Reduzindo o Custo do Teste de Integração com Algoritmos Evolutivos Multiobjetivos e Diferentes Medidas de Acoplamento

Wesley Klewerton Guez Assunção[1], **Thelma Elita Colanzi**[1 2], **Aurora Trinidad Ramirez Pozo**[1], **Silvia Regina Vergilio**[1]

[1]Departamento de Informática, Programa de Pós-Graduação em Informática, Universidade Federal do Paraná (UFPR)– Curitiba, PR – Brasil

[2]Departamento de Informática
Universidade Estadual de Maringá(UEM) – Maringá, PR – Brasil

{wesleyk, thelmae, aurora, silvia}@inf.ufpr.br

Resumo. *Para realizar o teste de software orientado a objetos é necessário determinar uma ordem para integrar e testar as classes que permita minimizar o número de stubs necessários para simular o comportamento de classes ainda não disponíveis durante a integração. O uso de algoritmos multiobjetivos permite considerar vários fatores e medidas que afetam a criação de stubs, e que por serem geralmente conflitantes impedem a obtenção de uma única solução. Em estudos realizados comparando diferentes algoritmos, o NSGA-II, um algoritmo evolutivo multiobjetivo (MOEA), tem obtido os melhores resultados. No entanto, existem outros MOEAs que não foram aplicados neste contexto e que podem melhorar o desempenho do NSGA-II. Além disso, os trabalhos existentes costumam utilizar somente duas medidas de acoplamento: número de métodos e de atributos. Considerando estas limitações, este artigo explora além do NSGA-II, o MOEA SPEA2 para resolver o problema em sistemas reais, considerando quatro medidas de acoplamento. Duas outras medidas são introduzidas: número de diferentes tipos de retorno e de diferentes tipos de parâmetros, em adição às duas medidas tradicionalmente utilizadas. Os resultados permitem a comparação entre os algoritmos, e mostram que ambos são eficientes para resolver o problema, já que encontram soluções de custo de teste mínimo para alguns sistemas, mesmo considerando um número maior de objetivos.*

1. Introdução

O teste de software é uma atividade considerada essencial para garantia de qualidade. Ele geralmente é realizado em etapas. A etapa de integração tem como objetivo testar a interação entre módulos, que são geralmente as classes de um software Orientado a Objetos(OO). Nesta etapa, é comum que algumas classes necessitem de recursos de outras classes que ainda estão em desenvolvimento, o que leva a necessidade de construção de *stubs*, aumentando a complexidade e o custo do teste. *Stubs* são pseudoimplementações que simulam recursos necessários, mas não disponíveis, que são indispensáveis para a execução da classe. Para reduzir o custo associado à construção dos *stubs* é necessário determinar uma ordem de desenvolvimento e teste das classes que minimize o número de *stubs* a serem gerados. O problema de determinar esta ordem ideal é conhecido na

literatura como problema de Ordenação e Teste de Integração de Classes (CITO - *Class and Integration Test Order*) [Abdurazik and Offutt 2006].

Este problema tem sido tema de vários trabalhos na literatura. Geralmente, as estratégias de solução são baseadas em um grafo direcionado chamado *Object Relation Diagrams* (ORDs) [Kung et al. 1995], no qual os vértices representam as classes, e as arestas os seus relacionamentos de dependência. Quando não existem ciclos de dependências no ORD, uma solução que não necessite de *stubs* é encontrada por uma simples ordenação topológica inversa do grafo, porém a maior parte dos programas desenvolvidos em Java apresentam ciclos de dependência [Melton and Tempero 2007], o que aumenta a complexidade para se encontrar uma solução e torna a resolução do problema CITO não trivial.

As estratégias baseadas em grafos [Kung et al. 1995, Tai and Daniels 1997, Traon et al. 2000, Briand and Labiche 2003, Bansal et al. 2009] possuem diversas limitações. Elas visam à minimização do número de ciclos que devem ser quebrados, mas entretanto, existem situações em que quebrar dois ciclos pode exigir menor custo do que quebrar somente um. Nestas situações, estas estratégias encontram apenas soluções subótimas [Briand et al. 2002]. Além disso, uma outra limitação é que estas estratégias são muito difíceis de serem adaptadas para considerar diversos fatores associados ao custo da construção de *stubs*, tais como: acoplamento por métodos e/ou por atributos, e outras restrições contratuais e/ou relacionadas ao desenvolvimento de software.

