valores de acoplamento entre os módulos que têm a dependência quebrada.

As quatro medidas de acoplamento utilizadas no experimento da abordagem MECBA têm como base as medidas utilizadas por Assunção et al. [3]. Para definição formal das quatro medidas, considera-se que: (i) m_i and m_j são dois módulos acoplados, (ii) módulo pode ser tanto uma classe quanto um aspecto, (iii) o termo "operação" representa um método de uma classe ou um método/adendo de um aspecto, e (iv) m_i depende m_j. Assim, tem-se que:

1. **Número de atributos (A)**: corresponde ao número de atributos declarados localmente em m_j quando referências ou ponteiros de instâncias de m_j aparecem na lista de argumentos de alguma operação de m_i, como um tipo de retorno, na lista de atributos de m_i, ou como parâmetros locais de alguma operação de m_i. Esta medida conta o número de atributos que devem ser emulados no *stub* se a dependência for quebrada. No caso de herança, não se conta o número de atributos herdados, assim como feito por Briand et al [9].

2. **Número de operações (O)**: corresponde ao número de operações, incluídos construtores, declarados localmente em m_j que são invocados por operações de m_i. Esta medida conta o número de operações que devem ser emuladas no *stub* se a dependência for quebrada. No caso de herança conta-se o número de operações declaradas no módulo pai.

3. **Número de tipos distintos de retornos (R)**: corresponde ao número de tipos distintos de retornos declarados localmente em m_j que são invocados por operações de m_i. Retorno do tipo `void` não são contados. Em caso de herança, conta-se o número de tipos distintos de retornos das operações declaradas no módulo pai.

4. **Número de tipos distintos de parâmetros (P)**: corresponde ao número de parâmetros declarados localmente em m_j e invocados por operações de m_i. Quando existe sobrecarga de operações, o número de parâmetros é igual a soma de todos os parâmetros distintos entre todas as implementações de cada método. Sempre considera-se que existe possibilidade de invocação de todas as implementações de determinada operação. Em caso de herança conta-se o número de tipos distintos de parâmetros das operações declaradas no módulo pai.

Considerando uma ordem de teste t, o custo para as medidas A, O, R e P é calculado pelas Formulas 4.1, 4.2, 4.3 e 4.4, respectivamente. AM é a matriz que representa as dependências de atributos entre os módulos, OM é a matriz que representa as dependências de operações entre os módulos, RM é a matriz que representa as dependências de tipos distintos de retornos entre operações de dois módulos e PM é a matriz que representa as dependências de tipos distintos de parâmetros de operações de dois módulos. Em cada matriz, o elemento da linha i e da coluna j (i, j) vai indicar o número total de determinada dependência entre o módulo i que depende de j. Entretanto, após calculadas as medidas para cada módulo, este deve ser considerado como já incluído na ordem de integração e teste (representado nas fórmulas por k), então a soma das dependências só deve ser efetuada se o módulo precedente não existir previamente, ou seja $j \neq k$.

$$A(t) = \sum_{i=1}^{n} \sum_{j=1}^{n} AM(i,j); j \neq k \tag{4.1}$$

$$O(t) = \sum_{i=1}^{n} \sum_{j=1}^{n} OM(i,j); j \neq k \tag{4.2}$$

$$R(t) = \sum_{i=1}^{n} \sum_{j=1}^{n} RM(i,j); j \neq k \tag{4.3}$$

$$P(t) = \sum_{i=1}^{n} \sum_{j=1}^{n} PM(i,j); j \neq k \tag{4.4}$$

Durante o experimento quatro sistemas OA foram utilizados, esses sistemas são apresentados na Tabela 4.1. Para obter as medidas de acoplamento, foi utilizado um *parser* para obter as informações das dependências entre os módulos a partir do código fonte, usando de engenharia reversa. O *parser* teve como entrada os arquivos do código fonte, desenvolvidos com a linguagem AspectJ, e retornou como saída um arquivo de texto com as matrizes utilizadas como entrada para os algoritmos.

Tabela 4.1: Sistemas Utilizados para Avaliação da MECBA

Software	**Versão**	**Classes**	**Aspectos**	**Referência**
AJHotDraw	0.4	290	31	http://sourceforge.net/projects/ajhotdraw/
AJHSQLDB	18	276	25	http://sourceforge.net/projects/ajhsqldb/
HealthWatcher	9	95	22	http://www.comp.lancs.ac.uk/∼greenwop/tao/
Toll-System	1	53	24	http://www.aosd-europe.net/

Para executar os MOEAs, após uma etapa de ajuste de parâmetros dos algoritmos, os autores adotaram a configuração apresentada na Tabela 4.2:

Tabela 4.2: Valores dos Parâmetros Utilizados para Avaliação da MECBA

Parâmetro	**NSGA-II**	**SPEA2**
Tamanho da População	300	300
Taxa de Cruzamento	0,95	0,95
Taxa de Mutação	0,02	0,02
Tamanho do Arquivo Externo	-	250
Número de Avaliações de *Fitness*	20000	20000

Com esta configuração, cada um dos dois MOEAs foi executado 30 vezes. Por apresentarem operações aleatórias em seu processo evolutivo, somente uma execução não seria suficiente para determinar o seu comportamento padrão.

Para avaliar os resultados e comparar o comportamento dos algoritmos, Colanzi et al. utilizaram os indicadores de qualidade GD, IGD, C, e ED já apresentados no Capítulo 2.

Com os resultados Colanzi et al. afirmam que os algoritmos evolutivos multiobjetivos são eficazes para solucionar o problema CAITO. Os resultados do estudo experimental apontaram ainda que para os sistemas HealthWatcher e Toll-System os dois algoritmos tiveram um comportamento similar, encontrando um única solução, já para os sistemas AJHotDraw e AJHSQLDB o algoritmo SPEA2 parece ser mais apropriado por encontrar soluções com melhor balanceamento entre os objetivos.

Ainda como resultados do trabalho Colanzi et al. apresentam exemplos de uso das soluções encontradas. Foram selecionadas para cada sistema duas soluções utilizando o indicador ED, com exceção dos sistemas que tiveram apenas uma solução, e apresentadas formas de priorizar determinada medida com base nos valores encontrados para cada objetivo correspondente as diferentes medidas de acoplamento.

4.4 Considerações Finais

Este capítulo apresentou o problema de integração e teste de módulos no contexto de OO e OA. Foram descritos os trabalhos que utilizam estratégias tradicionais e estratégias baseadas em meta-heurísticas, sendo estas divididas em baseadas em algoritmos que usam agregação de funções e baseadas em algoritmos multiobjetivos. Esses trabalhos visam a diminuir a complexidade e custo do teste e consequentemente do produto final.

Observa-se que o problema é de fato multiobjetivo, pois medidas diferentes e interdependentes precisam ser otimizadas simultaneamente [11, 18]. Por este motivo as estratégias baseadas em algoritmos multiobjetivos apresentam melhores resultados.

Em uma comparação entre três diferentes algoritmos multiobjetivos aplicados ao problema CITO, o MOEA apresentou melhor comportamento que os outros algoritmos [59]. Com base neste resultado, vários outros trabalhos aplicaram MOEAs tanto no contexto OO quanto no contexto OA. Dentre este trabalhos, a abordagem MECBA [17] apresenta os melhores resultados, portanto constitui a base para o presente trabalho.

Contudo, observa-se que dentre os trabalhos citados, nenhum deles considera uma característica presente na maioria dos projetos de sistemas a serem desenvolvidos: o agrupamento de módulos, também chamada de modularização [12]. Portanto, o objetivo deste trabalho é propor uma estratégia de integração e teste de módulos que considere os agrupamentos durante o estabelecimento de ordens para integração e teste de módulos.

No próximo capítulo apresenta-se a abordagem MECBA-Clu, baseada no trabalho de Colanzi et al. [17], elaborada para aplicar uma estratégia de integração e teste de módulos que considere a modularização, a fim de contribuir para o aumento da qualidade e redução do custo de teste durante o desenvolvimento de sistemas.

CAPÍTULO 5

MECBA-CLU

Os trabalhos apresentados no capitulo anterior tratam dos problemas CITO e CAITO com o objetivo de minimizar a complexidade e o custo para construção dos *stubs*, entretanto nenhum abordou restrições existentes no cenário de desenvolvimento de sistemas. Diante disso, este trabalho aborda agrupamento de módulos. Os agrupamentos de módulos são amplamente utilizados para facilitar a compreensão, o desenvolvimento e a manutenção de software [12, 62]. Em alguns casos a modularização também é aplicada devido a questões contratuais e/ou ao desenvolvimento distribuído [46].

Essa limitação constitui a motivação para o presente trabalho. Portanto, neste capítulo é introduzida uma estratégia de integração e teste de módulos baseada em agrupamentos e a abordagem que implementa esta estragégia, chamada MECBA-Clu (*Multi-Evolutionary and Coupling-Based Approach with Clusters*).

5.1 Integração e Teste de Módulos Baseada em Agrupamentos

A estratégia de ordenação de módulos proposta por este trabalho consiste em considerar um conjunto de agrupamentos de módulos durante o estabelecimento de uma ordem de integração e teste. Nesse contexto, um agrupamento corresponde a um conjunto de módulos que são agrupados pelo engenheiro de software e devem ser desenvolvidos e testados em conjunto. Tais módulos agrupados, devem estar unidos em uma ordem estabelecida, permitindo que sejam desenvolvidos e testados em sequência.

A Figura 5.1 apresenta um exemplo da utilização da estratégia baseada em agrupamentos de módulos. Considerando um sistema com doze módulos identificados de 1 a 12, o engenheiro de software pode determinar um conjunto de agrupamentos de módulos, através da análise das características que justificam que tais módulos devem estar agrupados. No exemplo, Figura 5.1, foram definidos três agrupamentos, identificados por A1,

A2 e A3, com quatro módulos, três módulos e cinco módulos, respectivamente. Portanto, seguindo a estratégia proposta, os módulos de cada agrupamento devem estar unidos e em sequência nas ordens de integração e teste estabelecidas.

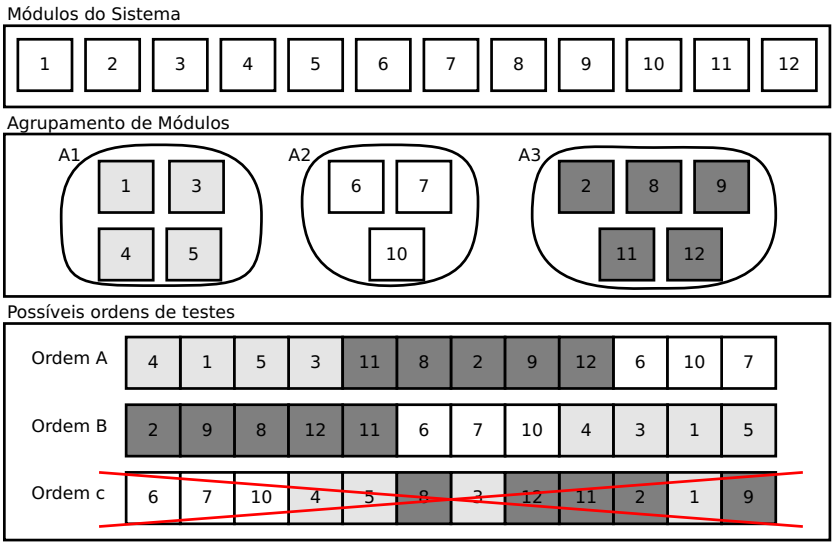


Figura 5.1: Exemplos da Utilização da Estratégia Baseada em Agrupamentos

Como exemplo foram geradas aleatoriamente três ordens para integração e teste, identificadas por Ordem A, Ordem B e Ordem C (Figura 5.1). As Ordens A e B são ordens válidas, pois os módulos que compõem os agrupamentos estão em sequência. Na Ordem A os módulos do agrupamento A1 devem ser desenvolvidos e testados primeiramente, seguidos pelos módulos do agrupamento A3 e finalmente os módulos do agrupamento A2. Na Ordem B os módulos do agrupamento A3 devem ser desenvolvidos e testados primeiramente, seguidos pelo módulos do agrupamento A2 e finalmente pelos módulos do agrupamento A1. Já a Ordem 3 não é uma sequência válida segundo a estratégia proposta, pois os módulos dos agrupamentos A1 e A3 ficaram dispersos, ou seja, os módulos dos agrupamentos A1 e A3 não estão agrupados como definidos pelo engenheiro de software.

Caso o engenheiro de software tenha definido os agrupamentos baseados em partes específicas para serem desenvolvidas por equipes distribuídas, as Ordens A e B do exemplo permitiriam que as equipes trabalhassem sem muita dependência entre si. Já para a Ordem 3 seria necessário que a equipe responsável pelo agrupamento A1 aguardasse que a equipe responsável pelo agrupamento A3 desenvolvesse alguns módulos para que essa finalizasse e testasse seu módulos.

5.2 A Abordagem MECBA-Clu

Como apresentado no Capítulo 4, dentre as abordagens utilizadas para solucionar os problemas CITO e CAITO, a abordagem MECBA mostra-se como a mais promissora. Portanto a estratégia de ordenação de módulos proposta no presente trabalho foi implementada através de uma extensão da abordagem MECBA.

A abordagem para integração e teste de módulos baseada em agrupamento e algoritmos de otimização multiobjetivos, chamada de MECBA-Clu (*Multi-Evolutionary and Coupling-Based Approach with Clusters*), é apresentada em detalhes na Figura 5.2.

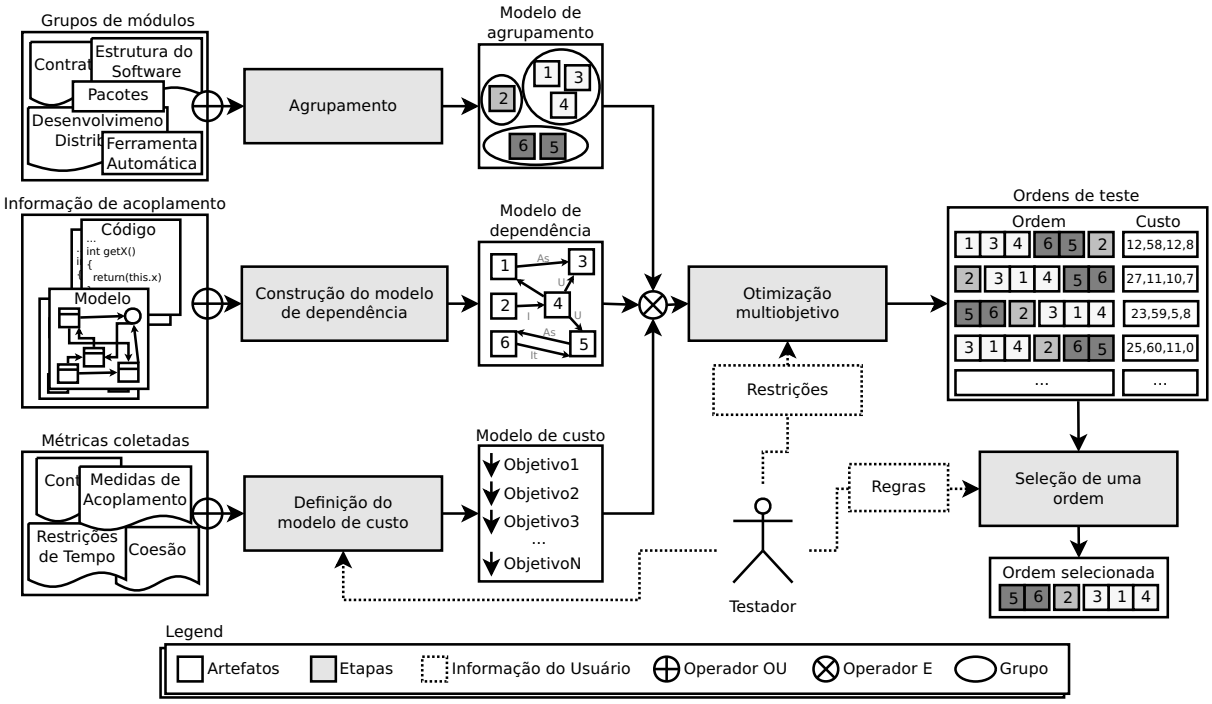


Figura 5.2: Etapas Detalhadas da Abordagem MECBA-Clu

A abordagem MECBA-Clu é formada por cinco etapas: (i) Definição do modelo de custo; (ii) Construção do modelo de dependência; (iii) Agrupamento; (iv) Otimização multiobjetivo; e (v) Seleção de uma ordem. Destas cinco etapas, quatro foram herdadas da abordagem MECBA (i, ii, iv e v), apresentadas na Seção 4.3.3 do Capítulo 4, e uma nova (iii) introduzida para lidar com os agrupamentos de módulos. A seguir a nova etapa é descrita.

- **Agrupamento**: esta etapa foi adicionada na abordagem MECBA-Clu com objetivo de permitir a determinação das regras de agrupamentos para os módulos que devem ser preservados em sequência dentro das ordens de teste a serem geradas. Esta etapa gera um modelo de agrupamento que é considerado durante a otimização. Por exemplo, pode-se desejar que as classes sejam testadas primeiramente e depois os aspectos. Neste caso tem-se dois grupos. Pode-se também optar que um grupo corresponda a classes de um mesmo pacote Java, ou ainda utilizar restrições relativas a um desenvolvimento distribuído.

A abordagem MECBA-Clu foi formulada com o objetivo de ser genérica e permitir a utilização de diferentes tipos de artefatos, algoritmos e regras em cada uma das etapas. Entretanto, algumas recomendações baseadas em trabalhos da literatura [3, 17, 59] são sugeridas para cada uma das etapas da abordagem, conforme segue:

- **Agrupamento**: recomenda-se utilizar agrupamentos de módulos definidos pelo engenheiro de software, que pode agrupar os módulos através da observação de características internas do sistema, ou através de restrições inerentes ao desenvolvimento. A critério de avaliações experimentais, o usuário pode estabelecer os agrupamentos segundo sua experiência ou de maneira aleatória;

- **Modelo de dependência**: recomenda-se utilizar os mesmos modelos utilizados nos vários experimentos que tratam dos problemas CITO e CAITO, que são o ORD para o contexto de OO e ORD estendido para o contexto de OA;

- **Modelo de custo**: medidas de acoplamento são amplamente utilizadas e representam com sucesso o custo de criação de *stubs*. As medidas de acoplamento portanto são recomendas para uso e avaliações da abordagem MECBA-Clu;

- **Otimização multiobjetivo**: dentre as meta-heurísticas aplicadas ao problema de integração e teste de módulos, os MOEAs apresentam melhor comportamento [59]. Recomenda-se a utilização dos algoritmos NSGA-II por ser o mais tradicional e utilizado em experimentos; o SPEA2 como opção de comparação para o NSGA-II,

pois ambos apresentam um processo evolutivo similar; e o PAES por apresentar um processo evolutivo diferenciado do tradicional;

- **Seleção de uma ordem**: a seleção de uma ordem, dentre todas as encontradas pelos algoritmos, deve ser baseada em qual medida o engenheiro de software/testador deseja priorizar. Caso for do interesse do usuário uma solução com bom balanceamento entre as medidas utilizadas, a decisão pode ser apoiada pelo indicador de qualidade ED.

Estas recomendações têm por objetivo oferecer uma direção em relação ao uso da abordagem MECBA-Clu, independente do cenário e das tecnologias disponíveis. Entretanto para avaliar a abordagem, foi sugerida uma forma de implementação, que é utilizada para a condução do experimento (descrito no Capítulo 6), cujos aspectos são apresentados a seguir.

5.3 Aspectos de Implementação

Os aspectos de implementação apresentados a seguir incluem detalhes quanto a possibilidade de aplicar a abordagem MECBA-Clu utilizando MOEAs (descritos no Capítulo 2), mas como exposto anteriormente, as etapas da MECBA-Clu são genéricas e possibilitam a extensão para outros cenários.

5.3.1 Representação do Problema

Para aplicar os MOEAs para o problema CITO/CAITO o primeiro passo é encontrar uma representação para o problema. Uma vez que o problema é relacionado a permutação de módulos, vários trabalhos [3, 11, 17, 18, 24] utilizaram uma representação formada por um vetor de números inteiros, no qual cada número corresponde a um identificador de um módulo. Entretanto, para considerar os agrupamentos durante o processo seletivo dos MOEAs, uma representação mais elaborada se faz necessária. Para tal foi implementada uma classe chamada `Cluster`, conforme apresentada na Figura 5.3.

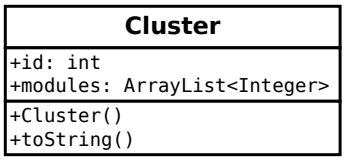


Figura 5.3: Classe Cluster

Um objeto da classe `Cluster` é composto por um identificador do agrupamento (`id`) do tipo inteiro e um vetor de inteiros para armazenar os módulos do agrupamento (`modules`). Uma solução gerada por um MOEA é composta por um vetor objetos da classe `Cluster` (`ArrayList<Cluster>`). O tamanho do vetor de objetos da classe `Cluster` é igual a quantidade de agrupamentos, e a soma de todos os tamanhos dos vetores de módulos (`modules`) é igual a quantidade de módulos de todo o sistema.

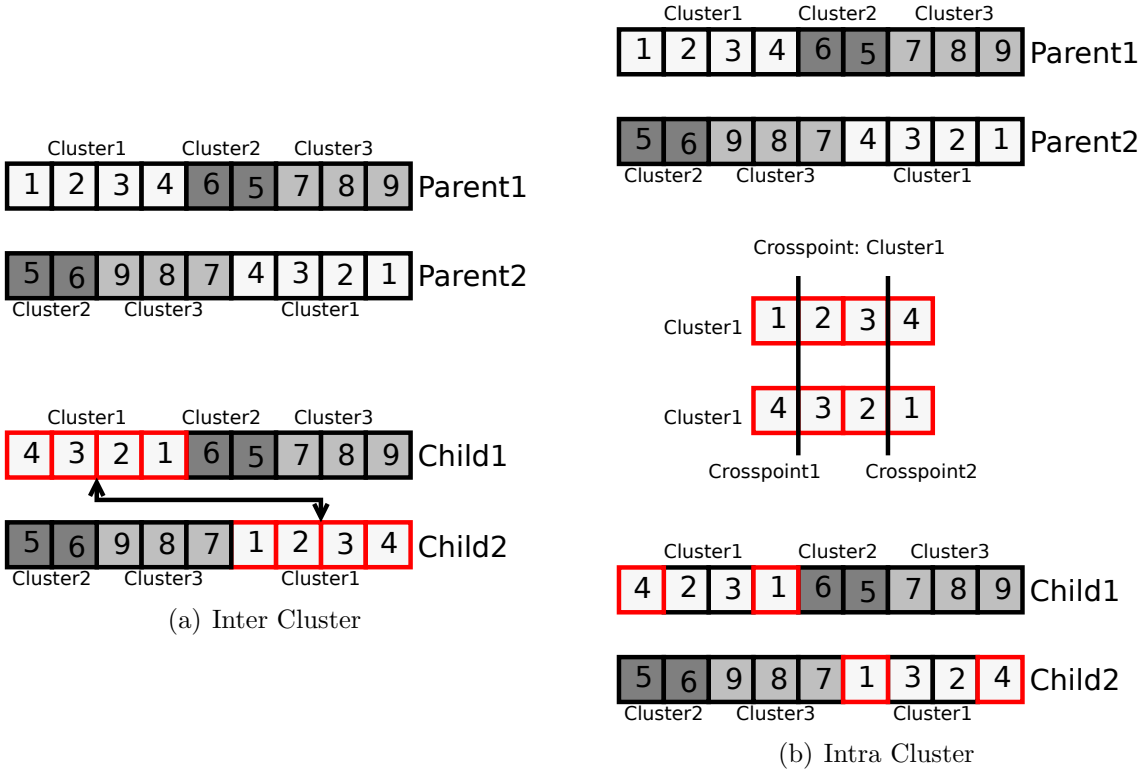
5.3.2 Operadores Genéticos

Devido à aplicação de MOEAs, é necessário utilizar operadores genéticos a fim de gerar novos indivíduos durante o processo evolutivo. Para a abordagem MECBA os operadores são aplicados de forma tradicional para problema de permutação, como por exemplo o problema do Caixeiro Viajante [23]. Entretanto, para a abordagem MECBA-Clu, uma nova forma de gerar e tratar com os indivíduos é necessária, devido as restrições dos agrupamentos que devem ser respeitadas. Dentre os operadores de seleção, cruzamento e mutação utilizados pela MECBA, apenas os dois últimos operadores necessitaram ser adaptados para a abordagem MECBA-Clu. Nas seções a seguir, os operadores de cruzamento e mutação utilizados na abordagem MECBA-Clu são apresentados.

5.3.2.1 Cruzamento

O operador de cruzamento utilizado na abordagem MECBA segue a estratégia de dois pontos de corte. Nesta estratégia, são selecionados dois pontos aleatoriamente, que são utilizados para determinar quais genes dos pais devem ser utilizados para gerar os filhos. Os genes que estão no intervalo dos dois pontos de corte selecionados, são trocados entre os indivíduos e os genes restantes são utilizados para preencher a sequência de genes, de forma que não existam genes repetidos.

Entretanto na abordagem MECBA-Clu, a simples seleção aleatória dos pontos de corte poderia quebrar a restrição de manter os módulos dos agrupamentos em sequência. Portanto, mediante a necessidade de respeitar os agrupamentos durante o cruzamento e devido a forma de representar o problema, foram implementadas duas possibilidades de gerar indivíduos filhos através da combinação entre os módulos. Estas duas possibilidades são apresentadas na Figura 5.4.



(a) Inter Cluster

(b) Intra Cluster

Figura 5.4: Operador de Cruzamento

A primeira forma de aplicar o cruzamento na abordagem MECBA-Clu é chamada de *Inter Cluster* (Figura 5.4(a)). Seu objetivo é gerar filhos que recebam um agrupamento trocado entre os seus pais. Como ilustrado no exemplo da Figura 5.4(a), após a seleção aleatória do agrupamento a ser trocado, o filho *Child 1* recebe o agrupamento *Cluster2* e *Cluster3* do pai *Parent1* e o *Cluster1* do pai *Parent2*. Já o filho *Child 2* recebe o agrupamento *Cluster2* e *Cluster3* do pai *Parent2* e o *Cluster1* do pai *Parent1*.

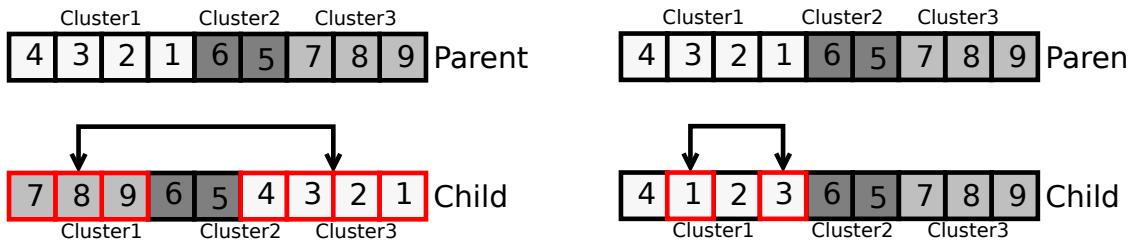
A segunda forma de aplicar o cruzamento é chamada *Intra Cluster* (Figura 5.4(b)). Seu objetivo é gerar filhos que recebam um agrupamento resultante do cruzamento dos agrupamentos dos pais. É aplicado o cruzamento com dois pontos de corte, mas neste

caso em um agrupamento específico. Como ilustrado na Figura 5.4(b), após a seleção aleatória do agrupamento que vai passar pelo cruzamento, dois pontos de corte também são determinados de forma aleatória. Com base nos pontos de corte, os módulos dos pais, que estão entre o intervalo dos pontos de corte, são trocados para gerar os filhos, e os outros módulos utilizados para completar o agrupamento. Os outros agrupamentos que não participam do cruzamento são copiados dos pais para os filhos.

Durante a composição de uma nova geração de indivíduos, são selecionados sempre dois indivíduos pais para aplicação do operador de cruzamento. A partir dos dois pais são gerados quatro filhos, dois a partir do cruzamento *Inter Cluster* e dois a partir do cruzamento *Intra Cluster*.

5.3.2.2 Mutação

Na abordagem MECBA, o operador de mutação utilizado segue a estratégia *swap mutation*. Nesta estratégia, dois genes são selecionados para serem trocados de posição. Entretanto, na abordagem MECBA-Clu, a mutação deve respeitar a composição dos agrupamentos. Diante disto, devido a forma de representar o problema, duas possibilidades de mutação foram implementadas. As duas possibilidades são apresentadas na Figura 5.5.



(a) Inter Cluster (b) Intra Cluster

Figura 5.5: Operador de Mutação

A primeira forma de mutação é chamada de *Inter Cluster* (Figura 5.5(a)). Seu objetivo é executar a mutação entre os agrupamentos, ou seja, selecionar aleatoriamente dois agrupamentos e trocá-los de posição. No exemplo da Figura 5.5(a) o *Cluster1* é trocado de posição com o *Cluster3*.

A segunda forma de mutação é chamada *Intra Cluster* (Figura 5.5(b)). Seu objetivo é executar a mutação na posição dos módulos dentro de um agrupamento, ou seja, selecionar aleatoriamente dois módulos, dentro de um agrupamento também selecionado aleatoriamente, e trocá-los de posição. No exemplo da Figura 5.5(b) o agrupamento *Cluster1* foi selecionado para receber a mutação, e os módulos identificados por 1 e 3 foram trocados de posição.

Durante o processo evolutivo o operador de mutação é aplicado em cada um dos indivíduos gerados pelo operador de cruzamento. Portanto, cada indivíduo gerado pelo operador de cruzamento é um pai para aplicação do operador de mutação.

5.3.3 Tratamento de Restrições

O engenheiro de software/testador pode determinar restrições que devem ser respeitadas durante o processo evolutivo dos MOEAs. Essas restrições são relativas a tipos de situações que podem levar a uma solução que exige muita complexidade para a integração e o teste. Um exemplo de restrição utilizada na abordagem MECBA, e também na abordagem MECBA-Clu, é não gerar ordens que quebrem dependências de herança, tanto para o contexto de OO quanto de OA, e dependências do tipo de declaração intertipo, no contexto de OA. Em outras palavras os módulos que apresentam dependência de herança ou declaração intertipo devem ser precedidos pelos módulos dos quais dependem.

Na abordagem MECBA, o tratamento de restrições corresponde a verificar do primeiro ao último módulo se existe quebra de precedência entre eles. No caso de existir quebra de restrição, o módulo dependente é enviado para a última posição da ordem, e a posição dos outros módulos é decrementada de um.

Na abordagem MECBA-Clu uma adaptação do tratamento de restrições foi necessário. O tratamento das restrições de precedência é executado em duas etapas: *Intra Cluster* e *Inter Cluster*, uma vez que podem existir restrições entre módulos dentro de um mesmo agrupamento e restrições entre módulos de diferentes agrupamentos.

Durante o tratamento *Intra Cluster* verifica-se se os módulos têm restrição em relação a outros módulos do mesmo agrupamento, e caso exista restrição, o módulo é enviado

para a última posição dentro do agrupamento, e os outros módulos têm sua posição decrementada em um. Este procedimento é executado para todos os agrupamentos, até que todas as precedências existentes dentro dos agrupamentos sejam respeitadas.

No tratamento *Inter Cluster* são verificadas as precedências existentes entre módulos de diferentes agrupamentos, e caso uma restrição seja quebrada, o agrupamento todo é enviado para o fim da ordem, e todos os outros agrupamentos ocupam as posições iniciais da ordem.

A Figura 5.6 apresenta um exemplo do tratamento de restrições utilizado na abordagem MECBA-Clu. Pode-se observar que na Ordem Inicial exitem cinco restrições de precedência indicadas pelas setas direcionadas da esquerda para a direita. Por exemplo, o módulo identificado por 11 depende e deve ser precedido pelo módulo identificado por 9; também o módulo 8 depende e deve ser precedido pelo módulo 1; e assim por diante.

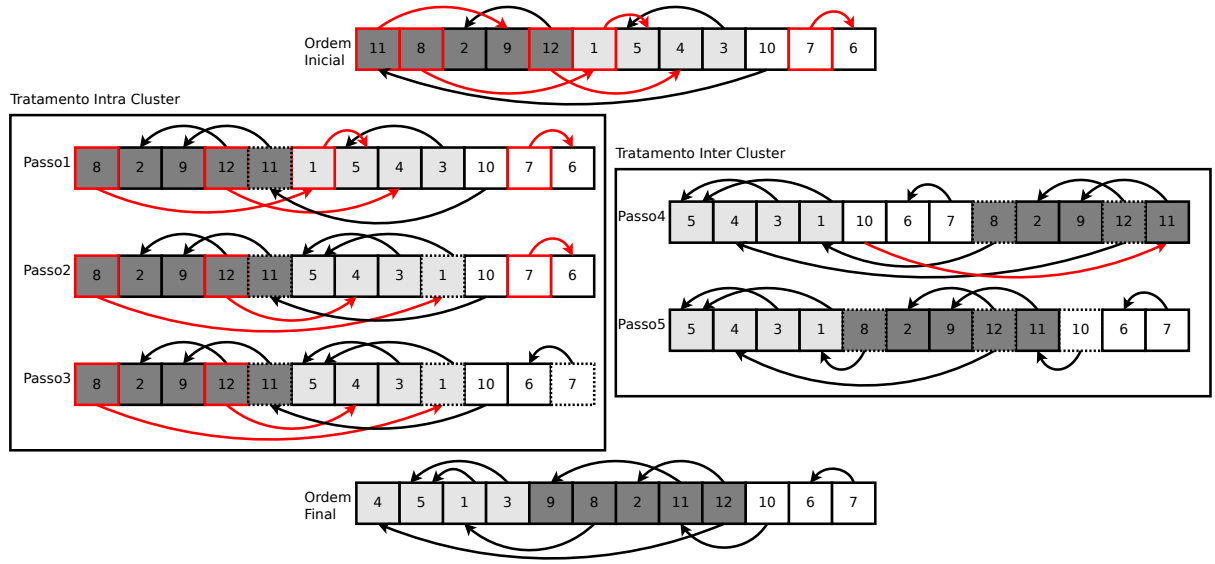


Figura 5.6: Tratamento de Restrições

O tratamento de restrições inicia-se pelo tratamento *Intra Cluster*. No Passo 1, o módulo 11 que depende do módulo 9 é enviado para a última posição do agrupamento, o que passa a satisfazer tal restrição. O módulo 8 depende do módulo 1, que está em outro agrupamento, portanto esta restrição não é tratada nesta etapa. O mesmo acontece com a dependência do módulo 12 para o módulo 4. No Passo 2 é tratada a restrição de precedência entre os módulos 1 e 5, sendo que o módulo 1 é enviado para a última posição

do agrupamento. No Passo 3 o módulo 7, que depende do módulo 6, é enviado para a última posição do agrupamento, satisfazendo a restrição. Neste ponto todas as restrições *Intra Cluster* estão satisfeitas.

No Passo 4 inicia-se a etapa do tratamento *Inter Cluster*. Neste ponto observa-se o tratamento da dependência do módulo 8 para o módulo 1, já que todo o primeiro agrupamento é colocado no fim da ordem. Desta maneira também já é satisfeita a restrição entre o módulo 12 que deve ser precedido pelo módulo 4. Seguindo o procedimento, para satisfazer a restrição entre os módulos 10 e 11, no Passo 5 o segundo agrupamento é colocado no fim da ordem. Neste ponto todas as restrições de precedência são atendidas e a ordem de integração e teste já é uma ordem válida.

Durante o processo evolutivo dos MOEAs, o tratamento de restrições é executado no momento da geração aleatória da população inicial, e após a aplicação dos operadores de cruzamento e mutação.

5.3.4 Ferramenta de Software

Para implementar as características citadas acima, utilizou-se o *framework* JMetal [23]. Este também foi o *framework* utilizado para implementar a abordagem MECBA. JMetal é um *framework* para otimização multiobjetivo, desenvolvido por Nebro e Durillo com a linguagem de programação Java que implementa uma grande variedade de meta-heurísticas.

Este *framework* possui implementado os algoritmos NSGA-II, SPEA2 e PAES apresentados no Capítulo 2. Oferece também vários indicadores de qualidade que são utilizados para comparar o comportamento dos diferentes algoritmos para um mesmo problema.

5.4 Considerações Finais

Este capítulo apresentou a estratégia de integração e teste de módulos baseada em agrupamentos, que consiste em considerar um conjunto de módulos que deve permanecer em sequência.