Estas limitações motivaram a utilização de algoritmos bioinspirados [Briand et al. 2002, Cabral et al. 2010, Pozo et al. 2011]. O primeiro trabalho [Briand et al. 2002] propôs a utilização de um Algoritmo Genético (AG) com uma função de *fitness* baseada em uma agregação de dois objetivos: o número de métodos e o número de atributos a serem simulados na construção dos *stubs*. Mas devido à dificuldade de ajustar os pesos correspondentes a estes objetivos, trabalhos mais recentes têm apresentado melhores resultados utilizando algoritmos multiobjetivos [Cabral et al. 2010, Pozo et al. 2011]. Em um destes estudos [Pozo et al. 2011] foram explorados três diferentes algoritmos multiobjetivos: NSGA-II, MTabu e PACO. O NSGA-II, um algoritmo evolutivo multiobjetivo (*Multi-objective Otimization Evolutionary Algorithm* - MOEA), foi o algoritmo que melhor se ajustou ao problema.

Apesar de estes estudos serem bastante promissores, algumas questões relacionadas ao problema CITO ainda permanecem em aberto. A primeira delas diz respeito aos objetivos utilizados. Nestes trabalhos apenas duas medidas, número de atributos e de métodos foram utilizadas, mas o problema é de fato multiobjetivo e diversas outras medidas de acoplamento entre classes que influenciam a complexidade de construir *stubs* precisam ser também consideradas. Estes estudos utilizam apenas sistemas pequenos, com poucas classes, o que gera um questionamento sobre o comportamento destas estratégias para sistemas grandes e complexos. Além disso, diferentes MOEAs existem, e não há registros de estudos de avaliação do desempenho de diferentes MOEAs para o problema em questão.

Para responder a estas questões este trabalho tem por objetivo analisar o comportamento de dois diferentes MOEAs, o NSGA-II e o SPEA2 quando aplicados ao problema CITO com quatro medidas de acoplamento associadas ao custo de construção de *stubs*. Duas outras medidas são introduzidas: número de diferentes tipos de retorno e número

de diferentes tipos de parâmetros, em adição às duas medidas tradicionalmente utilizadas: número de métodos e de atributos. Desta maneira foi possível, além da comparação dos resultados alcançados pelo NSGA-II e pelo SPEA2, analisar o comportamento dos MOEAs para resolver o problema utilizando vários objetivos, casos nos quais comumente o desempenho dos MOEAs pode ser prejudicado. O experimento conduzido utilizou quatro sistemas reais, maiores e mais complexos que aqueles utilizados nos trabalhos relacionados [Briand et al. 2002, Cabral et al. 2010, Pozo et al. 2011]. Os resultados são comparados utilizando-se três indicadores de qualidade: Distância Geracional (GD), Distância Geracional Inversa (IGD) e Distância Euclidiana a Solução Ideal (ED).

Este artigo está organizado da seguinte maneira: na Seção 2 são brevemente descritos os algoritmos multiobjetivos utilizados e na Seção 3 são discutidos aspectos da sua aplicação ao problema CITO. Na Seção 4 é dada uma descrição do estudo experimental realizado cujos resultados são apresentados e analisados na Seção 5. Finalmente, a Seção 6 contém as conclusões.

2. Algoritmos multiobjetivos

Problemas com objetivos múltiplos, ou multiobjetivo, estão presentes na maioria das disciplinas. Estes problemas são influenciados por diferentes fatores que determinam certos objetivos a serem, por exemplo, minimizados ou maximizados e para os quais não é possível encontrar uma solução única. Geralmente as funções objetivo são ditas conflitantes, e um conjunto de soluções ótimas surge. Para avaliar e determinar o grupo dessas boas soluções, utiliza-se o conceito de dominância de Pareto [Pareto 1927]. A ideia é descobrir soluções que não são dominadas por nenhuma outra no espaço de busca. Dado um conjunto de possíveis soluções para o problema, uma solução A domina uma solução B, se e somente se, A é melhor que B em satisfazer pelo menos um dos objetivos, sem ser pior que B em nenhum dos outros objetivos restantes. O conjunto de soluções não-dominadas formam a fronteira de Pareto, que fornece informações sobre o problema e é útil para resolver problemas reais, tais como os problemas da Engenharia de Software. Em muitas aplicações encontrar a fronteira de Pareto real é NP-difícil [Knowles et al. 2006], então o desafio é encontrar a fronteira mais próxima possível da fronteira de Pareto.

Algoritmos evolutivos multiobjetivos são extensões de AGs para problemas multiobjetivos que incorporam os conceitos de dominância de Pareto para criar estratégias distintas para evoluir e diversificar as soluções. Neste trabalho foram selecionados os MOEAs NSGA-II [Deb et al. 2002] e SPEA2 [Zitzler et al. 2001]. Ambos algoritmos utilizam uma população que evolui geração a geração aplicando os operadores de seleção, cruzamento e mutação.

O NSGA-II (*Non-dominated Sorting Genetic Algorithm*) [Deb et al. 2002] ordena, a cada geração, os indivíduos das populações pai e filha considerando a relação de não-dominância. Ele cria diversas fronteiras (linhas 10 e 11 da Figura 1(a)) e, depois da ordenação, as soluções com mais baixa dominância são descartadas. As fronteiras representam a estratégia de elitismo adotada pelo NSGA-II. Além disso, este algoritmo usa um operador de diversidade (*crowding distance*) que ordena os indivíduos de acordo com a sua distância em relação aos vizinhos da fronteira para cada objetivo, visando garantir maior espalhamento das soluções.