Para aplicar esta estratégia, foi proposta uma abordagem chamada MECBA-Clu que corresponde a uma extensão da abordagem MECBA. A abordagem MECBA-Clu foi con-

cebida com objetivo de ser genérica, permitindo a sua aplicação em diferentes contextos, como por exemplo o contexto de OO e de OA.

Com objetivo de permitir a utilização real da abordagem MECBA-Clu, foram expostos detalhes da implementação de características que diferem da abordagem MECBA para a abordagem proposta. A representação do problema foi elaborada para permitir que ao mesmo tempo que os módulos estejam agrupados, eles possam ser permutados para encontrar novas soluções. Para encontrar estas novas soluções os operadores genéticos de cruzamento e mutação também sofreram alterações, permitindo tanto alterações *inter cluster* quando *intra cluster*. Um novo tratamento de restrições mais elaborado se fez necessário, a fim de que as precedências de módulos fossem mantidas.

Para verificar o funcionamento real da abordagem MECBA-Clu, utilizando a nova estratégia de agrupamento, no capítulo a seguir é descrito o experimento conduzido.

CAPÍTULO 6

EXPERIMENTO

A seguir são apresentados os detalhes do experimento conduzido para avaliar o comportamento da abordagem MECBA-Clu.

6.1 Sistemas Utilizados

A estratégia de integração e teste de módulos baseada em agrupamentos foi definida para poder ser aplicada no contexto de OO e contexto de OA. Portanto, para cada contexto foram selecionados quatro sistemas que serão utilizados como fonte de informação para utilização da abordagem MECBA-Clu.

No contexto de OO foram utilizados quatro sistemas desenvolvidos em Java, também utilizados em trabalhos similares [3]. São eles: BCEL, JBoss, JHotDraw e MyBatis. O sistema BCEL[1] (*Byte Code Engineering Library*), é uma biblioteca utilizada para analisar, criar e manipular arquivos binários da linguagem de programação Java. Diante da grande quantidade de módulos que o sistema apresenta, neste experimento utilizou-se somente o pacote `org.apache.bcel.classfile` da versão 5.0. O sistema JBoss[2] é um servidor de aplicações Java. Para o experimento utilizou-se o subsistema Management da versão 6.0.0M5. Outro sistema utilizado é um *framework* de gráficos bidimensionais para editores de desenho estruturado chamado de JHotDraw[3]. Para o experimento utilizou-se o pacote `org.jhotdraw.draw` da versão 7.5.1. O último sistema selecionado foi o MyBatis[4], que é um *framework* de mapeamento de classes para aplicações OO que utilizam bancos de dados relacionais. A versão utilizada do sistema MyBatis é a 3.0.2.

No contexto de OA também foram utilizados quatro sistemas, todos desenvolvidos em AspectJ, que também foram utilizados por Colanzi et al. para avaliar a abordagem

[1]BCEL disponível em: http://archive.apache.org/dist/jakarta/bcel/old/v5.0/
[2]JBoss AS disponível em: http://www.jboss.org/jbossas/downloads.html
[3]JHotDraw disponível em: http://sourceforge.net/projects/jhotdraw/
[4]MyBatis disponível em: http://code.google.com/p/mybatis/downloads/list

MECBA [17]. São eles: AJHotDraw, AJHSQLDB, Health-Watcher e Toll System. O sistema AJHotDraw[5] é uma refatoração do sistema OO JHotDraw, utilizando OA. No experimento utilizou-se a versão 0.4 do AJHotDraw. Outro sistema utilizado é o AJSHQLDB[6], versão 18, que também é uma refatoração de um sistema OO chamado HSQLDB. Este sistema é um gerenciador de banco de dados escrito em Java. O sistema Health-Watcher[7], é um sistema para coletar e gerenciar informações de reclamações e notificações de saúde pública. Foi utilizada a versão 9. O último sistema OA utilizado chama-se Toll System[8], que é um sistema inicialmente desenvolvido para demonstrar características do contexto de OA. O Toll System está em sua primeira versão.

Na Tabela 6.1 são apresentadas as características dos sistemas descritos acima. Nas segunda e terceira colunas têm-se o tipo e a versão do sistema, respectivamente. Nas últimas três colunas são apresentadas informações internas dos sistemas. Na coluna quatro tem-se a quantidade de linhas de código, na coluna cinco a quantidade de classes e na coluna seis a quantidade de aspectos, que só existem no contexto de OA.

Tabela 6.1: Sistemas Utilizados no Experimento

Sistema	**Tipo**	**Versão**	**LOC**	**Classes**	**Aspectos**
BCEL	OO	5.0	2999	45	-
JBoss	OO	6.0.0M5	8434	148	-
JHotDraw	OO	7.5.1	20273	197	-
MyBatis	OO	3.0.2	23535	331	-
AJHotDraw	OA	0.4	18586	288	31
AJHSQLDB	OA	18	68550	275	30
Health-Watcher	OA	9	5479	95	22
Toll System	OA	1	2496	49	24

A MECBA-Clu deve ser aplicada durante o projeto do sistema à ser desenvolvido, uma vez que estabelece uma ordem para desenvolvimento e teste de módulos. Porém, é difícil obter documentação da arquitetura de sistemas complexos, portanto, para o experimento utilizou-se engenharia reversa para obter informações de acoplamento. A ferramenta utilizada foi a mesma que utilizada em trabalhos relacionados [3, 17], um

[5]AJHotDraw disponível em: http://sourceforge.net/projects/ajhotdraw/
[6]AJHSQLDB disponível em: http://sourceforge.net/projects/ajhsqldb/files/
[7]HealthWatcher disponível em: http://www.comp.lancs.ac.uk/∼greenwop/tao/aspectj.htm
[8]Toll System disponível em: http://www.aosd-europe.net/

parser baseado na ferramenta AJATO[9] (*AspectJ and Java Assessment Tool*). Este *parser* tem como entrada o código fonte Java ou AspectJ e retorna uma arvore sintática, usada para construir os modelos de dependência de cada sistema.

6.2 Definição dos Agrupamentos de Módulos

Como já exposto no Capítulo 5, o engenheiro de software pode determinar um regra de divisão de agrupamentos com base em diferentes aspectos do desenvolvimento de software. Para executar os experimentos deste trabalho, utilizou-se uma regra a fim de padronizar a definição dos agrupamentos entre os oito sistemas utilizados.

O procedimento consiste em montar um grafo direcionado utilizando somente as dependências de herança e declarações intertipo, considerando assim apenas as dependências que não podem ser quebradas. O grafo foi construído utilizando a linguagem DOT[10]. A partir deste grafo, aplicou-se a ferramenta *ccomps* do software Graphviz [25]. A ferramenta *cccomps* separa os componentes conectados do grafo, ou seja, separa os módulos que têm relação, formando subgrafos. A partir dos subgrafos gerados, foi estabelecida uma regra para divisão dos agrupamentos. A Figura 6.1 apresenta um fragmento dos subgrafos gerados para o sistema Toll System. Esta figura será utilizada para exemplificar a regra utilizada para definição dos agrupamentos para o experimento.

Na Figura 6.1 observa-se que foram gerados diversos subgrafos. Existem subgrafos com um módulo, como por exemplo: {1}, {4}, {5}, etc.; subgrafos com dois módulos, como por exemplo: {3, 2} e {28, 29}; subgrafos com três módulos, como por exemplo: {7, 8, 9} e {10, 11, 12}; e outros subgrafos com mais módulos, como por exemplo {22, 23, 24, 25, 26} e {13, 14, 15, 16, 17, 18, 19}. Com base nestes subgrafos, observou-se que agrupamentos com menos de quatro módulos não teriam representatividade para a aplicação da abordagem MECBA-Clu, pois os operadores genéticos implementados não teriam bons comportamentos. Portanto, os módulos dos subgrafos de tamanho um, foram agrupados todos em um grupo, como observado no agrupamento A1 do exemplo. Também os sub-

[9] AJATO disponível em: http://www.teccomm.les.inf.puc-rio.br/emagno/ajato/
[10] http://www.graphviz.org/content/dot-language

grafos de tamanho dois formaram um único agrupamento, identificado no exemplo por A2. Os subgrafos de tamanho três formaram o agrupamento A3. Já os outros subgrafos com tamanho igual ou superior a quatro, foram separados para formarem seus próprios agrupamentos, como é o caso dos agrupamentos A4 e A5 do exemplo.

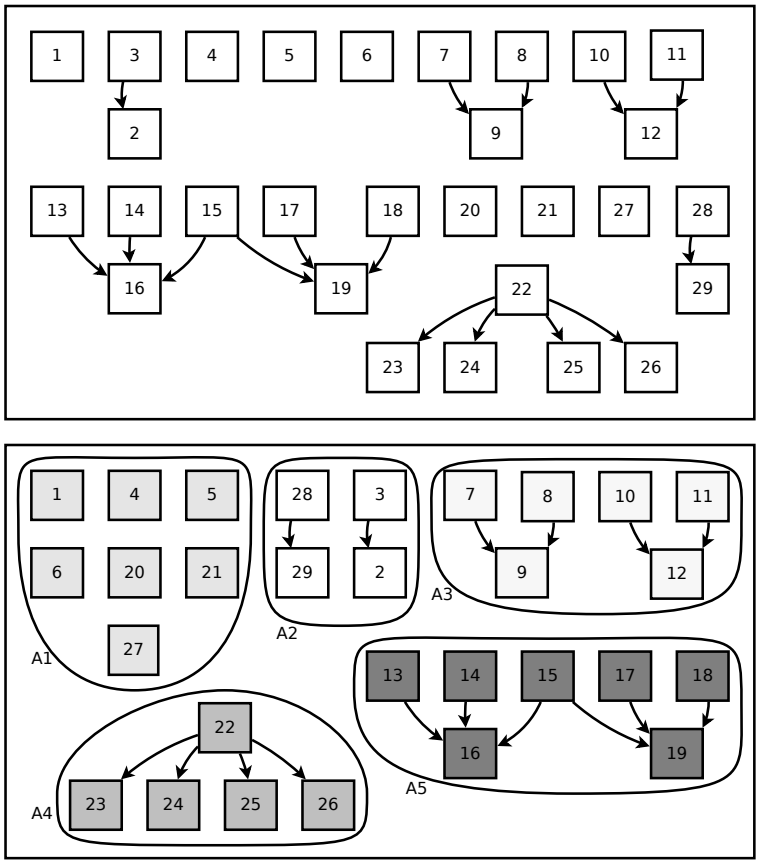


Figura 6.1: Exemplo da Regra para Definição de Agrupamentos

Esta regra foi utilizada para todos os sistemas. A quantidade de agrupamentos definidos para cada sistema é apresentada na Tabela 6.2.

Tabela 6.2: Quantidade de Agrupamentos de Cada Sistema

Sistema	**# Agrupamentos**
BCEL	3
JBoss AS	8
JHotDraw	13
MyBatis	24
AJHotDraw	12
AJHSQLDB	15
Health-Watcher	7
Toll System	7

6.3 Medidas de Acoplamento

O modelo de custo da abordagem MECBA-Clu é utilizado para mensurar o custo necessário para utilizar determinada ordem de módulos encontrada por um MOEA. Na literatura (ver Capítulo 4) as medidas de acoplamento são largamente utilizadas por terem influência direta na complexidade de *stubs* que devem ser construídos. No experimento deste trabalho essas mesmas medidas foram utilizadas, tratando-se dois cenários a fim de verificar o comportamento dos MOEAs. Os dois cenários são:

- Utilização de duas medidas tradicionais: (i) número de atributos; e (ii) número de métodos. Estas medidas foram inicialmente utilizadas por Briand et al. [9] no contexto OO e por Galvan et al. [24] no contexto OA.

- Utilização de quatro medidas, formadas pelas duas medidas tradicionais: (i) número de atributos; e (ii) número de métodos, e por mais duas medidas que também têm influência na complexidade de *stubs* a serem constuídos: (iii) número de tipos distintos de retornos; e (iv) número de tipos distintos de parâmetros. Estas medidas foram inicialmente tratadas por Assunção et al. [3] no contexto OO e por Colanzi et al. [17] no contexto OA.

A definição formal de cada uma das quatro medidas é apresentada no Capítulo 4, Seção 4.3.3.1 que apresenta detalhes da abordagem MECBA.

6.4 Configuração de Parâmetros dos MOEAs

Os parâmetros para os MOEAs foram ajustados a fim de que executem cada qual com seu processo evolutivo específico, mas diante de um ambiente similar. A base para este ajuste foram os trabalhos que utilizam os mesmos algoritmos e os mesmos sistemas que os utilizados neste experimento [3, 4, 6, 5, 17]. Os valores adotados são apresentados na Tabela 6.3.

O número de avaliações de *fitness* é utilizado como critério de parada. Como todos os algoritmos utilizam a mesma quantidade de avaliações de *fitness*, através deste mesmo

Tabela 6.3: Valores dos Parâmetros Utilizados pelos MOEAs

Parâmetro	**MECBA-Clu**			**MECBA**		
	NSGA-II	**PAES**	**SPEA2**	**NSGA-II**	**PAES**	**SPEA2**
Tamanho da População	300	300	300	300	300	300
Taxa de Cruzamento	0,95	-	0,95	0,95	-	0,95
Taxa de Cruzamento *Inter Cluster*	1,0	-	1,0	-	-	-
Taxa de Cruzamento *Intra Cluster*	1,0	-	1,0	-	-	-
Taxa de Mutação	0,02	1	0,02	0,02	1	0,02
Taxa de Mutação *Inter Cluster*	0,5	0,5	0,5	-	-	-
Taxa de Mutação *Intra Cluster*	0,5	0,5	0,5	-	-	-
Tamanho do Arquivo Externo	-	250	250	-	250	250
Número de Avaliações de *Fitness*	60000	60000	60000	60000	60000	60000

esforço é possível analisar o desempenho de cada algoritmo. Cada MOEA foi executado trinta vezes para cada sistema, para verificar o seu real comportamento.

6.5 Ferramentas para os Indicadores de Qualidade

As ferramentas descritas a seguir foram utilizadas para obter os indicadores de qualidade das soluções encontradas pelos MOEAs, além de oferecer recursos para facilitar a análise dos dados.

Além de ser utilizado para implementar os MOEAs, o *framework* JMetal [23] também oferece vários indicadores de qualidade para serem utilizados com os resultados. Dentre esses, destacam-se os indicadores de Distância Geracional e Distância Geracional Invertida utilizados neste trabalho. Também utilizando os recursos oferecidos pelo *framework*, implementou-se o indicador Distância Euclidiana da Solução Ideal.

Uma ferramenta chamada GUIMOO [42] (*Graphical User Interface for Multi Objective Optimization*) foi utilizada para o cálculo do indicador de Cobertura. Esta ferramenta utiliza o mesmo padrão de arquivos do JMetal, facilitando a obtenção do indicador. Além do indicador de Cobertura, a ferramenta GUIMOO também permite gerar gráficos das soluções no espaço de busca, facilitando a visualização e interpretação dos resultados.

Para as comparações entre os MOEAs através de médias de indicadores, utilizou-se o software `R` [47] para executar o teste de Friedman [27], de modo a verificar se os MOEAs são considerados estatisticamente diferentes ($p - value \leq 0,05$).

6.6 Resultados e Análises

A seguir são apresentados os resultados do experimento, conduzido a fim de verificar o comportamento dos algoritmos quando aplicados na otimização do problema CITO/CAITO, utilizando a estratégia de agrumamento de módulos.

Inicialmente são expostos os resultados relativos tanto a abordagem MECBA quanto a abordagem MECBA-Clu, com objetivo de analisar a diferença no espaço de busca entre ambas abordagens. A abordagem MECBA foi aplicada aos mesmos sistemas e parâmetros utilizados com a abordagem MECBA-Clu, por isto os resultados apresentados aqui diferem daqueles apresentados nos trabalhos relacionados e descritos no Capítulo 4. Posteriormente, são comparados os MOEAs para verificar qual tem melhor o comportamento para solucionar o problema, utilizando a estratégia de agrupamento de módulos.

6.6.1 Obtenção dos Conjuntos de Soluções

Para calcular os indicadores de qualidade é necessário utilizar diferentes conjuntos de soluções encontradas, conjuntos estes obtidos de diferentes maneiras. A definição destes conjuntos já foi apresentada na Seção 2.3 do Capítulo 2. A seguir é descrito como foi obtido cada um dos conjuntos a serem utilizados nas análises dos resultados.

- PF_{approx}: para o experimento, o conjunto PF_{approx} de determinado sistema é obtido em cada execução de cada algoritmo. Portanto, como cada algoritmo é executado trinta vezes para cada problema, ao final das execuções obtêm-se trinta conjuntos PF_{approx}.

- PF_{known}: este conjunto é formado através da união de todas as soluções dos trinta conjuntos PF_{approx}, eliminando-se as soluções dominadas e repetidas. Já que os algoritmos têm aleatoriedade em seu processo evolutivo, então cada execução pode encontrar soluções diferentes. Diante disto, é no conjunto PF_{known} que todas as melhores soluções encontradas por um MOEA estão disponíveis.

- PF_{true}: este conjunto é obtido através da união de todas as soluções dos conjuntos

PF_{known} de todos os MOEAs utilizados no experimento, eliminando-se as soluções dominadas e repetidas [65]. Uma vez que o conjunto PF_{known} tem as melhores soluções de determinado MOEA, então juntar todas estas soluções implica em ter todas as melhores soluções conhecidas para o problema.

6.6.2 Comparação entre as Abordagens MECBA e MECBA-Clu

As abordagens MECBA e MECBA-Clu são duas propostas diferenciadas para solucionar os problemas CITO e CAITO, portanto seus resultados não podem ser comparados assumindo que deveriam ter um comportamento similar.

O objetivo desta seção é apresentar os resultados e analisar qual o impacto, em termos de complexidade, que é introduzido na solução dos problemas CITO e CAITO quando a consideração dos agrupamentos de módulos é introduzida. Como o experimento foi executado com duas e quatro medidas (ver Seção 6.3), então esta análise é feita para as duas situações.

Para analisar a diferença de complexidade para se obter soluções nas abordagens MECBA e MECBA-Clu, utilizaram-se dois dados relativos a execução dos MOEAs: (i) número de soluções encontradas; e (ii) tempo de execução.

A Tabela 6.4 apresenta os dados do número de soluções encontradas e tempo de execução para duas (A e O) e quatro medidas (A, O, R e P). Nas terceira e sexta colunas é apresentada a cardinalidade do conjunto PF_{true}. Nas quarta e sétima colunas é apresentada a média da quantidade de soluções dos conjuntos PF_{approx} e entre parênteses a cardinalidade do conjunto PF_{known}. Nas quinta e oitava colunas são apresentadas, respectivamente, a média do tempo de execução, em segundos, utilizados para obter cada PF_{approx} e o desvio padrão entre parênteses.

Sobre a quantidade de soluções encontradas, pode-se verificar através do tamanho do conjunto PF_{true}, considerando as duas medidas (A e O), que para os sistemas BCEL, MyBatis, AJHotDraw e AJHSQLDB o número de soluções encontradas pela MECBA-Clu foi menor que o número de soluções encontradas pela MECBA. Já para os sistemas Health Watcher e Toll System o número de soluções encontradas pela MECBA-Clu foi maior que

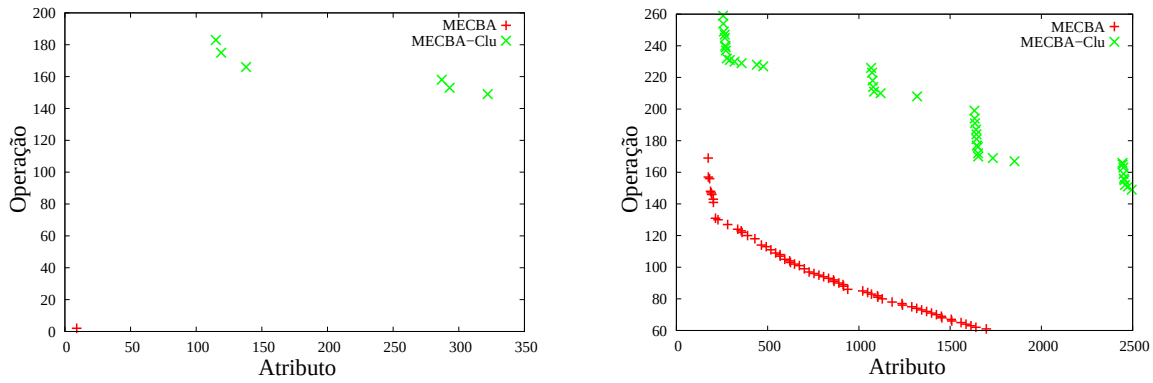
Tabela 6.4: Número de Soluções e Tempo de Execução

Medidas A e O							
Sistema	**MOEA**	**MECBA**			**MECBA-Clu**		
		# PF_{true}	**Número de Soluções**	**Tempo de Execução**	**#** PF_{true}	**Número de Soluções**	**Tempo de Execução**
BCEL	NSGA-II	29	28,73 (29)	5,37 (0,04)	5	1,17 (4)	7,49 (0,12)
	PAES		25,70 (29)	1,88 (0,07)		1,13 (1)	3,21 (0,16)
	SPEA2		28,93 (29)	184,51 (21,88)		1,07 (3)	3676,41 (203,81)
JBoss	NSGA-II	1	1,00 (1)	19,25 (0,18)	1	1,10 (1)	40,79 (0,37)
	PAES		1,00 (1)	10,46 (0,08)		1,23 (1)	21,17 (0,40)
	SPEA2		1,00 (1)	2666,66 (585,32)		1,07 (1)	3888,54 (167,14)
JHotDraw	NSGA-II	3	1,40 (3)	31,30 (0,18)	3	4,27 (3)	70,60 (0,20)
	PAES		1,83 (2)	19,06 (0,13)		3,40 (3)	38,37 (0,24)
	SPEA2		1,23 (2)	3213,17 (677,40)		4,00 (8)	1977,26 (339,40)
MyBatis	NSGA-II	63	60,60 (63)	79,18 (0,33)	42	17,97 (29)	183,87 (0,53)
	PAES		43,00 (54)	52,30 (0,20)		30,80 (42)	104,93 (0,40)
	SPEA2		58,60 (57)	132,41 (15,55)		15,13 (37)	1043,54 (221,12)
AJHotDraw	NSGA-II	7	4,57 (6)	81,44 (0,41)	4	2,13 (4)	187,48 (0,92)
	PAES		5,87 (7)	53,53 (0,31)		2,90 (4)	105,49 (0,94)
	SPEA2		4,63 (6)	1375,29 (418,06)		2,13 (3)	2181,10 (323,16)
AJHSQLDB	NSGA-II	40	31,53 (35)	67,13 (0,21)	20	5,57 (2)	152,61 (0,85)
	PAES		26,23 (40)	44,57 (0,23)		11,77 (20)	87,76 (0,45)
	SPEA2		26,40 (36)	101,65 (3,86)		5,73 (20)	1248,60 (334,99)
Health Watcher	NSGA-II	1	1,00 (1)	13,00 (0,17)	6	5,83 (6)	26,48 (0,09)
	PAES		1,07 (1)	6,72 (0,07)		3,57 (6)	13,55 (0,32)
	SPEA2		1,00 (1)	2897,21 (744,19)		6,00 (6)	1511,87 (177,33)
Toll System	NSGA-II	1	1,00 (1)	7,10 (0,08)	2	1,67 (2)	12,20 (0,16)
	PAES		1,00 (1)	2,72 (0,02)		1,67 (2)	5,55 (0,20)
	SPEA2		1,00 (2)	3541,92 (804,71)		1,73 (2)	3434,27 (278,99)
Medidas A, O, R e P							
Sistema	**MOEA**	**MECBA**			**MECBA-Clu**		
		# PF_{true}	**Número de Soluções**	**Tempo de Execução**	**#** PF_{true}	**Número de Soluções**	**Tempo de Execução**
BCEL	NSGA-II	37	37,43 (37)	5,91 (0,05)	15	7,57 (11)	8,61 (0,11)
	PAES		39,30 (37)	6,58 (1,25)		3,40 (8)	29,89 (22,25)
	SPEA2		36,70 (37)	123,07 (18,84)		8,53 (19)	3786,79 (476,23)
JBoss	NSGA-II	1	1,00 (1)	18,73 (0,20)	2	1,97 (2)	42,50 (0,47)
	PAES		1,13 (1)	10,69 (0,62)		2,87 (2)	56,15 (12,50)
	SPEA2		1,00 (1)	2455,35 (612,18)		2,17 (2)	3536,01 (335,97)
JHotDraw	NSGA-II	11	8,40 (10)	29,85 (0,34)	153	45,80 (110)	71,90 (0,45)
	PAES		10,47 (19)	24,29 (1,50)		85,47 (143)	51,18 (2,82)
	SPEA2		9,63 (9)	922,99 (373,98)		49,17 (102)	532,83 (81,93)
MyBatis	NSGA-II	789	276,37 (941)	74,03 (0,87)	200	72,60 (103)	189,91 (0,83)
	PAES		243,60 (679)	104,30 (7,91)		108,43 (200)	132,37 (3,91)
	SPEA2		248,77 (690)	128,88 (2,65)		64,33 (144)	517,52 (67,52)
AJHotDraw	NSGA-II	94	70,03 (79)	75,05 (0,57)	31	16,30 (36)	194,34 (0,83)
	PAES		40,73 (84)	62,07 (2,16)		26,57 (31)	115,12 (2,82)
	SPEA2		68,87 (78)	195,56 (28,22)		17,53 (31)	1005,36 (268,37)
AJHSQLDB	NSGA-II	266	156,63 (360)	62,34 (0,53)	240	62,07 (196)	160,38 (1,64)
	PAES		145,97 (266)	75,62 (5,27)		122,57 (240)	122,01 (4,92)
	SPEA2		119,10 (52)	104,29 (0,68)		58,30 (170)	505,11 (101,90)
Health Watcher	NSGA-II	1	1,00 (1)	12,72 (0,15)	11	10,70 (11)	27,52 (0,10)
	PAES		1,07 (1)	8,27 (0,58)		7,47 (12)	46,98 (5,34)
	SPEA2		1,00 (1)	2580,39 (596,29)		10,20 (11)	990,19 (95,94)
Toll System	NSGA-II	1	1,00 (1)	7,33 (0,09)	4	4,27 (4)	13,23 (0,09)
	PAES		1,07 (1)	4,10 (0,75)		3,50 (4)	31,13 (16,23)
	SPEA2		1,00 (1)	3516,71 (570,76)		4,00 (4)	2229,26 (271,47)

o número de soluções encontradas pela MECBA. Nos sistemas JBoss e JHotDraw não houve diferença. Considerando as quatro medidas (A, O, R e P), verifica-se uma situação semelhante, com exceção dos sistemas JBoss e JHotDraw para os quais a MECBA-Clu

encontrou mais soluções que a MECBA. Em resumo, observa-se em ambas situações, tanto para a duas ou quatro medidas, que para os sistemas que apresentam várias soluções encontradas pela MECBA, menos soluções são encontradas pela MECBA-Clu, e o contrário para sistemas com poucas soluções.

A explicação para este comportamento está nas características que a estratégia de agrupamentos introduz no espaço de busca. Quando é necessário considerar agrupamentos para encontrar uma nova solução, o espaço de busca fica limitado a determinadas regiões que satisfazem as restrições. Para exemplificar, na Figura 6.2 são apresentadas as soluções das abordagens MECBA e MECBA-Clu no espaço de busca, para os sistemas Health Watcher e MyBatis, utilizando duas medidas. As soluções apresentadas correspondem ao conjunto PF_{true} de cada sistema.



(a) Sistema Health Watcher (b) Sistema MyBatis

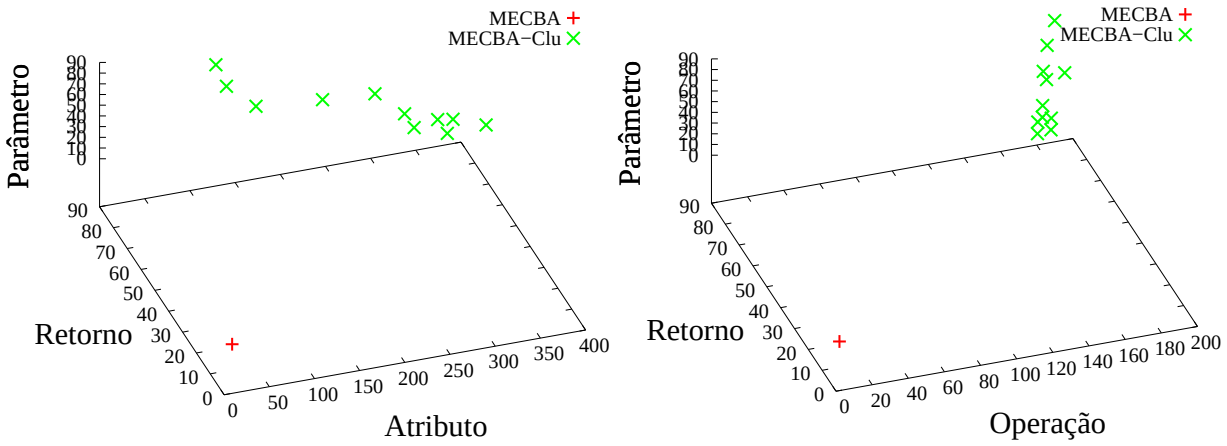
Figura 6.2: PF_{true}'s no Espaço de Busca, Medidas A e O

Na Figura 6.2(a) verifica-se que a solução encontrada pela MECBA está próxima dos objetivos mínimos (A=0, B=0) entretanto, diante das restrições dos agrupamentos, esta solução sub-ótima não é possível na abordagem MECBA-Clu. A partir disto os MOEAs exploram novos locais no espaço de busca, possibilitando encontrar uma maior quantidade de soluções.

Por outro lado, na Figura 6.2(b) observa-se que as soluções encontradas pela MECBA formam uma fronteira quase contínua no espaço de busca, mas as soluções encontradas pela MECBA-Clu estão agrupadas em diferentes regiões do espaço de busca, como visto na Figura 6.2(a), pois respeitam as restrições da estratégia. Uma vez que o espaço de

busca é restrito, não existe a possibilidade de encontrar a mesma grande quantidade de soluções da MECBA, portanto uma quantidade menor de soluções foi encontrada.

Considerando as quatro medidas (A, O, R e P), a Figura 6.3 apresenta as soluções do Health Watcher no espaço de busca. Apesar de se estar utilizando quatro medidas, a apresentação gráfica só é possível utilizando três medidas por gráfico. Através da análise dos dois gráficos, verifica-se o mesmo comportamento encontrado quando utilizando-se duas medidas. A solução encontrada pela MECBA está próxima dos objetivos mínimos (A=0, O=0, R=0, P=0), entretanto as restrições dos agrupamentos não permitem esta solução, levando os MOEAs a encontrarem soluções em outras regiões do espaço de busca, onde um maior número de soluções é possível, aumentando consequentemente o número de soluções encontradas.



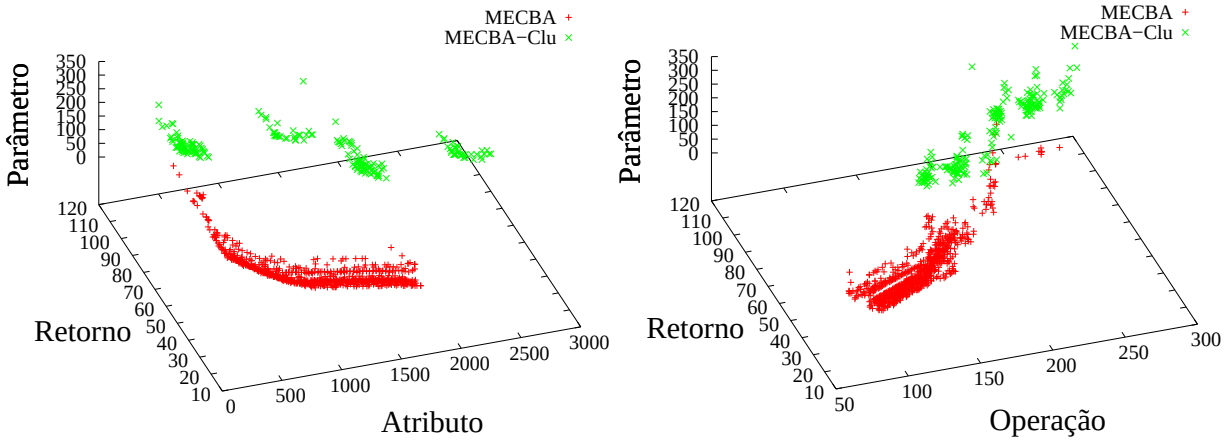
(a) Medidas A, R e P

(b) Medidas O, R e P

Figura 6.3: PF_{true}'s do Sistema Health Watcher no Espaço de Busca

A Figura 6.4 apresenta as soluções no espaço de busca para o sistema MyBatis. Assim como observado com duas medidas, pode-se verificar o motivo da abordagem MECBA-Clu encontrar menos soluções que a abordagem MECBA. Através dos gráficos das Figuras 6.4(a) e 6.4(b) verifica-se que as soluções encontradas pela MECBA estão localizadas em uma mesma região do gráfico, mais próximas aos objetivos mínimos. Entretanto, devido as restrições dos agrupamentos, estas soluções não são possíveis, portanto os MOEAs exploram novas regiões do espaço de busca. Tais restrições dificultam a obtenção de soluções, diminuindo a quantidade de soluções encontradas. Pode-se observar claramente

nos gráficos da Figura 6.4 que somente algumas regiões do espaço de busca satisfazem às restrições do problema, por isso as soluções ficaram concentradas em determinadas regiões.



(a) Medidas A, R e P

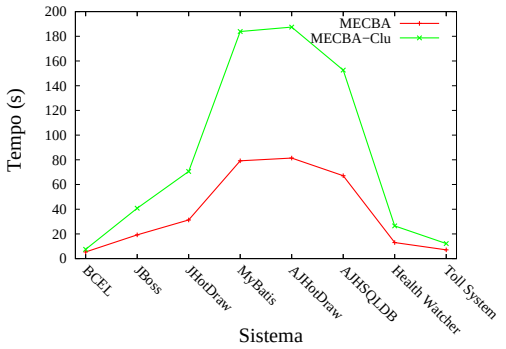
(b) Medidas O, R e P

Figura 6.4: PF_{true}'s do Sistema MyBatis no Espaço de Busca

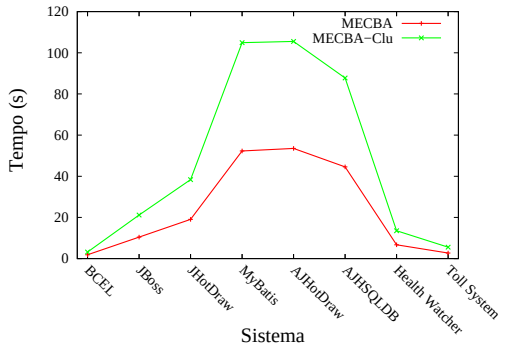
Fica claro então que a utilização da estratégia com agrupamentos deixa a busca por soluções mais complexa, limitando o espaço de busca.

Em relação ao tempo de processamento (quinta e oitava colunas da Tabela 6.4), observa-se que apesar do esforço ser o mesmo (Número de Avaliações de *Fitness*), houve grande diferença de tempo entre as abordagens. Os gráficos das Figuras 6.5 e 6.6 apresentam a comparação do tempo de execução entre MECBA e MECBA-Clu.

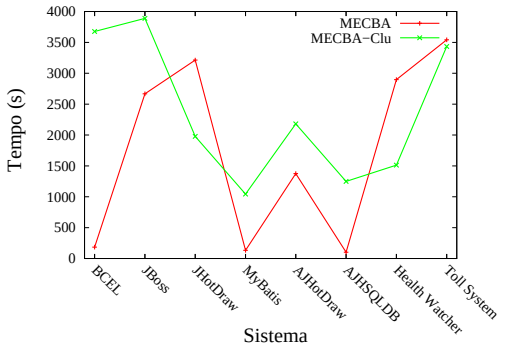
Em todos os sistemas, o NSGA-II (Figura 6.5(a)) e o PAES (Figura 6.5(b)) demoraram mais na abordagem MECBA-Clu. O SPEA2 (Figura 6.5(c)) na abordagem MECBA-Clu foi mas rápido que na MECBA para os sistemas JHotDraw, Health Watcher e Toll System, já para os outros sistemas a abordagem MECBA-Clu demorou mais para executar. Dos três MOEAs, o SPEA2 foi o único que teve comportamento diferenciado para três sistemas e também foi o MOEA que mais demorou para finalizar as execuções, o que leva a inferir que em alguns casos seu comportamento fica muito aleatório quando submetido a muitas restrições do espaço de busca.