Ambas ordenações, de fronteiras e de distância de multidão, são usadas pelo ope-

rador de seleção (linhas 6 e 16 da Figura 1(a)) 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, a distância de multidão serve como critério de desempate.

```
   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      Recombinação 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         Determinar a distância de multidão para cada solução das fronteiras 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
```

(a) NSGA-II

```
   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 com substituição
12     Aplicar Cruzamento e Mutação
13 fim
```

(b) SPEA2

Figura 1. Pseudocódigos dos algoritmos adaptados de [Coello et al. 2006]

O algoritmo SPEA2 (*Strength Pareto Evolutionary Algorithm*) [Zitzler et al. 2001], cujo pseudocódigo é mostrado na Figura 1(b), além de ter sua população regular, mantém um arquivo externo que armazena soluções não-dominadas encontradas a cada geração. Para cada solução que está no arquivo e na população é calculado um valor de *strength* que também é usado no cálculo do *fitness* do indivíduo. O valor de *strength* de uma solução *i* corresponde a quantidade de indivíduos *j*, pertencentes ao arquivo e a população, dominados por *i*. Este valor é utilizado durante o processo de seleção. O *fitness* de uma solução é a soma dos valores de *strength* de todas as soluções dominadas por *i*, tanto do arquivo externo como da população regular (linha 4 da Figura 1(b)). No processo evolutivo, o arquivo externo é utilizado para complementar a nova geração com soluções não dominadas. Como o tamanho do arquivo é fixo, se este tamanho for excedido, um algoritmo de agrupamento é utilizado a fim de reduzi-lo [Coello et al. 2006] (linhas 6 e 7 da Figura 1(b)).

3. MOEAs aplicados ao problema CITO

Para aplicar os MOEAs a fim de resolver o problema CITO é preciso encontrar uma boa representação para o problema. E esta representação influencia na implementação de todos os estágios do algoritmo. Como o problema CITO é relacionado à permutação de classes, as quais formam as ordens de teste, o cromossomo é representado por um vetor cujas posições contém inteiros que representam as classes. O tamanho do vetor é igual ao número de classes de cada sistema. Então, se cada classe é representada por um inteiro, uma solução válida para um problema de 8 classes seria $(3, 5, 4, 2, 1, 7, 6, 8)$. Neste exemplo, a primeira classe a ser testada e integrada seria a classe representada pelo número 3. Esta representação é a mesma utilizada em [Cabral et al. 2010].

Outra questão relacionada aos MOEAs é a escolha das funções objetivo. Como anteriormente mencionado, podem ser usados no problema CITO várias medidas e fa-

tores como acoplamento e restrições de tempo. Neste trabalho foram usadas quatro funções baseadas em diferentes medidas de acoplamento: acoplamento de atributos, acoplamento de métodos, número de diferentes tipos de retorno [Abdurazik and Offutt 2006], número de diferentes tipos de parâmetros [Abdurazik and Offutt 2006]. As duas primeiras medidas foram usadas na maioria dos trabalhos relacionados encontrados na literatura [Ré and Masiero 2007, Briand et al. 2002, Cabral et al. 2010]. Então, considerando-se que c_i e c_j são duas classes acopladas, essas medidas de acoplamento são definidas como segue:

- Acoplamento de atributos (A) = Número de atributos localmente declarados em c_j quando referências ou apontadores para uma instância de c_j aparecem na lista de argumentos de alguns métodos de c_i, como o tipo de seu valor de retorno, na lista de atributos de c_i, ou como parâmetros locais de métodos de c_i (adaptado de [Briand et al. 2002]). Assim como no experimento realizado por [Briand et al. 2002], no caso de herança o número de atributos herdados das classes ancestrais não foi contado.
- Acoplamento de Métodos (M) = Número de métodos (inclusive construtores) localmente declarados em c_j que são invocados por métodos de c_i (adaptado de [Briand et al. 2002])[1].
- Número de tipos de retorno diferentes (Rd) = Número de tipos de retorno distintos dos métodos localmente declarados em c_j que são invocados por métodos de c_i [Abdurazik and Offutt 2006]. Retornos do tipo *void* não são contados como tipo de retorno uma vez que este tipo representa a ausência de retorno[1].
- Número de tipos de parâmetros diferentes (Pd) = Número de tipos de parâmetros distintos dos métodos localmente declarados em c_j que são invocados por métodos de c_i [Abdurazik and Offutt 2006]. Quando há sobrecarga de métodos, o número de parâmetros equivale ao total de tipos de parâmetros diferentes entre todas as implementações de cada método sobrecarregado. Desta forma, considera-se o pior caso representado por situações nas quais o acoplamento se daria pela chamada de todas as opções de sobrecarga de um dado método[1].