(a) NSGA-II

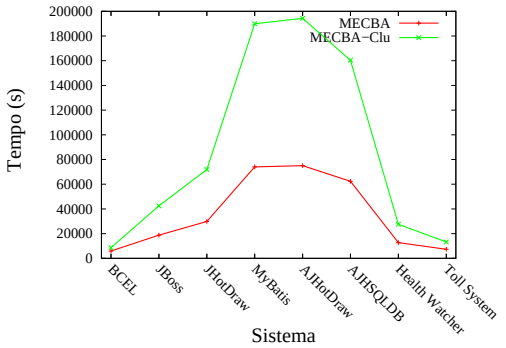


(b) PAES

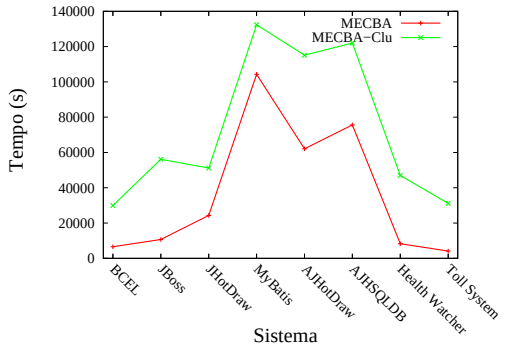


(c) SPEA2

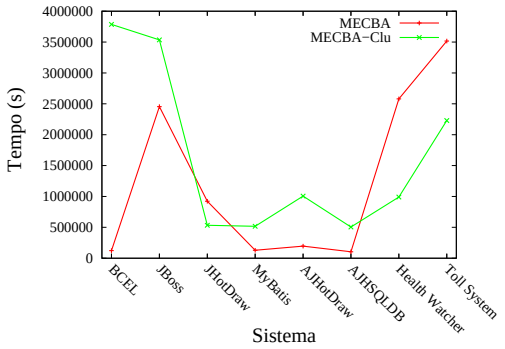
Figura 6.5: Tempo de Execução, Medidas A e O



(a) NSGA-II



(b) PAES



(c) SPEA2

Figura 6.6: Tempo de Execução, Medidas A, O, R e P

6.6.3 Comparação entre os MOEAs na Abordagem MECBA-Clu

Nesta seção são apresentados os resultados dos quatro indicadores de qualidade utilizados para comparação dos MOEAs. São eles: (i) Cobertura; (ii) Distância Geracional; (iii) Distância Geracional Invertida; e (iv) Distância Euclidiana da Solução Ideal. Estes indicadores já foram apresentados no Capítulo 2, Seção 2.3.

6.6.3.1 Resultados para o Indicador de qualidade C

A Tabela 6.5 apresenta os valores do indicador de Cobertura para os conjuntos PF_{known} de cada MOEA para cada sistema. Este indicador demonstra quanto um conjunto de soluções domina outro conjunto de soluções. A leitura da tabela é feita por linha/coluna, no qual o MOEA que aparece na linha domina em determinado valor o MOEA que aparece na coluna. Os valores estão em um intervalo de 0 a 1, onde 0 indica que não existe dominação e 1 indica que todo o conjunto é dominado. Só são significantes os valores acima de 0,5, pois isto indica que mais de 50% do conjunto é dominado.

Considerando duas medidas, os resultados da Cobertura demonstram que, existe diferença diferença entre os MOEAs para quatro sistemas: JHotDraw, MyBatis, AJHotDraw e AJHSQLDB. Para os sitema JHotDraw as soluções do NSGA-II dominam todas as soluções do SPEA2 e as soluções do PAES domina 75% das soluções do SPEA2, já o NSGA-II e PAES são equivalentes. Para os Sistemas MyBatis, AJHotDraw e AJHSQLDB a mesma situação é observada, no qual as soluções do NSGA-II dominam todas as soluções do SPEA2 e as soluções do PAES dominam todas as soluções do NSGA-II e SPEA2. Nos outros casos os MOEAs são equivalentes.

Considerando quatro medidas, os resultados deste indicador demonstram diferença entre os MOEAS para cinco sistemas: BCEL, MyBatis, AJHotDraw, AJHSQLDB e Toll System. Através da análise da Tabela 6.5 verifica-se que para o sistema BCEL, as soluções do NSGA-II dominam 75% das soluções do PAES e aproximadamente 60% das soluções do SPEA2. As soluções do SPEA2 também dominam 75% das soluções do PAES. Para o sistema MyBatis, as soluções do PAES dominam todas as soluções do NSGA-II e do SPEA2 e as soluções do NSGA-II dominam aproximadamente 73% das soluções do SPEA2. Para o

Tabela 6.5: Valores do Indicador C

Medidas A e O								
Sistema	**MOEA**	**NSGA-II**	**PAES**	**SPEA2**	**Sistema**	**NSGA-II**	**PAES**	**SPEA2**
BCEL	**NSGA-II**	-	0	0,3333	AJHotDraw	-	0	1
	PAES	0	-	0		1	-	1
	SPEA2	0	0	-		0	0	-
JBoss	**NSGA-II**	-	0	0	AJHSQLDB	-	0	1
	PAES	0	-	0		1	-	1
	SPEA2	0	0	-		0	0	-
JHotDraw	**NSGA-II**	-	0,3333	1	Health Watcher	-	0	0
	PAES	0	-	0,75		0	-	0
	SPEA2	0	0,3333	-		0	0	-
MyBatis	**NSGA-II**	-	0	1	Toll System	-	0	0
	PAES	1	-	1		0	-	0
	SPEA2	0	0	-		0	0	-
Medidas A, O, R e P								
Sistema	**MOEA**	**NSGA-II**	**PAES**	**SPEA2**	**Sistema**	**NSGA-II**	**PAES**	**SPEA2**
BCEL	**NSGA-II**	-	0,75	0,578947	AJHotDraw	-	0	0,16129
	PAES	0	-	0		1	-	1
	SPEA2	0,181818	0,75	-		0,666667	0	-
JBoss	**NSGA-II**	-	0	0	AJHSQLDB	-	0	0,117647
	PAES	0	-	0		1	-	1
	SPEA2	0	0	-		0,540816	0	-
JHotDraw	**NSGA-II**	-	0,027972	0,166667	Health Watcher	-	0,166667	0
	PAES	0,427273	-	0,45098		0	-	0
	SPEA2	0,345455	0,020979	-		0	0,166667	-
MyBatis	**NSGA-II**	-	0	0,729167	Toll System	-	0,5	0
	PAES	1	-	1		0	-	0
	SPEA2	0,349515	0	-		0	0,5	-

sistema AJHotDraw, as soluções do PAES também dominam todas as solução do NSGA-II e SPEA2, entretanto as soluções do SPEA2 dominam aproximadamente 66% das soluções do NSGA-II. Para o sistema AJHSQLDB, é observado um comportamento similar ao observado para o sitema AJHotDraw, mas com as soluções do SPEA2 dominando um pouco menos as soluções NSGA-II (54%). Para o sistema Toll System, as soluções tanto do NSGA-II quanto do SPEA2 dominam 50% das soluções do PAES.

Baseado no indicador Cobertura, considerando-se duas medidas, verifica-se que o PAES e o NSGA-II tiveram os melhores resultados, seguidos pelo SPEA2. Considerando-se quatro medidas o NSGA-II teve os melhores resultados, seguido pelo SPEA2 e por fim o PAES.

6.6.3.2 Resultados para os Indicadores de qualidade GD e IGD

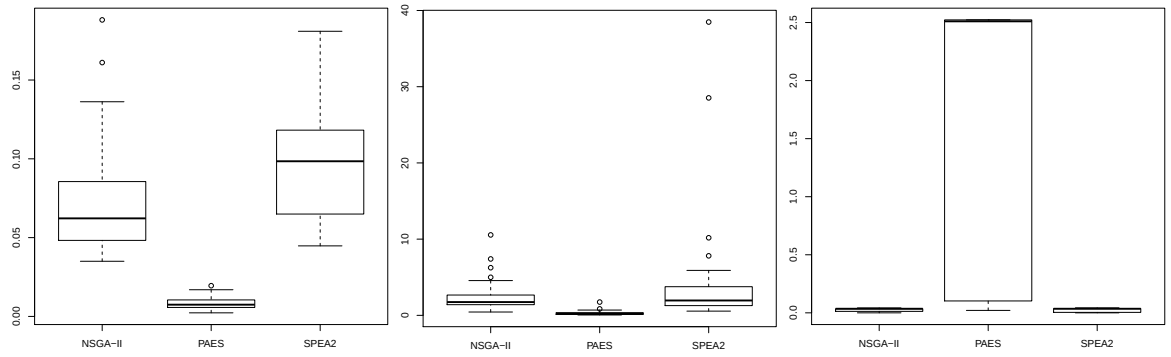
A Tabela 6.6 apresenta os resultados dos indicadores GD e IGD. Estes resultados correspondem a média e desvio padrão dos valores de GD e IGD obtidos na comparação de cada um dos trinta PF_{approx} em relação ao conjunto PF_{true} de cada sistema. Para verificar a diferença significativa entre os resultados dos MOEAs, utilizou-se o teste estatístico de Friedman utilizando $\alpha = 0,05$, ou seja, o teste indica diferença com 95% de certeza.

Tabela 6.6: Média e Desvio Padrão de GD e IGD

Medidas A e O							
Indicador	**Sistema**	**NSGA-II**		**PAES**		**SPEA2**	
		Média	**Desvio Padrão**	**Média**	**Desvio Padrão**	**Média**	**Desvio Padrão**
GD	BCEL	0,2888	0,8553	0,7619	2,1855	0,1388	0,6824
	JBoss	1,0296	3,1415	1,9434	3,8951	0,6864	2,6121
	JHotDraw	0,3720	0,2746	0,2066	0,0397	0,3642	0,2418
	MyBatis	0,0751	0,0378	**0,0085**	0,0040	0,0975	0,0340
	AJHotDraw	1,5608	1,8485	1,6239	1,8858	3,0483	3,5933
	AJHSQLDB	2,5780	2,2159	**0,3095**	0,3423	4,6926	8,2254
	Health Watcher	**0,0243**	0,0149	1,4052	1,2165	**0,0250**	0,0167
	Toll System	0,6374	1,0567	1,1504	3,2555	0,6043	0,7542
IGD	BCEL	0,0234	0,0864	0,6674	2,1874	0,0220	0,1118
	JBoss	0,7450	2,2732	1,4202	2,8244	0,4967	1,8901
	JHotDraw	0,3117	0,1178	**0,2203**	0,0467	0,3136	0,1830
	MyBatis	0,1089	0,0769	**0,0074**	0,0042	0,1497	0,0799
	AJHotDraw	**1,6874**	2,1683	**1,2220**	1,7638	3,1557	3,9410
	AJHSQLDB	4,7587	5,1358	**0,2346**	0,2199	8,0318	14,2419
	Health Watcher	0,0093	0,0143	0,0466	0,0644	0,0120	0,0171
	Toll System	0,6311	1,0331	1,1184	3,1199	0,5974	0,7412
Medidas A, O, R e P							
Indicador	**Sistema**	**NSGA-II**		**PAES**		**SPEA2**	
		Média	**Desvio Padrão**	**Média**	**Desvio Padrão**	**Média**	**Desvio Padrão**
GD	BCEL	**0,1334**	0,0349	1,9266	1,2256	**0,1257**	0,0387
	JBoss	0,7956	1,6492	1,1045	1,9222	0,5147	1,2308
	JHotDraw	0,0132	0,0039	**0,0076**	0,0021	0,0126	0,0026
	MyBatis	0,0444	0,0125	**0,0082**	0,0026	0,0479	0,0160
	AJHotDraw	0,0471	0,0174	**0,0150**	0,0063	0,0431	0,0133
	AJHSQLDB	0,1064	0,0508	**0,0164**	0,0057	0,1117	0,0507
	Health Watcher	**0,0215**	0,0176	0,3165	0,4961	**0,0248**	0,0163
	Toll System	4,3433	22,7890	0,3727	0,6018	0,2199	0,2045
IGD	BCEL	0,2205	0,0420	0,9239	1,2194	0,2024	0,0345
	JBoss	0,6312	1,2291	0,8242	1,4220	0,4361	0,9378
	JHotDraw	0,0254	0,0102	**0,0135**	0,0054	0,0229	0,0069
	MyBatis	0,0512	0,0223	**0,0104**	0,0034	0,0578	0,0191
	AJHotDraw	0,0618	0,0310	**0,0137**	0,0072	0,0555	0,0295
	AJHSQLDB	0,1704	0,1218	**0,0205**	0,0091	0,1757	0,0847
	Health Watcher	**0,0110**	0,0160	0,2206	0,2024	**0,0077**	0,0156
	Toll System	4,3298	22,7914	0,3033	0,4469	0,1958	0,1306

Considerando duas medidas, o teste estatístico apresentou diferença no indicador GD para os sistemas: MyBatis, AJHSQLDB e HealthWatcher. Para o indicador IGD o teste apresentou diferença entre os sitemas: JHotDraw, MyBatis, AJHotDraw e AJHSQLDB.

Diante da diferença apresentada pelo teste de Friedman, para determinar os melhores MOEAs considerando no indicador GD, na Figuras 6.7 são apresentados os gráficos *Bloxplots*. Para o sistema MyBatis (Figura 6.7(a)) e AJHSQLDB (Figura 6.7(b)) o PAES é melhor que o NSGA-II e SPEA2, já NSGA-II e SPEA2 não apresentam diferença significativa. Para o sistema HealthWatcher (Figura 6.7(c)), os MOEAs NSGA-II e SPEA2 são melhores que o PAES, mas não apresentam diferença entre si.



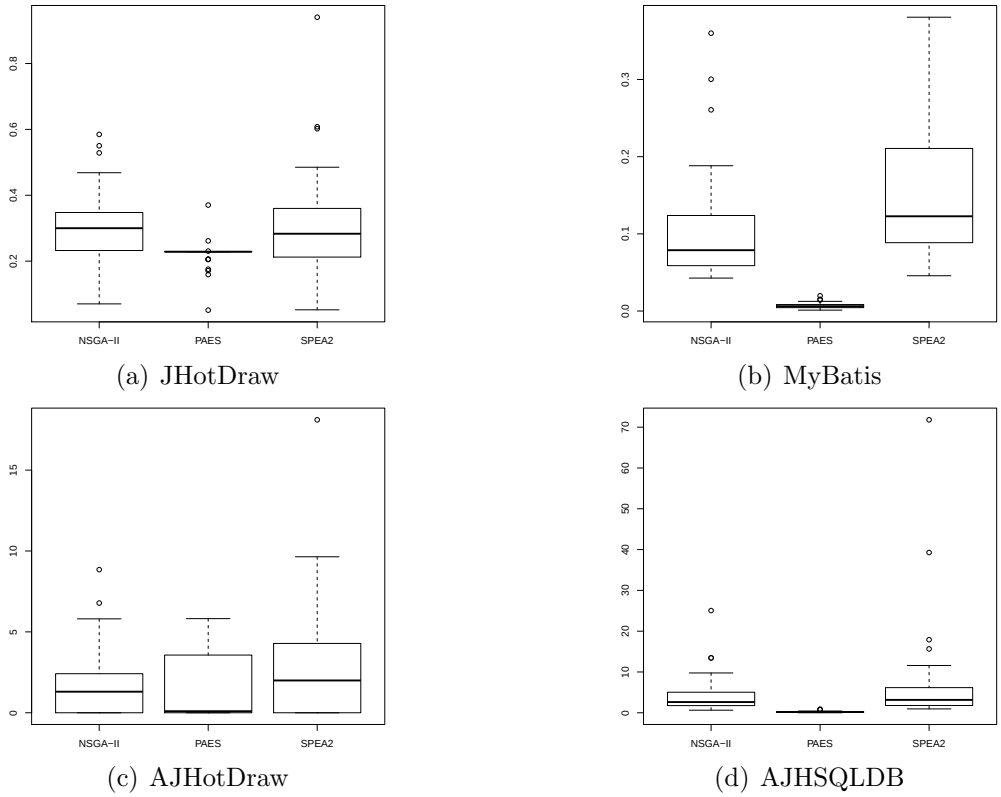
(a) MyBatis (b) AJHSQLDB (c) Health Watcher

Figura 6.7: *Boxplots* para o Indicador GD, Medidas A e O

A Figura 6.8 apresenta os gráficos *Bloxplots* para o indicador IGD. Para o sistema JHotDraw (Figura 6.8(a)) o PAES foi melhor que o NSGA-II, já o NSGA-II e o SPEA2 não têm diferença significativa. Para o sistema AJHotDraw (Figura 6.8(a)) o PAES foi melhor que o SPEA2, já o NSGA-II e o SPEA2 não têm diferença significativa. Para os sitemas MyBatis (Figura 6.8(b)) e AJHSQLDB (Figura 6.8(d)) o PAES foi melhor que o NSGA-II e SPEA2, já o NSGA-II e SPEA2 não têm diferença significativa.

Considerando quatro medidas, para o indicador GD o teste estatístico apontou diferença significativa entre os sistemas: BCEL, JHotDraw, MyBatis, AJHotDraw, AJHSQLDB e Health Watcher. Para o indicador IGD o teste apontou diferença significativa entre os sistemas: JHotDraw, MyBatis, AJHotDraw, AJHSQLDB e Health Watcher.

Diante da diferença apresentada pelo teste de Friedman, para determinar os melhores MOEAs considerando o indicador GD, na Figuras 6.9 são apresentados os gráficos



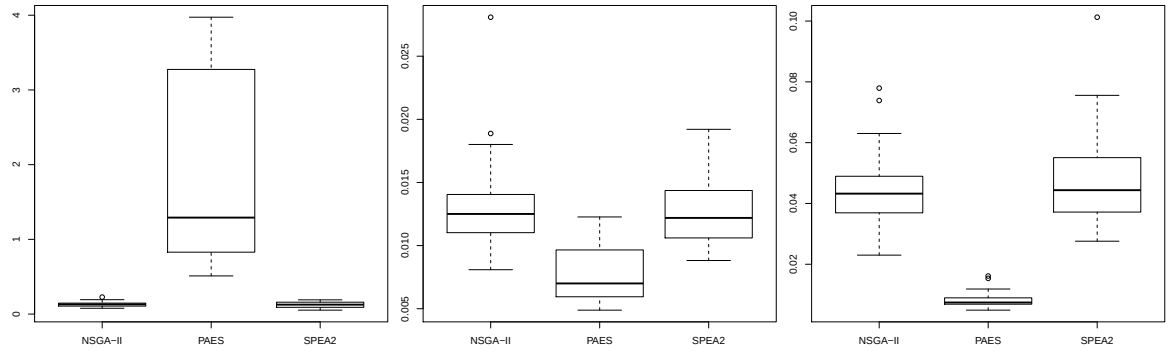
(a) JHotDraw (b) MyBatis

(c) AJHotDraw (d) AJHSQLDB

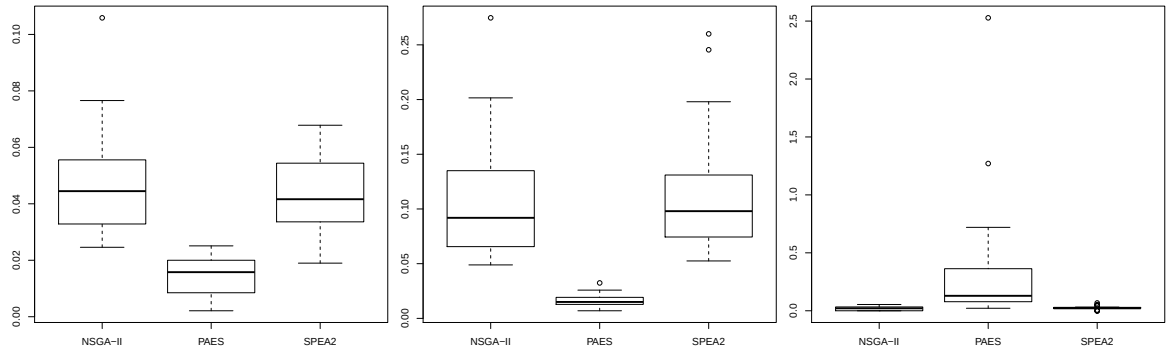
Figura 6.8: *Boxplots* do Indicador IGD, Medidas A e O

Bloxplots. Para os sistemas BCEL (Figura 6.9(a)) e Health Watcher (Figura 6.9(f)), o NSGA-II e SPEA2 são melhores que o PAES, e não apresentam diferença significativa entre si. Para os sistemas JHotDraw (Figura 6.9(b)), MyBatis (Figura 6.9(c)), AJHotDraw (Figura 6.9(d)) e AJHSQLDB (Figura 6.9(e)), o PAES é melhor que o NSGA-II e o SPEA2, e estes últimos não apresentam diferença.

Para o indicador IGD, os gráficos *Boxplots* são apresentados na Figura 6.10. Para os sistemas JHotDraw (Figura 6.10(a)), MyBatis (Figura 6.10(b)), AJHotDraw (Figura 6.10(c)) e AJHSQLDB (Figura 6.10(d)), o PAES é melhor que o NSGA-II e SPEA2, e estes últimos não apresentam diferença. Para o sistema Health Watcher (Figura 6.10(e)), o NSGA-II e SPEA2 são melhores que o PAES, e não apresentam diferença significativa entre si.

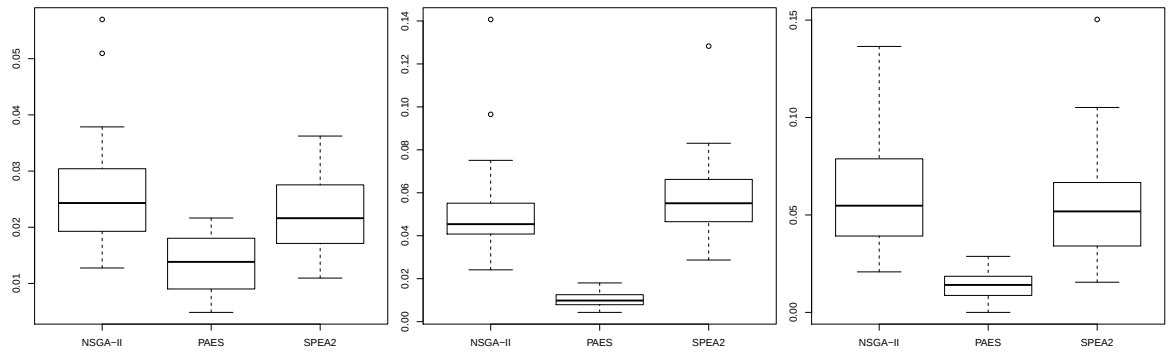


(a) BCEL (b) JHotDraw (c) MyBatis

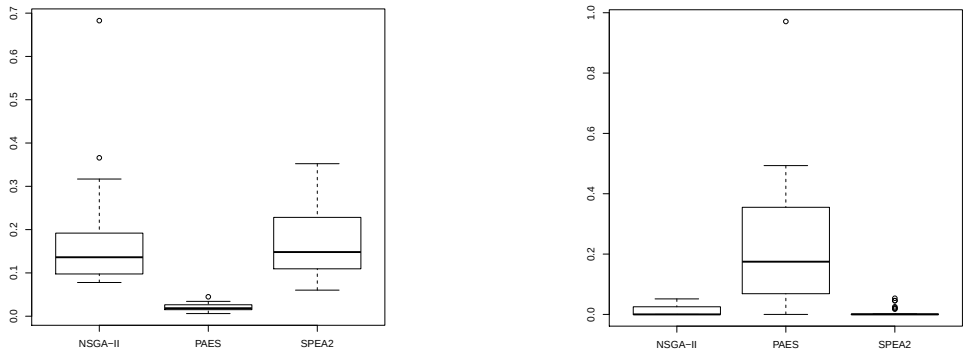


(d) AJHotDraw (e) AJHSQLDB (f) Health Watcher

Figura 6.9: *Boxplots* do Indicador GD, Medidas A, O, R e P



(a) JHotDraw (b) MyBatis (c) AJHotDraw



(d) AJHSQLDB (e) Health Watcher

Figura 6.10: *Boxplots* do Indicador IGD, Medidas A, O, R e P

6.6.3.3 Resultados para o Indicador de qualidade ED

Para o indicador ED a Tabela 6.7 apresenta os valores obtidos. Na segunda coluna desta tabela, são apresentados os custos das soluções ideais, obtidos considerando os menores valores de cada um dos objetivos, independente de estarem na mesma solução, de todas as soluções do conjunto PF_{true} de cada sistema. Nas outras colunas é apresentada a solução mais próxima da solução ideal e o custo de tal solução.

Tabela 6.7: Custo da Solução Ideal e Menores ED Encontradas

Medidas A e O							
Sistema	**Solução Ideal**	**NSGA-II**		**PAES**		**SPEA2**	
		Menor ED	**Custo da Solução**	**Menor ED**	**Custo da Solução**	**Menor ED**	**Custo da Solução**
BCEL	(48,56)	**4,2426**	(51,59)	**4,2426**	(51,59)	**4,2426**	(51,59)
JBoss	(25,17)	**0,0000**	(25,17)	**0,0000**	(25,17)	**0,0000**	(25,17)
JHotDraw	(283,258)	**17,8045**	(297,269)	**17,8045**	(297,269)	21,6333	(301,270)
MyBatis	(256,149)	144,6271	(327,275)	**85,3756**	(276,232)	159,1540	(337,286)
AJHotDraw	(190,100)	8,6023	(197,105)	**4,1231**	(191,104)	10,0000	(198,106)
AJHSQLDB	(3692,729)	579,8552	(4258,855)	**97,1236**	(3720,822)	762,1863	(4404,1001)
Health Watcher	(115,149)	**26,3059**	(119,175)	**26,3059**	(119,175)	**26,3059**	(119,175)
Toll System	(68,41)	**1,0000**	(68,42)	**1,0000**	(68,42)	**1,0000**	(68,42)
Medidas A, O, R e P							
Sistema	**Solução Ideal**	**NSGA-II**		**PAES**		**SPEA2**	
		Menor ED	**Custo da Solução**	**Menor ED**	**Custo da Solução**	**Menor ED**	**Custo da Solução**
BCEL	(40,54, 33,59)	24,5764	(57,59, 50,60)	74,0000	(51,59, 34,132)	**23,4094**	(45,63, 52,68)
JBoss	(25,17, 4,14)	**2,0000**	(25,17, 6,14)	**2,0000**	(25,17, 6,14)	**2,0000**	(25,17, 6,14)
JHotDraw	(283,258, 92,140)	**63,2297**	(301,274, 105,197)	**63,2297**	(301,274, 105,197)	**63,2297**	(301,274, 105,197)
MyBatis	(259,148, 57,145)	203,2855	(1709,204, 81,191)	**147,5263**	(282,235, 78,260)	221,4746	(386,267, 97,276)
AJHotDraw	(190,100, 40,62)	51,6817	(196,105, 43,113)	**49,1325**	(197,106, 45,110)	49,6488	(200,106, 45,110)
AJHSQLDB	(3732,737, 312,393)	526,5302	(4217,879, 415,499)	**167,2692**	(3836,810, 365,488)	403,7809	(4069,879, 403,538)
Health Watcher	(115,149, 49,52)	**39,7869**	(138,166, 67,73)	**39,7869**	(138,166, 67,73)	**39,7869**	(138,166, 67,73)
Toll System	(68,41, 18,16)	**5,4772**	(68,42, 20,21)	**5,4772**	(68,42, 20,21)	**5,4772**	(68,42, 20,21)

Considerando-se duas medidas, para os sistemas BCEL, JBoss, Health Watcher e Toll System todos os MOEAs encontraram uma mesma solução mais próxima da solução ideal. Para o sistema JHotDraw, o NSGA-II e PAES encontram a solução mais próxima. Para os sistemas MyBatis, AJHotDraw e AJHSQLDB o PAES encontrou a solução mais próxima

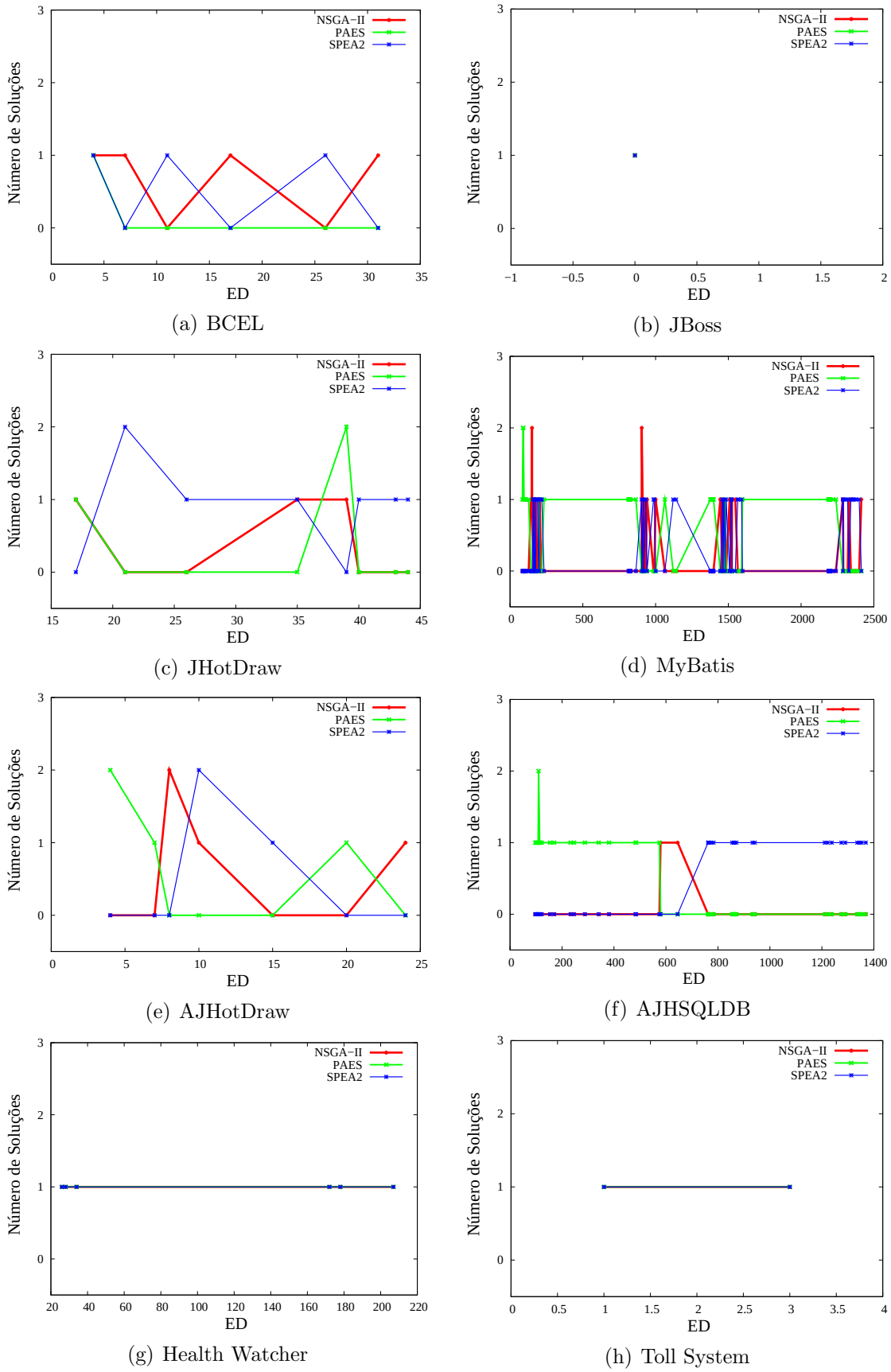
da solução ideal. Portanto, para o indicador ED o PAES foi o melhor MOEA, seguido pelo NSGA-II e por último o SPEA2.

Considerando-se quatro medidas, verifica-se que para os sistemas JBoss, JHotDraw, Health Watcher e Toll System, todos os MOEAs encontraram uma mesma solução com o menor valor de ED. Para o sistema BCEL, o SPEA2 encontrou a solução com o melhor valor de ED. Para os sistemas MyBatis, AJHotDraw e AJHSQLDB, o PAES encontrou a solução com menor valor de ED.

Estas soluções mais próximas dos objetivos mínimos são as que apresentam melhor *tradeoff* em relação as duas medidas utilizadas, sendo boas candidatas a serem adotadas pelo engenheiro de software/testador. Entretanto, a fim de verificar mais a fundo o comportamento dos MOEAs com base no indicador ED, os gráficos da Figura 6.11 e 6.12 apresentam a relação entre a distância das soluções encontradas por cada MOEA e a quantidade de soluções encontradas, para duas e quatro medidas respectivamente. Isto permite uma visão mais precisa sobre qual dos algoritmos encontra uma maior quantidade de soluções com bons *tradeoffs* entre as medidas utilizadas.

Nos gráficos das medidas A e O, para o sistema BCEL (Figura 6.11(a)) o NSGA-II apresenta um pequena vantagem em relação à convergência das soluções para a solução ideal, seguido pelo SPEA2 e por fim o PAES, com apenas uma solução. Para o sistema JBoss (Figura 6.11(b)) todos os MOEAS encontraram uma única e idêntica solução. Para o sistema JHotDraw (Figura 6.11(c)), o SPEA2 encontra a maior quantidade das soluções com os melhores ED. Para os sistemas MyBatis (Figura 6.11(d)) e AJHotDraw (Figura 6.11(e)), o PAES teve a melhor convergência de soluções, seguido pelo NSGA-II e por fim o SPEA2. Para o sistema AJHSQLDB (Figura 6.11(f)) o PAES tem praticamente todas as soluções com menor ED, seguido pelo NSGA-II e por fim o SPEA2. Para os sistemas Health Watcher (Figura 6.11(g)) e Toll System (Figura 6.11(h)) os três MOEAs obtiveram as mesmas soluções, com os mesmos valores de ED.

Nos gráficos das medidas A, O, R e P, para o sistema BCEL (Figura 6.12(a)), o SPEA2 obteve a maior quantidade de soluções próximas a solução ideal, seguido pelo NSGA-II e por último pelo PAES. Para os sistemas JBoss (Figura 6.12(b)), JHotDraw (Fi-



(a) BCEL

(b) JBoss

(c) JHotDraw

(d) MyBatis

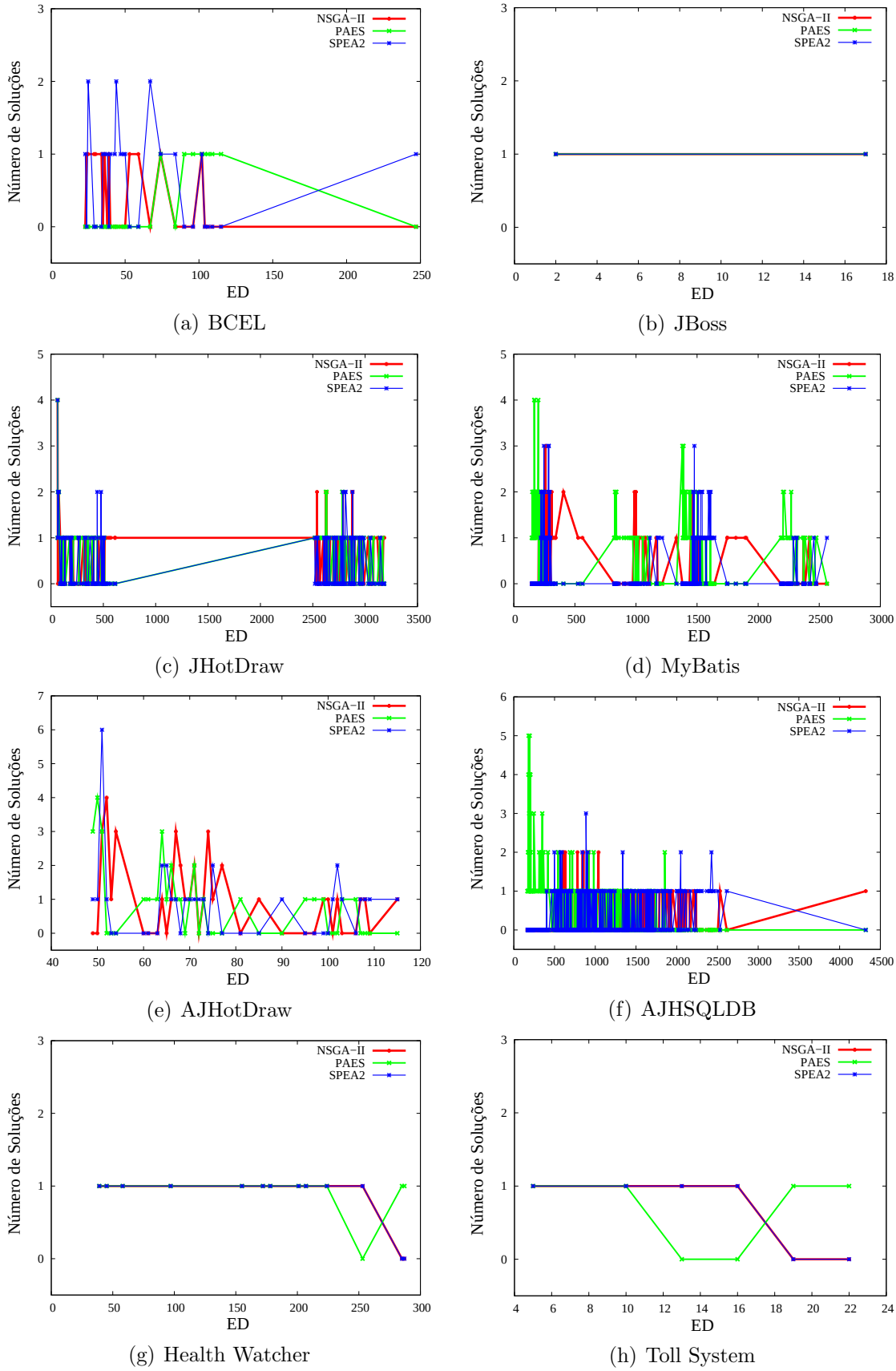
(e) AJHotDraw

(f) AJHSQLDB

(g) Health Watcher

(h) Toll System

Figura 6.11: Número de Soluções X ED, Medidas A e O



(a) BCEL (b) JBoss

(c) JHotDraw (d) MyBatis

(e) AJHotDraw (f) AJHSQLDB

(g) Health Watcher (h) Toll System

Figura 6.12: Número de Soluções X ED, Medidas A, O, R e P

gura 6.12(c)) e Health Watcher (Figura 6.12(g)), todos os MOEAs tiveram o mesmo comportamento, encontrando a mesma quantidade de soluções mais próximas da solução ideal. Para os sistemas MyBatis (Figura 6.12(d)), AJHotDraw (Figura 6.12(e)) e AJHSQLDB (Figura 6.12(f)) o PAES obteve as soluções mais próximas da solução ideal, seguido pelo SPEA2 e por último pelo NSGA-II. Para o sistema TollSystem (Figura 6.12(h)), o NSGA-II e SPEA2 encontraram as soluções mais próximas da solução ideal, seguidos pelo PAES.