Os dados de entrada de cada MOEA são formados por cinco matrizes que são lidas de um arquivo de texto e são: uma (1) matriz de dependência entre as classes e outras quatro correspondentes a cada uma das métricas descritas acima; (2) matriz de acoplamento de atributos (métrica A); (3) matriz de acoplamento de métodos (métrica M); (4) matriz de diferentes tipos de retornos (métrica Rd); e (5) matriz de diferentes tipos de parâmetros (métrica Pd). A matriz de dependência é representada por um vetor de duas dimensões que contém o tipo de dependência entre as classes. Com base nessa matriz são definidas as restrições para cada um dos algoritmos. A restrição que deve ser considerada durante o processamento do algoritmo é não quebrar as dependências relacionadas a herança[2] [Briand et al. 2002]. As métricas de acoplamento A, M, Rd e Pd são usadas para calcular o *fitness* de cada solução, sendo que a soma das dependências entre as classes de cada métrica corresponde a um objetivo, e para reduzir o custo dos *stubs* deve-se minimizar todos os objetivos (métricas).

[1]Nas medidas M, Rd e Pd, em caso de herança, também são contados os métodos invocados herdados das classes ancestrais, já que a sobrecarga de métodos pode influenciar a complexidade de criação de *stubs*.

[2]Diferente do trabalho de Briand et al [Briand et al. 2002], consideramos que dependências relacionadas a Agregação podem ser quebradas e, por isso, elas não são consideradas restrições.

4. Descrição do Estudo Experimental

O estudo experimental realizado envolve a aplicação dos MOEAs NSGA-II e SPEA2, que foram implementados usando as versões disponíveis no *framework* JMetal [Durillo et al. 2010].

Foram utilizados quatro sistemas OO na realização do estudo experimental: MyBatis, JHotDraw, BCEL e JBoss. A Tabela 1 contém informações sobre os sistemas como número de classes e de linhas de código (LOC). MyBatis é um *framework* de mapeamento de classes de aplicações OO em base de dados relacionais. JHotDraw é um *framework* de gráficos bidimensionais para editores de desenho escritos em Java. Neste trabalho foi utilizado somente o pacote org.jhotdraw.draw da versão 7.5.1. JBoss AS é um servidor de aplicações Java. Neste trabalho foi utilizado somente o subsistema Management da versão 6.0.0M5. O último sistema é o BCEL (*Byte Code Engineering Library*), uma biblioteca que possibilita aos seus usuários analisar, criar e manipular binários Java. Somente o pacote org.apache.bcel.classfile da versão 5.0 foi utilizado no experimento.

Tabela 1. Sistemas utilizados no estudo experimental

Software	Versão	Classes	Dependências	LOC	Disponível em:
BCEL	5.0	45	289	2999	http://archive.apache.org/dist/jakarta/bcel/old/v5.0/
JBoss	6.0.0M5	150	367	8434	http://www.jboss.org/jbossas/downloads.html
JHotDraw	7.5.1	197	809	20273	http://sourceforge.net/projects/jhotdraw/
MyBatis	3.0.2	331	1271	23535	http://code.google.com/p/mybatis/downloads/list

Considerando a dificuldade em obter documentação arquitetural de sistemas complexos para usar no experimento, optou-se por desenvolver um *parser* para realizar a engenharia reversa do código dos sistemas obtidos a fim de identificar os tipos de relacionamentos existentes entre as classes. O *parser* desenvolvido teve como base o software AJATO[3] (*AspectJ and Java Assessment Tool*).

Para determinar os parâmetros a serem utilizados nos dois MOEAs utilizou-se uma estratégia de calibragem formada por três passos que incluem a definição: (i) do tamanho da população e do número máximo de avaliações; (ii) da taxa de cruzamento; e (iii) da taxa de mutação. Em todos os passos, os MOEAs foram executados quatro vezes com os mesmos parâmetros a fim de ter uma visão mais completa do comportamento de cada algoritmo. Durante esse processo foram utilizadas as matrizes de dependência obtidas para o sistema BCEL [Briand et al. 2002] que foi utilizado previamente em um contexto similar [Briand et al. 2002, Cabral et al. 2010]. A Tabela 2 mostra os valores adotados para o estudo experimental. O número de avaliações de *fitness* foi utilizado como critério de parada. Com isso, é possível comparar as soluções obtidas com custo computacional similar. Além disso, eles foram executados em um mesmo computador servidor e o tempo de execução foi registrado.