6.6.4 Discussão dos Resultados

Para avaliar o comportamento dos algoritmos com base nos resultados obtidos com duas e quatro medidas, a Tabela 6.8 apresenta um resumo dos melhores MOEAs para cada um dos quatro indicadores de qualidade. Para cada sistema é apresentado o MOEA que obteve o melhor resultado do indicador correspondente à coluna. Em caso de mais de um MOEA terem obtido os melhores resultados, todos eles são apontados na célula, portanto a ordem de aparição dos MOEAs em cada célula não esta relacionada a uma precedência de resultados.

Tabela 6.8: Melhores MOEAs por Indicador de Qualidade

Sistema	**Medias A e O**				**Medias A, O, R e P**			
	C	**GD**	**IGD**	**ED**	**C**	**GD**	**IGD**	**ED**
BCEL	NSGA-II PAES SPEA2	NSGA-II PAES SPEA2	NSGA-II PAES SPEA2	NSGA-II	NSGA-II SPEA2	NSGA-II SPEA2	NSGA-II PAES SPEA2	SPEA2
JBoss	NSGA-II PAES SPEA2	NSGA-II PAES SPEA2	NSGA-II PAES SPEA2	NSGA-II PAES SPEA2	NSGA-II PAES SPEA2	NSGA-II PAES SPEA2	NSGA-II PAES SPEA2	NSGA-II PAES SPEA2
JHotDraw	NSGA-II PAES	NSGA-II PAES SPEA2	PAES	SPEA2	NSGA-II PAES SPEA2	PAES	PAES	NSGA-II PAES SPEA2
MyBatis	NSGA-II PAES	PAES	PAES	PAES	NSGA-II PAES	PAES	PAES	PAES
AJHotDraw	NSGA-II PAES	NSGA-II PAES SPEA2	PAES	PAES	NSGA-II SPEA2	PAES	PAES	PAES
AJHSQLDB	NSGA-II PAES	PAES	PAES	PAES	NSGA-II SPEA2	PAES	PAES	PAES
Health Watcher	NSGA-II PAES SPEA2	NSGA-II SPEA2	NSGA-II PAES SPEA2	NSGA-II PAES SPEA2	NSGA-II PAES SPEA2	NSGA-II SPEA2	NSGA-II SPEA2	NSGA-II PAES SPEA2
Toll System	NSGA-II PAES SPEA2	NSGA-II PAES SPEA2	NSGA-II PAES SPEA2	NSGA-II PAES SPEA2	NSGA-II SPEA2	NSGA-II PAES SPEA2	NSGA-II PAES SPEA2	NSGA-II SPEA2

Para o sistema BCEL, o melhor MOEA foi o NSGA-II, seguido com pouca diferença pelos SPEA2 e distante dos dois o PAES. Para o sistema JBoss, todos os MOEAs apresentaram o mesmo comportamento. Para o sistema JHotDraw, o PAES foi o melhor, seguido pelos NSGA-II e SPEA2. Para o sistema MyBatis, o PAES foi o melhor com bastante diferença seguido pelo NSGA-II. Para o sistema AJHotDraw, o PAES foi o melhor bem a frente do NSGA-II, e este seguido pelo SPEA2. Para o sistema AJHSQLDB, o PAES foi o melhor apresentando bastante diferença sobre o NSGA-II. Para os sistemas Health Watcher e Toll System, o NSGA-II e SPEA2 foram os melhores, seguidos pelo PAES, com pouca diferença.

Diante desta análise, pode-se verificar que o PAES é o melhor MOEA, pois obteve os melhores resultados para cinco sistemas: JBoss, JHotDraw, MyBatis, AJHotDraw e AJHSQLDB. Estes cinco sistemas são os que têm maior número de módulos (Tabela 6.1) e maior número de agrupamentos (Tabela 6.2). Portanto em casos que se deve estabelecer ordens de integração e teste para sistemas grandes e com vários agrupamentos, o PAES é a melhor opção dentre os MOEAs analisados.

O NSGA-II é o segundo melhor MOEA, obtendo os melhores resultados para quatro sistemas: BCEL, JBoss, Health Watcher e Toll System. Por fim o SPEA2 obteve os melhores resultados para três sistemas: JBoss, Health Watcher e Toll System. Tanto NSGA-II quanto o SPEA2 apresentam comportamento similares, sendo melhores para sistemas com poucos módulos (Tabela 6.1) e poucos agrupamentos (Tabela 6.2).

6.6.5 Exemplo de Uso das Soluções - Seleção de Uma Ordem

Conforme apresentado no Capítulo 5, existe uma etapa na abordagem MECBA-Clu que refere-se a seleção de uma ordem de integração e teste de módulos dentre as várias ordens resultantes da etapa de otimização multiobjetivo.

Para exemplificar uma regra possível de ser utilizada nesta etapa, na Tabela 6.9 são apresentadas algumas soluções obtidas pelo MOEA PAES para o sistema JHotDraw, considerando quatro medidas de acoplamento. O sistema JHotDraw é o quarto sistema com maior número de módulos (197 módulos) e o terceiro sistema com maior número de

agrupamentos (13 agrupamentos). Para este sistema o PAES encontrou cento e quarenta e três (143) soluções. Portanto, é necessário que o testador escolha qual dessas ordens vai utilizar.

Na Tabela 6.9 a primeira coluna apresenta o custo de cada solução (A,O,R,P) e a segunda coluna apresenta a ordem dos módulos, em seus respectivos grupos.

Tabela 6.9: Algumas Soluções do PAES para o Sistema JHotDraw

Custo da Solução	**Ordem de Módulos**
(**283**,292,102,206)	{87, 9, 196, 187, 67, 185}, {68, 86, 173, 105, 118, 48, 24, 170, 93, 99, 30, 104, 102, 91, 116, 23, 82, 190, 186, 46, 134, 121, 49, 89, 164, 188, 25, 115, 117, 189, 100, 50, 157, 69, 155, 26, 38, 191}, {84, 85, 58, 0, 79, 66, 98, 7, 111}, {96, 180, 123, 114, 101, 53, 12, 75, 36, 140, 107, 144, 145, 47, 62, 11, 34, 77, 97, 146, 57, 3, 73, 103, 152, 28, 158, 2, 179, 83, 122, 16, 129, 182, 27, 45, 176, 159, 5, 31, 52, 156, 165, 166, 32, 4, 150, 192, 54, 110, 137, 1, 8, 151, 113, 65, 95, 135, 132, 130}, {128, 181, 147, 125, 169, 124, 61}, {138, 108, 15, 33, 29, 154, 153}, {56, 6, 139, 42, 39, 41, 44, 40, 172, 43}, {184, 160, 76, 119, 18, 70, 178, 55, 35, 71, 64, 59, 175, 74, 126, 167, 72, 177, 174}, {136, 94, 60, 171}, {133, 51, 14, 109, 21, 148}, {10, 90, 195, 78, 88, 183, 81}, {17, 63, 19, 20, 22, 161, 37, 106, 163, 112, 168, 142, 127, 149, 92, 143, 193, 162, 80}, {131, 13, 120, 194, 141}
(322,**258**,103,192)	{87, 9, 196, 187, 67, 185}, {84, 85, 0, 58, 66, 98, 7, 111, 79}, {138, 108, 15, 33, 29, 154, 153}, {68, 86, 173, 105, 118, 48, 24, 170, 93, 99, 30, 104, 102, 91, 116, 23, 82, 190, 186, 46, 134, 121, 49, 89, 164, 188, 25, 115, 117, 189, 100, 50, 157, 69, 155, 26, 38, 191}, {96, 180, 123, 114, 101, 53, 12, 75, 36, 140, 107, 144, 16, 145, 47, 62, 11, 34, 77, 97, 146, 57, 3, 73, 103, 152, 28, 158, 2, 132, 110, 179, 83, 122, 32, 129, 182, 27, 45, 176, 159, 5, 31, 52, 156, 165, 166, 135, 4, 150, 192, 54, 137, 1, 8, 151, 113, 130, 65, 95}, {56, 6, 139, 42, 39, 41, 44, 40, 172, 43}, {17, 63, 19, 20, 22, 161, 37, 106, 163, 112, 168, 142, 127, 149, 92, 143, 193, 162, 80}, {184, 160, 76, 119, 18, 70, 178, 55, 35, 71, 64, 59, 175, 74, 126, 167, 72, 177, 174}, {136, 94, 60, 171}, {133, 51, 14, 109, 21, 148}, {128, 181, 147, 125, 169, 124, 61}, {131, 13, 120, 194, 141}, {10, 90, 195, 78, 88, 183, 81}
(2918,326,**92**,201)	{96, 180, 123, 114, 101, 53, 12, 75, 36, 140, 107, 144, 145, 47, 62, 11, 34, 77, 97, 146, 57, 3, 73, 103, 152, 28, 158, 2, 179, 83, 122, 16, 129, 182, 27, 45, 176, 159, 5, 31, 52, 156, 165, 166, 32, 4, 150, 192, 54, 110, 137, 1, 8, 151, 113, 65, 95, 135, 132, 130}, {138, 108, 15, 33, 29, 154, 153}, {87, 9, 196, 187, 67, 185}, {84, 85, 58, 0, 79, 66, 98, 7, 111}, {128, 181, 147, 125, 169, 124, 61}, {68, 86, 173, 105, 118, 48, 24, 170, 93, 99, 30, 104, 102, 91, 116, 23, 82, 190, 186, 46, 134, 121, 49, 89, 164, 188, 25, 115, 117, 189, 100, 50, 157, 69, 155, 26, 38, 191}, {56, 6, 139, 42, 39, 41, 44, 40, 172, 43}, {184, 160, 76, 119, 18, 70, 178, 55, 35, 71, 64, 59, 175, 74, 126, 167, 72, 177, 174}, {136, 94, 60, 171}, {133, 51, 14, 109, 21, 148}, {10, 90, 195, 78, 88, 183, 81}, {17, 63, 19, 20, 22, 161, 37, 106, 163, 143, 168, 142, 127, 149, 92, 112, 193, 162, 80}, {131, 13, 120, 194, 141}
(3423,313,103,**140**)	{138, 108, 15, 29, 154, 33, 153}, {56, 6, 42, 39, 139, 43, 40, 172, 44, 41}, {96, 103, 114, 123, 180, 101, 165, 12, 97, 47, 34, 36, 146, 140, 107, 77, 32, 45, 53, 75, 57, 145, 83, 11, 16, 156, 3, 95, 73, 152, 158, 192, 4, 28, 113, 144, 166, 110, 137, 27, 5, 159, 52, 62, 54, 2, 182, 179, 122, 31, 129, 150, 135, 132, 130, 1, 8, 65, 151, 176}, {187, 87, 9, 67, 196, 185}, {84, 85, 0, 58, 66, 7, 98, 79, 111}, {128, 181, 124, 147, 169, 61, 125}, {17, 19, 37, 168, 127, 161, 20, 163, 106, 22, 63, 112, 142, 143, 149, 193, 92, 80, 162}, {10, 88, 195, 78, 183, 81, 90}, {136, 171, 94, 60}, {133, 51, 14, 21, 148, 109}, {134, 157, 24, 25, 102, 191, 89, 26, 115, 173, 46, 104, 49, 30, 91, 100, 170, 82, 116, 164, 105, 121, 68, 93, 38, 99, 190, 117, 50, 48, 69, 86, 189, 155, 118, 186, 188, 23}, {131, 141, 13, 194, 120}, {184, 119, 76, 160, 18, 35, 71, 178, 70, 55, 64, 59, 74, 175, 167, 177, 174, 72, 126}
(**301**,**274**,**105**,**197**)	{138, 29, 108, 15, 33, 154, 153}, {187, 87, 9, 196, 185, 67}, {24, 116, 93, 82, 25, 190, 49, 91, 30, 99, 170, 104, 105, 26, 115, 173, 191, 164, 121, 86, 50, 189, 46, 69, 186, 134, 38, 48, 155, 102, 100, 188, 117, 89, 23, 118, 157, 68}, {101, 96, 114, 12, 53, 180, 123, 62, 182, 16, 156, 140, 107, 103, 145, 45, 75, 144, 34, 36, 146, 11, 97, 3, 152, 158, 95, 73, 2, 122, 179, 176, 47, 28, 27, 31, 5, 165, 54, 77, 4, 57, 159, 113, 150, 52, 129, 166, 192, 83, 110, 32, 137, 135, 1, 8, 151, 65, 132, 130}, {128, 147, 124, 125, 169, 181, 61}, {131, 13, 141, 120, 194}, {58, 84, 66, 0, 7, 98, 111, 85, 79}, {10, 81, 78, 88, 195, 90, 183}, {133, 51, 14, 148, 109, 21}, {56, 6, 139, 42, 40, 39, 44, 43, 172, 41}, {136, 60, 94, 171}, {184, 76, 160, 119, 18, 64, 55, 74, 59, 35, 70, 126, 178, 175, 177, 71, 174, 167, 72}, {17, 161, 163, 20, 63, 142, 168, 19, 149, 106, 112, 193, 22, 143, 37, 127, 92, 162, 80}

Para demonstrar como cada solução deve ser analisada, tem-se como base a primeira solução da tabela, com o custo de (283,292,102,206). A ordem apresentada na segunda coluna, {87, 9, 196, ...}, ..., {..., 120, 194, 141}, indica a sequência em que os módulos devem ser desenvolvidos, integrados e testados. Utilizando esta ordem, para realização

do teste de integração de todo o sistema será necessário construir *stubs* para simular 283 atributos, 292 operações, que podem ser métodos de classes ou adendos de aspectos, 120 tipos distintos de retornos e 206 tipos distintos de parâmetros.

Para escolher dentre as cinco soluções apresentadas na Tabela 6.9, utilizou-se a regra que consiste em selecionar as soluções que têm o custo menor para determinada medida. A medida com menor custo é destacada em negrito na tabela, portanto, a primeira solução tem o menor custo para a medida A, a segunda solução tem o menor custo para a medida O, a terceira solução tem o menor custo para a medida R e a quarta solução tem o menor custo para a medida P. A quinta solução apresenta o melhor balanceamento de custo dentre as quatro medidas e foi selecionada com base no indicador ED (Tabela 6.7).

Como somente uma ordem de módulos é utilizada para determinar a sequência de integração e teste de módulos, para decidir qual solução utilizar, o engenheiro de software/testador pode optar por qual medida deseja priorizar. Por exemplo, caso o sistema em desenvolvimento apresente atributos complexos de serem construídos, então a primeira solução deve ser utilizada; ou caso o sistema apresente parâmetros das operações difíceis de serem simulados, a terceira solução deve ser utilizada. Porém, caso o engenheiro de software/testador optar por priorizar todos as medidas ao mesmo tempo, a quinta solução é a melhor opção, pois está mais próxima dos custos mínimos.

Esta diversidade de soluções com diferentes *tradeoffs* entre as medidas utilizadas é uma das grandes vantagens da utilização da otimização multiobjetivo, flexibilizando a seleção de uma ordem de módulos que atenda as necessidades do engenheiro de software/testador.

6.7 Considerações Finais

Para avaliar a utilização da abordagem MECBA-Clu, este capítulo apresentou o experimento conduzido com três MOEAs aplicados a oito sistemas reais.

Através da análise dos resultados, comparando o comportamento da abordagem MECBA em relação a abordagem MECBA-Clu, é possível verificar que considerar agrupamentos de módulos durante a resolução dos problemas CITO e CAITO eleva a complexidade para se encontrar soluções, limitando o espaço de busca em determinadas regiões.

Em relação ao comportamento dos MOEAs na abordagem MECBA-Clu, todos encontraram soluções aceitáveis, e com bom *tradeoff* entre as medidas de acoplamento. Através dos indicadores de qualidade, considerando tanto duas quanto quatro medidas, é possível observar que o PAES obteve melhores resultados, seguido pelo NSGA-II, e SPEA2.

Pode ser observado também que o conjunto de soluções encontradas durante a otimização multiobjetivo permite uma decisão flexível sobre qual ordem utilizar para o desenvolvimento dos módulos e consequentemente a integração e teste.