A fim de conhecer o comportamento de cada algoritmo para o problema CITO, cada MOEA foi executado 30 rodadas para cada sistema. Em cada rodada, cada algoritmo encontra um conjunto aproximado de soluções denominado PF_{approx}. Além disso, para cada MOEA, obtém-se o conjunto PF_{known} que é formado pelas soluções encontradas por cada algoritmo nas 30 rodadas, eliminando-se as repetidas e dominadas. Como

[3]AJATO disponível em: http://www.teccomm.les.inf.puc-rio.br/emagno/ajato/

Tabela 2. Parâmetros para o NSGA-II e SPEA2

Parâmetro	NSGA-II	SPEA2
Tamanho da População	300	300
Número de avaliações de *fitness*	20000	20000
Taxa de mutação	0,02	0,02
Taxa de Cruzamento	0,95	0,95
Tamanho do arquivo	-	250

neste caso, o conjunto Pareto real (PF_{true}) não é conhecido, no experimento o PF_{true} foi obtido através da união de todos os PF_{approx} encontrados por cada MOEA, eliminando-se as soluções dominadas e repetidas [Zitzler et al. 2003]. Os conjuntos PF_{true} e PF_{known} foram utilizados na análise e comparação dos algoritmos NSGA-II e SPEA2, realizadas por meio de três indicadores de qualidade: Distância Geracional(GD), Distância Geracional Inversa(IGD) e Distância Euclidiana a Solução Ideal(ED).

O indicador GD [van Veldhuizen and Lamont 1999] é usado para calcular a distância entre um conjunto de Pareto encontrado, o Pareto Conhecido (PF_{known}) em relação ao PF_{true}, ou seja, é uma medida de erro na qual verifica-se o quão distante está um ponto encontrado do seu correspondente mais próximo no conjunto PF_{true}. O IGD [Radziukyniene and Zilinskas 2008] é baseado no GD, porém com objetivo de calcular a distância do conjunto PF_{true} em relação ao conjunto PF_{known}, ou seja, observa-se o inverso da GD. Após obter os resultados dos indicadores GD e IGD, foi executado o teste pareado de Wilcoxon [Gárcia et al. 2009] de modo a verificar se os MOEAs são considerados estatisticamente iguais ($p - value \geq 0,05$).

O indicador Distância Euclidiana a Solução Ideal (ED) consiste em determinar os melhores valores para cada um dos objetivos entre todas as soluções de PF_{true}, encontrando um ponto considerado como uma solução ideal para o problema, então calcula-se a partir deste ponto a distância euclidiana em relação a todas as soluções de PF_{known}. Este indicador tem como base a técnica de classificação *Compromise Programming* [Cochrane and Zeleny 1973] que é usada em otimização multiobjetivo como técnica de apoio ao tomador de decisão para selecionar uma solução entre as várias encontradas.

5. Resultados e Análise

A Tabela 3 apresenta alguns resultados do experimento como a cardinalidade de PF_{true}, o número de soluções encontradas por cada um dos MOEAs e seu tempo de execução. A segunda coluna contém a cardinalidade de PF_{true} após todas as 30 execuções dos MOEAs. Na quarta coluna apresenta-se a cardinalidade média de PF_{approx} e entre parênteses o total de diferentes soluções de PF_{approx} retornadas por cada algoritmo. A partir destes resultados é possível estimar a complexidade do problema CITO para cada sistema. JBoss e BCEL são mais simples, uma vez que para estes sistemas, os dois algoritmos encontraram todas as soluções em PF_{true} em quase todas as execuções. Entretanto, JHotDraw é mais complexo, pois para ele, os algoritmos encontraram algumas das soluções em PF_{true} em cada execução e o SPEA2 encontrou um maior número de diferentes soluções do que o NSGA-II. MyBatis tem uma PF_{true} com maior cardinalidade que os outros sistemas e novamente o SPEA2 encontrou um maior número de diferentes soluções do que o NSGA-II. Apesar disso, é possível perceber que o SPEA2 requer um maior tempo

de execução com um maior desvio padrão do que o NSGA-II (quinta e sexta colunas da Tabela 3).

Os dois algoritmos encontraram uma única e igual solução para o sistema JBoss com os seguintes valores para cada objetivo: A = 10, M = 6, Rd = 2 e Pd = 9. Então, esta solução envolve criar *stubs* para simular dependências de 10 atributos e 6 métodos com 9 diferentes tipos de retorno e 2 diferentes tipos de parâmetro. Apesar de alcançar uma solução ótima, as soluções iniciais geradas pelos algoritmos multiobjetivos não são ótimas. Inicialmente os valores de *fitness* são altos e ao longo das gerações eles conseguem baixar o *fitness* até alcançar a solução supracitada.

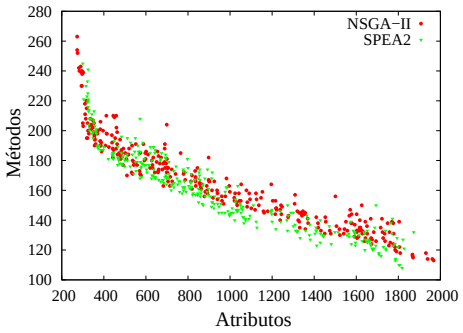
Tabela 3. Número de soluções não dominadas

Sistema	# PF_{true}	MOEA	Número de Soluções	Tempo de Execução(ms)	Desvio Padrão
BCEL	37	NSGA-II	36,8 (37)	1925,40	76,06
		SPEA2	36,4 (37)	18102,23	2184,92
JBoss	1	NSGA-II	1,0 (1)	8023,53	194,16
		SPEA2	1,0 (1)	269933,13	60335,63
JHotDraw	15	NSGA-II	8,0 (8)	13386,83	230,28
		SPEA2	7,6 (14)	22259,20	2790,60
MyBatis	612	NSGA-II	190,2 (315)	34558,13	404,42
		SPEA2	175,8 (326)	49626,30	486,18

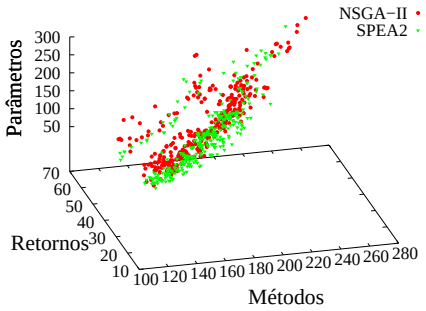
O número de classes, dependências e ciclos influencia a cardinalidade da PF_{Approx} obtida pelos algoritmos. Por exemplo, JBoss tem 150 classes, 367 dependências e 8 ciclos, e para ele os dois MOEAs alcançaram uma única solução. Por outro lado, NSGA-II e SPEA2 obtiveram 37 soluções para o BCEL que tem 45 classes, 289 dependências e 416.091 ciclos. Neste caso, o número de ciclos tem a maior influência sobre o número de soluções encontradas. Mas é possível notar que os dois sistemas que possuem mais relações de dependências, JHotDraw e MyBatis, são mais complexos para serem resolvidos. Entretanto, a respeito do JHotDraw, parece que as métricas não são altamente interdependentes e conflitantes já que o número de soluções alcançadas pelos MOEAs é menor. Pode ser que os métodos do JHotDraw sejam menos acoplados ou mais simples que os métodos do MyBatis.

Na Figura 2 são apresentados os gráficos que representam as soluções no espaço de busca para o sistema MyBatis. Apesar de as soluções apresentarem quatro dimensões, correspondentes aos quatro objetivos, somente é possível apresentar duas ou três dimensões, portanto decidiu-se por agrupar as medidas mais relacionadas: A e M na Figura 2(a) e M, Rd e Pd na Figura 2(b).

Para calcular os indicadores GD e IGD utiliza-se: o conjunto PF_{known} para cada algoritmo e o conjunto PF_{true}, formado por todas as soluções encontradas pelos dois MOEAs nas 30 rodadas, eliminando-se as repetidas e dominadas. Pela Tabela 4 pode-se observar que os algoritmos apresentam resultados bem próximos para os indicadores GD e IGD. Inclusive o teste de Wilcoxon retornou os $p - values$: 0.5229 para o BCEL, 0.9824 para o JHotDraw, e 0.2601 para o MyBatis, referente ao GD, o que indica que não existe diferença significativa para nenhum sistema. Da mesma forma, no indicador IGD, os $p - values$ retornados pelo teste de Wilcoxon não indicam diferença significativa entre o NSGA-II e o SPEA2 (BCEL= 0.08778, JHotDraw= 0.7117 e MyBatis= 0.1304).



(a) Métricas A e M



(b) Métricas M, Rd e Pd

Figura 2. Soluções no espaço de busca para o sistema MyBatis

Tabela 4. Média e Desvio Padrão dos indicadores GD e IGD

Indicador	Sistema	NSGA-II		SPEA2	
		Média	Desvio Padrão	Média	Desvio Padrão
GD	BCEL	0,0049	0,0013	0,0047	0,0010
	JBoss	0	0	0	0
	JHotDraw	0,5414	0,3912	0,5813	0,5512
	MyBatis	0,0086	0,0015	0,0092	0,0019
IGD	BCEL	0,0051	0,0017	0,0045	0,0018
	JBoss	0	0	0	0
	JHotDraw	0,6442	0,8970	0,6106	0,5378
	MyBatis	0,0138	0,0027	0,0122	0,0034

Para o cálculo da ED o custo da solução ideal para cada sistema é apresentado na segunda coluna da Tabela 5. Esta tabela também apresenta a menor distância encontrada e seu respectivo *fitness* para os sistemas usados no experimento. A menor distância alcançada encontra-se destacada em negrito. O NSGA-II alcançou as soluções que tem a menor ED a partir das soluções ideais para o JHotDraw e o MyBatis. Para os outros dois sistemas ambos MOEAs alcançaram a mesma ED.