CAPÍTULO 7

CONCLUSÃO

Dentre os trabalhos encontrados na literatura que tratam dos problemas CITO e CAITO, nenhum deles considera os agrupamentos de vários módulos durante o estabelecimento de uma ordem para integração e teste. Entretanto, o agrupamento de módulos é uma característica que está presente no desenvolvimento de software, seja por questões contratuais, pelo desenvolvimento distribuído ou pela organização do código fonte.

Considerar os agrupamentos de módulos refere-se a manter os módulos de um mesmo grupo em sequência dentro da ordem de teste estabelecida. Portanto, este trabalho descreve a proposta de uma estratégia para estabelecer ordens de integração e teste de módulos com base em agrupamentos.

Dentre os vários trabalhos que propuseram diferentes maneiras de estabelecer sequencias para integração e teste de módulos, a abordagem MECBA se apresenta como a mais promissora com os melhores resultados. Diante disso, foi implementada a abordagem MECBA-Clu, que é uma extensão da abordagem MECBA mas que considera os agrupamentos de módulos.

A abordagem MECBA permite o estabelecimento de sequências de módulos para sistemas nos contextos de OO e de OA. Trata o problema como multiobjetivo permitindo considerar várias medidas de acoplamento para representar o custo da ordem estabelecida. Este tratamento do problema como multiobjetivo, é efetuado por MOEAs, que encontram um conjunto de boas soluções, oferecendo um melhor *tradeoff* para o engenheiro de software/testador. Como diferencial a abordagem MECBA-Clu introduz o tratamento dos agrupamentos como uma restrição a ser considerada pelos MOEAs. Além dos modelos de dependência e custo, entradas para os MOEAs, na abordagem MECBA-Clu os agrupamentos também são informados, para serem considerados no processo evolutivo dos algoritmos.

Para avaliar a abordagem MECBA-Clu foi conduzido um experimento, utilizando oito sistemas reais (quatro sistemas OO e quatro sistema OA), três MOEAs e duas diferentes configurações de medidas de acoplamento (A e O) e (A, O, R e P). O experimento teve por objetivo (i) verificar a complexidade para se encontrar as soluções quando os agrupamentos são considerados e (ii) verificar o comportamento dos MOEAs quando otimizam um problema com maior número de restrições.

Os resultados da abordagem MECBA-Clu foram comparados com os resultados da abordagem MECBA, a fim de verificar a complexidade introduzida no espaço de busca. Observou-se que as soluções da MECBA são distribuídas uniformemente no espaço de busca, enquanto as soluções da abordagem MECBA-Clu ficam concentradas em determinadas regiões. Portanto, para considerar os agrupamentos de módulos, o espaço de busca fica limitado a determinadas regiões, tornado a busca mais complexa.

Para comparar o comportamento entre os MOEAs, quando tratam de mais restrições do que comumente utilizado pela abordagem MECBA, foram utilizados quatro indicadores de qualidade. Com base nos indicadores de qualidade e analisando-se os resultados de quando utilizadas duas e quatro medidas de acoplamento, é possível afirmar que o PAES tem melhor desempenho para estabelecer ordens de teste considerando agrupamentos, seguido pelo NSGA-II e por fim o SPEA2. Entretanto, apesar de apresentarem diferentes comportamentos, todos encontraram soluções válidas e aceitáveis, oferecendo um bom *tradeoff* para o engenheiro de software/testador.

A partir do exposto, pode-se resumir a contribuição do presente trabalho em três tipos:

- **Contribuição Teórica**: através da proposta de uma nova estratégia que considera os agrupamentos de módulos durante o estabelecimento de ordens para a integração e teste de tais módulos. Esta estratégia pode ser utilizada como base para novos estudos em teste de software;

- **Contribuição Empírica**: as análises e comparações entre as diferentes abordagens e algoritmos permitem a pesquisadores e profissionais da área de teste de software terem um referencial base para guiar suas atividades de teste quando da necessidade de considerarem agrupamentos de módulos;

- **Contribuição Prática**: através da implementação da abordagem MECBA-Clu, esta pode ser utilizada como ferramenta experimental para o estabelecimento de ordens para a integração e teste de módulos, auxiliando na redução do custo relativo a complexidade dos *stubs* que necessitam ser construídos, durante a atividade de teste de software.

7.1 Trabalhos Futuros

Com base neste trabalho, novos trabalhos podem ser executados a fim de consolidar as conclusões aqui obtidas ou revelar diferentes comportamentos não observados. Dentre os possíveis trabalhos, destacam-se:

- Conduzir experimentos utilizando diferentes sistemas com maior número de módulos, ou com mais dependências entre os módulos;
- Implementar a estratégia *Incremental+* de Ré et al. [51] no contexto de AO juntamente com a abordagem MECBA-Clu e comparar com a estratégia *Combined* utilizada neste trabalho;
- Analisar o quão restrito fica o espaço de busca quando novas restrições são acrescentadas, além das utilizadas até o momento;
- Neste trabalho os agrupamentos iniciais foram estabelecidos pelo engenheiro de software/testador, mas podem ser explorados algoritmos de "*clustering*" para estabelecer os agrupamentos de módulos similares;
- A abordagem deverá ser avaliada em outros contextos, tais como o de componentes e serviços;
- Experimentos com sistemas reais poderão ser conduzidos, considerando por exemplo restrições de um contexto de desenvolvimento distribuído de software;
- Conduzir experimentos com diferentes MOEAs, ou com diferentes meta-heurísticas multiobjetivos, como por exemplo: Otimização Por Colônia de Formigas (*Pareto Ant*

Colony Optimization - PACO), Colônia Artificial de Abelhas (*Multi-Objective Artificial Bee Colony - MOABC*), Busca Tabu (*Multi-Objective Tabu Search - MTABU*), Sistemas Imunológicos Artificais (*Multi-Objective Artificial Immune System*), Otimização Por Núvem de Partículas (*Multi-Objective Particle Swarm Optimization - MOPSO*), etc.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] A. Abdurazik e J. Offutt. Coupling-based class integration and test order. *International Workshop on Automation of Software Test*, Shanghai, China, 2006. ACM.

[2] R. T. Alexander e J. M. Bieman. Towards the systematic testing of aspect-oriented programs. Relatório técnico CS-04-105, Colorado State University, Fort Collins, Colorado, 2004.

[3] W. K. G. Assunção, T. E. Colanzi, A. T. R. Pozo, e S. R. Vergilio. Establishing integration test orders of classes with several coupling measures. *13th Genetic and Evolutionary Computation Conference (GECCO'2011)*, páginas 1867–1874, 2011.

[4] W. K. G. Assunção, T. E. Colanzi, A. T. R. Pozo, e S. R. Vergilio. Uma avaliação do uso de diferentes algoritmos evolutivos multiobjetivos para integração de classes e aspectos. *II Workshop on Search Based Software Engineering (WESB'2011)*, 2011.

[5] W. K. G. Assunção, T. E. Colanzi, S. R. Vergilio, e A. T. R. Pozo. Estabelecendo sequências de teste de integração de classes: Um estudo comparativo da aplicação de três algoritmos evolutivos multiobjetivos. *Workshop de Tolerância a Falhas (WTF'2011)*, 2011.

[6] W. K. G. Assunção, T. E. Colanzi, S. R. Vergilio, e A. T. R. Pozo. Generating integration test orders for aspect oriented software with multi-objective algorithms. *Revista de Informática Teórica e Aplicada (RITA)*, 2012. Submetido para publicação.

[7] R. V. Binder. *Testing object-oriented systems: models, patterns, and tools*. Addison-Wesley Longman Publishing Co., Boston, MA, USA, 1999.

[8] L. C. Briand, J. Feng, e Y. Labiche. *Experimenting with Genetic Algorithms and Coupling Measures to Devise Optimal Integration Test Orders*. Carleton University, Technical Report SCE-02-03, 2002.

[9] L. C. Briand, J. Feng, e Y. Labiche. Using genetic algorithms and coupling measures to devise optimal integration test orders. *14th International Conference on Software Engineering and Knowledge Engineering*, 2002.

[10] L. C. Briand e Y. Labiche. An investigation of graph-based class integration test order strategies. *IEEE Transactions on Software Engineering*, 29(7):594–607, 2003.

[11] R. V. Cabral, A. T. R. Pozo, e S. R. Vergilio. A Pareto ant colony algorithm applied to the class integration and test order problem. *22nd IFIP International Conference on Testing Software and Systems (ICTSS'2010)*. Springer, 2010.

[12] David N. Card, Gerald T. Page, e Frank E. McGarry. Criteria for software modularization. *8th international conference on Software engineering (ICSE'1985)*, páginas 372–377, Los Alamitos, CA, USA, 1985. IEEE Computer Society Press.

[13] M. Ceccato, P. Tonella, e F. Ricca. Is AOP code easier or harder to test than OOP code. *First Workshop on Testing Aspect-Oriented Program (WTAOP'2005)*, Chicago, Illinois, 2005.

[14] H. Y. Chen. An approach for object-oriented cluster-level tests based on UML. *IEEE International Conference on Systems, Man and Cybernetics*, volume 2, páginas 1064–1068, Los Alamitos, California, USA, 2003. IEEE Computer Society Press.

[15] J. L. Cochrane e M. Zeleny. *Multiple Criteria Decision Making*. University of South Carolina Press, Columbia, 1973.

[16] C. A. C. Coello, G.L. Lamont, e D.A. van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Genetic and Evolutionary Computation. Springer, Berlin, Heidelberg, 2nd edition, 2007.

[17] T. E. Colanzi, W. K. G. Assunção, A. T. R. Pozo, e S. R. Vergilio. Integration testing of classes and aspects with a multi-evolutioanry and coupling-based approach. *3th International Symposium on Search Based Software Engineering (SSBSE'2011)*, páginas 188–203. Springer Verlag, 2011.

[18] T. E. Colanzi, W. K. G. Assunção, S. R. Vergilio, e A. T. R. Pozo. Generating integration test orders for aspect oriented software with multi-objective algorithms. *Proceedings of the Latin-American Workshop on Aspect Oriented Software (LA-WASP'2011)*, 2011.

[19] T. E. Colanzi, S. R. Vergilio, W. K. G. Assunção, e A. T. R. Pozo. Search based software engineering: Review and analysis of the field in brazil. *Journal of Systems and Software - Special Issue on Software Engineering in Brazil: Retrospective and Prospective Views*, 2012. Submetido para publicação.

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

[21] K. Deb, L. Thiele, M. Laumanns, e E. Zitzler. *Evolutionary Multiobjective Optimization: Theoretical Advances and Applications*. Springer, 2005.

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

[23] J. J. Durillo, A. J. Nebro, e E. Alba. The JMetal Framework for Multi-Objective Optimization: Design and Architecture. *Congress on Evolutionary Computation (CEC'2010)*, páginas 4138–4325, Barcelona, Spain, 2010.

[24] R. Galvan, A. T. R. Pozo, e S. R. Vergilio. Establishing Integration Test Orders for Aspect-Oriented Programs with an Evolutionary Strategy. *Latinamerican Workshop on Aspect Oriented Software*, 2010.

[25] Emden R. Gansner e Stephen C. North. An open graph visualization system and its applications to software engineering. *Software: Practice and Experience*, 30(11):1203–1233, 2000.

[26] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[27] S. Gárcia, D. Molina, M. Lozano, e F. Herrera. A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 Special Session on Real Parameter Optimization. *Journal of Heuristics*, 15(6):617–644, 2009.

[28] M. Harman, S. A. Mansouri, e Y. Zhang. Search Based Software Engineering: A Comprehensive Analysis and Review of Trends Techniques and Applications. Relatório Técnico TR-09-03, 2009.

[29] M. J. Harrold e G. Rothermel. Performing data flow testing on classes. *2nd ACM Symposium on Foundations of Software Engineering (SIGSOFT'1994)*, páginas 154–163, New York, NY, USA, 1994. ACM.

[30] J. Jaroenpiboonkit e T. Suwannasart. Finding a test order using object-oriented slicing technique. *14th Asia-Pacific Software Engineering Conference*, páginas 49 56, Washington, DC, USA, 2007.

[31] J. Jaroenpiboonkit e T. Suwannasart. Class ordering tool - a tool for class ordering in integration testing. *International Conference on Advanced Computer Theory and Engineering*, páginas 724 – 728, 2008.

[32] J. Knowles, L. Thiele, e E. Zitzler. A tutorial on the performance assessment of stochastic multiobjective optimizers. Relatório técnico, Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Switzerland, 2006. Revised version.

[33] J. D. Knowles e D. W. Corne. Approximating the nondominated front using the pareto archived evolution strategy. *Evolutionary Computation*, 8:149–172, 2000.

[34] N. A. Kraft, E. L. Lloyd, B. A. Malloy, e P. J. Clarke. The implementation of an extensible system for comparison and visualization of class ordering methodologies. *Journal of Systems and Software*, 79:1092 – 1109, 2006.

[35] D. Kung, J. Gao, P. Hsia, Y. Toyoshima, e C. Chen. A test strategy for object-oriented programs. *19th Computer Software and Applications Conference*, 1995.

[36] D. C. Kung, J. Gao, P. Hsia, J. Lin, e Y. Toyoshima. Class firewall, test order and regression testing of object-oriented programs. *Journal of Object-Oriented Program*, 8(2):51–65, 1995.

[37] O. A. L. Lemos. Teste de programas orientados a aspectos: uma abordagem estrutural para AspectJ. Dissertação de Mestrado, Instituto de Ciências Matemáticas e de Computação – Universidade de São Paulo (USP), São Carlos, SP, 2005.

[38] J. C. Maldonado e S. C. P. F. Fabbri. *Qualidade de Software - Teoria e Prática*, páginas 73–84. Prentice Hall, São Paulo, 2001.

[39] C. Mao e Y. Lu. AICTO: An improved algorithm for planning inter-class test order. *Proceedings of the The Fifth International Conference on Computer and Information Technology*, CIT '05, páginas 927–931, Washington, DC, USA, 2005. IEEE Computer Society.

[40] R. McDaniel e J. D. McGregor. Testing the polymorphic interactions between classes. Relatório técnico TR94-103, Clemson University, 1994.

[41] H. Melton e E. Tempero. An empirical study of cycles among classes in Java. *Empirical Software Engineering*, 12:389–415, 2007.

[42] OPAC : Parallel Cooperation Optimization. Graphical user interface multi-objective optimization. Relatório técnico, 2009.

[43] V. Pareto. *Manuel D'Economie Politique*. Ams Press, Paris, 1927.

[44] D. E. Perry e G. E. Kaiser. Adequate testing and object-oriented programming. *J. Object Oriented Program.*, 2:13–19, 1990.

[45] R. S. Pressman. *Software Engineering : A Practitioner's Approach*. McGraw Hill, New York, NY, 2001.

[46] R. Prikladnicki, J.L.N. Audy, D. Damian, e T.C. de Oliveira. Distributed software development: Practices and challenges in different business strategies of offshoring and

onshoring. *Second IEEE International Conference on Global Software Engineering (ICGSE'2007)*, páginas 262 –274, 2007.

[47] R-Project. The R project for statistical computing. <http://www.r-project.org/>, 2012.

[48] I. Radziukyniene e A. Zilinskas. Evolutionary Methods for Multi-Objective Portfolio Optimization. *Proceedings of the World Congress on Engineering 2008 Vol II*, 2008.

[49] R. Ré. *Uma contribuição para a minimização do número de stubs no teste de integração de programas orientados a aspectos*. Tese de Doutorado, Instituto de Ciências Matemáticas e de Computação, ICMC/USP, São Carlos, SP, 2009.

[50] R. Ré, O. A. L. Lemos, e P. C. Masiero. Minimizing stub creation during integration test of aspect-oriented programs. *3rd Workshop on Testing Aspect-Oriented Programs*, páginas 1–6, Vancouver, British Columbia, Canada, 2007.

[51] R. Ré e P. C. Masiero. Integration testing of aspect-oriented programs: a characterization study to evaluate how to minimize the number of stubs. *Brazilian Symposium on Software Engineering*, páginas 411–426, 2007.

[52] I. Sommerville. *Software Engineering (International Computer Science Series)*. Pearson Addison Wesley, 2004.

[53] K.-C. Tai e F. J. Daniels. Test order for inter-class integration testing of object-oriented software. *21st International Computer Software and Applications Conference*, páginas 602–607. IEEE Computer Society, 1997.

[54] R. Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[55] Y. L. Traon, T. Jéron, J.-M. Jézéquel, e P. Morel. Efficient object-oriented integration and regression testing. *IEEE Transactions on Reliability*, páginas 12–25, 2000.

[56] D. A. Van Veldhuizen e G. B. Lamont. Evolutionary Computation and Convergence to a Pareto Front. John R. Koza, editor, *Late Breaking Papers at the Genetic Programming 1998 Conference*, University of Wisconsin, Madison, Wisconsin, USA, 1998. Stanford University Bookstore.

[57] D. A. van Veldhuizen e G. B. Lamont. Multiobjective evolutionary algorithm test suites. *Proceedings of the 1999 ACM symposium on Applied computing*, SAC 99, páginas 351–357, New York, NY, USA, 1999. ACM.

[58] S. R. Vergilio, T. E. Colanzi, A. T. R. Pozo, e W. K. G. Assunção. Search based software engineering: A review from the Brazilian symposium on software engineering. *Brazilian Symposyum on Software Engineering (SBES'2011) - Track SBES is 25*, páginas 50–55, 2011.

[59] S. R. Vergilio, A. T. R. Pozo, J. Árias, R. V. da Cabral, e T. Nobre. Multi-objective optimization algorithms applied to the class integration and test order problem. *International Journal on Software Tools for Technology Transfer (STTT)*, páginas 1–15. Publicado online.

[60] A. M. R. Vincenzi. *Orientação a objeto: Definição e análise de recursos de teste e validação.* Tese de Doutorado, Instituto de Ciências Matemáticas e de Computação, ICMC/USP, São Carlos, SP, 2004.

[61] S. Yang, Y. Ong, e Y. Jin. *Evolutionary Computation in Dynamic and Uncertain Environments (Studies in Computational Intelligence).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[62] Edward Yourdon e Larry L. Constantine. *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1979.

[63] E. Zitzler, M. Laumanns, e L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. Relatório Técnico 103, Gloriastrasse 35, CH-8092 Zurich, Switzerland, 2001.

[64] E. Zitzler e L. Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE Transaction Evolutionary Computation*, 3(4):257–271, 1999.

[65] Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M. Fonseca, e Viviane Grunert da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on Evolutionary Computation*, 7:117–132, 2003.