Tabela 5. Custo da Solução Ideal e menor ED

Sistema	Custo da Solução Ideal	MOEA	Menor ED	*Fitness* da solução de menor ED
BCEL	45, 24, 0, 96	NSGA-II	**32,186**	64, 39, 15, 111
		SPEA2	**32,186**	64, 39, 15, 111
JBoss	10, 6, 2, 9	NSGA-II	**0**	10, 6, 2, 9
		SPEA2	**0**	10, 6, 2, 9
JHotDraw	28, 11, 1, 13	NSGA-II	**5,099**	28, 12, 1, 18
		SPEA2	5,744	30, 13, 1, 18
MyBatis	221, 100, 19, 83	NSGA-II	**195,0**	319, 195, 49, 219
		SPEA2	202,309	330, 214, 45, 207

A solução de menor ED alcançada pelo NSGA-II para o sistema JHotDraw tem o menor valor para dois dos quatro objetivos (A e Rd), enquanto a solução de menor ED gerada pelo SPEA2 obtém o valor mínimo somente para o objetivo Rd. É interessante observar que no caso do BCEL e do MyBatis nenhum dos MOEAs alcançou os valores mínimos para todos os objetivos em suas soluções. Em relação ao JBoss, os dois algoritmos alcançaram todos os objetivos mínimos já que encontraram a solução ideal.

NSGA-II tem soluções mais perto da solução ideal do que o SPEA2 para os sistemas JHotDraw e MyBatis. Apesar disso, não há diferença entre eles em termos de distribuição das soluções, exceto para o JHotDraw, no qual o SPEA2 é melhor. O gráfico da Figura 3 mostra o número de soluções encontradas por ambos MOEAs em relação a ED para o JHotDraw. Nota-se que, apesar de o NSGA-II ter obtido a solução de menor ED, o SPEA2 é o melhor já que ele tem mais soluções com baixas EDs.

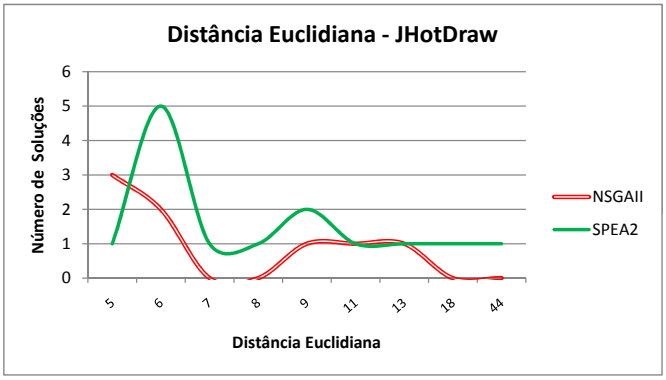


Figura 3. Número de soluções X Distância Euclidiana - JHotDraw

Logo, NSGA-II e SPEA2 tem um desempenho bem parecido para os sistemas BCEL e MyBatis neste indicador de qualidade. Mas, apesar de o NSGA-II ter alcançado as soluções de menor ED para os outros dois sistemas, a distribuição de soluções em termos de ED aponta que o SPEA2 tem o melhor desempenho para o JHotDraw.

Em termos de aplicação prática, as soluções representam um bom compromisso entre as medidas de acoplamento utilizadas. Isso pode ser ilustrado analisando o custo das soluções S1 (326, 220, 54, 286) e S2 (326, 221, 51, 275), para as métricas (A, M, Rd, Pd), obtidas pelo SPEA2 para o MyBatis. O uso das medidas tradicionais A e M indicaria que S1 é melhor que S2 já que M é menor. No entanto, o uso das métricas Rd e Pd mostra que, apesar de S2 precisar de 1 método a mais, ela requer 3 tipos de retorno e 11 tipos de parâmetros a menos que S1, implicando em um menor esforço para a criação de *stubs*.

Quando os MOEAs são usados, o testador pode usufruir da variedade de soluções encontradas de acordo com suas preferências e necessidades. O testador pode conduzir o teste de integração priorizando alguma das métricas de acoplamento e a decisão sobre a priorização de objetivos pode ser influenciada por diferentes fatores, tais como restrições do negócio, econômicas ou contratuais.

6. Conclusões

Neste trabalho foi analisado o emprego de dois MOEAs, NSGA-II e SPEA2, para resolver o problema CITO. Foi realizado um estudo experimental para minimizar a criação de *stubs* em quatro sistemas reais: BCEL, JBoss, JHotDraw e MyBatis. Além disso foram utilizados quatro objetivos que aumentam a complexidade da criação de *stubs*: acoplamento de atributos, acoplamento de métodos, número de diferentes tipos de retorno e número de diferentes tipos de parâmetros.

Os dois MOEAs alcançaram uma única solução para o sistema JBoss, indicando que neste caso os objetivos não estão em conflito. No caso do BCEL e do MyBatis, os

três indicadores apontaram equivalência entre os dois MOEAs. Quando se considera o sistema JHotDraw, os dois MOEAs são equivalentes em termos de GD e IGD, mas o SPEA2 obteve a melhor distribuição de soluções no indicador ED.

Considerando todos os sistemas, não é possível destacar um dos MOEAs como o melhor. Os dois se mostraram adequados e eficazes para os sistemas do experimento. A partir dos resultados é possível afirmar que o SPEA2 tem uma convergência ligeiramente melhor que a do NSGA-II. O SPEA2 cobre toda a fronteira PF_{Approx} e também apresenta uma boa distribuição de soluções nas regiões próximas à solução ideal. E como frequentemente os tomadores de decisão preferem soluções próximas à solução ideal, neste caso, poderia-se escolher o SPEA2. Futuramente, pode-se utilizar outros indicadores na tentativa de evidenciar a superioridade de algum dos algoritmos.

A resposta para a questão de pesquisa é sim, os dois algoritmos são eficazes para tratar o problema CITO usando quatro objetivos, já que eles alcançaram resultados similares apesar de usarem diferentes estratégias na resolução de problemas multiobjetivos. Os resultados indicam que a abordagem multiobjetivo apresenta uma variedade de boas soluções especialmente em sistemas complexos.

Futuramente, pretende-se realizar experimentos utilizando outros sistemas a fim de confirmar as evidências identificadas neste trabalho. Além disso, seria interessante analisar o desempenho de outros MOEAs nesse mesmo contexto.

7. Agradecimentos

Os autores agradecem a Edison Klafke Fillus por sua contribuição no experimento. Este trabalho foi apoiado pela CAPES/Reuni e pelo CNPq.

Referências

Abdurazik, A. and Offutt, J. (2006). Coupling-based Class Integration and Test Order. In *Proceedings of The 2006 International Workshop on Automation of Software Test*, pages 50–56.

Bansal, P., Sabharwal, S., and Sidhu, P. (2009). An investigation of strategies for finding test order during integration testing of object oriented applications. In *Proceedings of International Conference on Methods and Models in Computer Science 2009*, pages 1–8. IEEE.

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

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

Cabral, R., Pozo, A., and Vergilio, S. (2010). A Pareto Ant Colony Algorithm Applied to the Class Integration and Test Order Problem. In *22nd IFIP International Conference on Testing Software and Systems (ICTSS'10)*. Springer.

Cochrane, J. and Zeleny, M. (1973). *Multiple Criteria Decision Making*. University of South Carolina Press, Columbia.

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

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

Durillo, J., Nebro, A., and Alba, E. (2010). The jMetal framework for multi-objective optimization: Design and architecture. In *2010 IEEE Congress on Evolutionary Computation(CEC)*, pages 4138–4325, Barcelona, Spain.

Gárcia, S., Molina, D., Lozano, M., and Herrera, F. (2009). 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.

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

Kung, D. C., Gao, J., Hsia, P., Lin, J., and Toyoshima, Y. (1995). Class firewal, test order and regression testing of object-oriented programs. *J. of Object-Oriented Progr.*, 8(2).

Melton, H. and Tempero, E. (2007). An empirical study of cycles among classes in Java. *Empirical Software Engineering*, 12:389–415.

Pareto, V. (1927). *Manuel D'Economie Politique*. Ams Press, Paris.

Pozo, A., Bertoldi, G., Árias, J., Cabral, R., and Vergilio, S. (2011). Multi-objective optimization algorithms applied to the class integratio n and test order problem. *Software Tools for Technology Transfer*. Submitted.

Radziukyniene, I. and Zilinskas, A. (2008). Evolutionary Methods for Multi-Objective Portfolio Optimization. In *Proceedings of the World Congress on Engineering 2008 Vol II*.

Ré, R. and Masiero, P. C. (2007). Integration testing of aspect-oriented programs: a characterization study to evaluate how to minimize the number of stubs. In *Brazilian Symposium on Software Engineering*, pages 411–426.

Tai, K.-C. and Daniels, F. J. (1997). Test order for inter-class integration testing of object-oriented software. In *21st International Computer Software and Applications Conference*, pages 602–607. IEEE Computer Society.

Traon, Y. L., Jéron, T., Jézéquel, J.-M., and Morel, P. (2000). Efficient object-oriented integration and regression testing. *IEEE Transactions on Reliability*, pages 12–25.

van Veldhuizen, D. A. and Lamont, G. B. (1999). Multiobjective evolutionary algorithm test suites. In *Proceedings of the 1999 ACM symposium on Applied computing*, SAC '99, pages 351–357, New York, NY, USA. ACM.

Zitzler, E., Laumanns, M., and Thiele, L. (2001). SPEA2: Improving the Strength Pareto Evolutionary Algorithm. Technical Report 103, Gloriastrasse 35, CH-8092 Zurich, Switzerland.

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