CBSoft2014

Congresso Brasileiro de Software: Teoria e Prática

Congresso Brasileiro de Software: **Teoria e Prática**

28 de setembro a 03 de outubro de 2014 – Maceió/AL

V Workshop de Engenharia de Software Baseada em Busca

WESB 2014

Anais

Anais

Volume 02
ISSN 2178-6097

WESB 2014

V Workshop de Engenharia de Software Baseada em Busca

COORDENADORES DO COMITÊ DE PROGRAMA
Auri Marcelo Rizzo Vincenzi - Universidade Federal de Goiás (UFG)
Celso Gonçalves Camilo Junior - Universidade Federal de Goiás (UFG)

COORDENAÇÃO DO CBSOFT 2014
Baldoino Fonseca - Universidade Federal de Alagoas (UFAL)
Leandro Dias da Silva - Universidade Federal de Alagoas (UFAL)
Márcio Ribeiro - Universidade Federal de Alagoas (UFAL)

REALIZAÇÃO
Universidade Federal de Alagoas (UFAL)
Instituto de Computação (IC/UFAL)

PROMOÇÃO
Sociedade Brasileira de Computação (SBC)

PATROCÍNIO
CAPES, CNPq, INES, Google

APOIO
Instituto Federal de Alagoas, Aloo Telecom, Springer, Secretaria de Estado do Turismo AL, Maceió Convention & Visitors Bureau, Centro Universitário CESMAC e Mix Cópia

PROCEEDINGS

Volume 02
ISSN 2178-6097

WESB 2014

V Workshop de Engenharia de Software Baseada em Busca

PROGRAM CHAIR
Auri Marcelo Rizzo Vincenzi - Universidade Federal de Goiás (UFG)
Celso Gonçalves Camilo Junior - Universidade Federal de Goiás (UFG)

CBSOFT 2014 GENERAL CHAIRS
Baldoino Fonseca - Universidade Federal de Alagoas (UFAL)
Leandro Dias da Silva - Universidade Federal de Alagoas (UFAL)
Márcio Ribeiro - Universidade Federal de Alagoas (UFAL)

ORGANIZATION
Universidade Federal de Alagoas (UFAL)
Instituto de Computação (IC/UFAL)

PROMOTION
Sociedade Brasileira de Computação (SBC)

SPONSORS
CAPES, CNPq, INES, Google

SUPPORT
Instituto Federal de Alagoas, Aloo Telecom, Springer, Secretaria de Estado do Turismo - AL, Maceió Convention & Visitors Bureau, Centro Universitário CESMAC and Mix Cópia

Apresentação

Sejam todos bem-vindos ao Workshop de Engenharia de Software Baseada em Busca – WESB. Nesta quinta edição, o workshop se consagra com um importante fórum de discussão sobre a aplicação de técnicas de busca para solucionar problemas da Engenharia de Software no cenário nacional. No contexto do WESB, técnicas de busca englobam tanto técnicas tradicionais, como força bruta ou branch-and-bound, quanto meta-heurísticas, como algoritmos genéticos e outros algoritmos bio-inspirdados. O WESB é um workshop sobre fundamentos teóricos, de experiências práticas e de automatização da Engenharia de Software Baseada em Busca (SBSE – Search Based Software Engineering) em projetos acadêmicos e industriais.

O objetivo é a troca de experiências, opiniões e debates entre os participantes, visando a discussão do cenário atual e perspectivas de colaborações e pesquisas futuras na área. Os trabalhos submetidos para esta quinta edição foram cuidadosamente revisados por pelo menos três avaliadores do comitê de programa, que contou com pesquisadores de diferentes regiões do país, e com a colaboração de alguns revisores externos. Estes anais contêm os trabalhos selecionados dentre as submissões recebidas. Ao todo, oito trabalhos completos foram selecionados e serão apresentados nas duas seções técnicas que integram o evento. Os principais temas abordados incluem: teste de software, linha de produto de software, planejamento do projeto de software, requisitos, release e ferramentas para SBSE. Além das seções técnicas, a programação também inclui palestras convidadas e discussões sobre SBSE.

Gostaríamos de parabenizar a todos os autores dos trabalhos selecionados e também agradecer aos autores de todas as submissões realizadas. Agradecemos especialmente aos membros do comitê de programa e aos eventuais revisores por eles elencados que cumpriram com presteza os prazos estabelecidos. Sabemos que sem esta valiosa colaboração não teríamos conseguido concluir o processo de revisão no tempo previsto. Agradecemos também aos organizadores do CBSoft 2014 pela infraestrutura disponibilizada e pela oportunidade oferecida.

Desejamos um excelente evento a todos e que o WESB'2014 contribua para ampliar e consolidar a área de Engenharia de Software Baseada em Busca no Brasil.

Auri Marcelo Rizzo Vincenzi (INF/UFG)
Celso Gonçalves Camilo Junior (INF/UFG)
Coordenadores do Gerais do WESB'2014

Biografia dos Coordenadores

Auri Marcelo Rizzo Vincenzi (INF/UFG)

Auri Marcelo Rizzo Vincenzi é bacharel em Ciência da Computação pela Universidade Estadual de Londrina – UEL (1995) e mestre (1998) e doutor (2004) em Ciências da Computação e Matemática Computacional pela Universidade de São Paulo – ICMC/USP, atuando na área de Engenharia de Software. Durante o doutorado realizou sanduíche na University of Texas at Dallas UTDallas – EUA. Foi professor assistente na Universidade Católica de Santos – UNISANTOS (2006-2008), professor adjunto da Universidade Federal de Goiás – UFG (2005-2006) e professor colaborador no Centro Universitário Eurípides de Marília – UNIVEM (2004-2005). Desde 2008 é professor na Universidade Federal de Goiás – UFG. É membro da Sociedade Brasileira de Computação (SBC), da Association for Computing Machinery (ACM) e do Institute of Electrical and Electronics Engineer (IEEE).

Celso Gonçalves Camilo Junior (INF/UFG)

Celso Gonçalves Camilo Junior possui graduação em Ciência da Computação pela Pontifícia Universidade Católica de Goiás (2001), mestrado em Engenharia Elétrica e de Computação pela Universidade Federal de Goiás (2005) e doutorado em Inteligência Artificial pela Universidade Federal de Uberlândia (2010). Atualmente é professor adjunto da Universidade Federal de Goiás, diretor do APOEMA Inovação e Presidente do Comitê Gestor da METROGYN. É revisor de diversos eventos e periódicos científicos, como RITA, Neurocomputing, Neural Computing & Applications e Natural Computing Research. Tem mais de 40 trabalhos científicos publicados em veículos internacionais e nacionais, 7 Patentes/Registros e várias orientações de trabalhos de graduação, mestrado e doutorado. Além disso, é consultor e coordenador de vários projetos de pesquisa aplicada. Tem experiência na área de Ciência da Computação, com ênfase em Inteligência Computacional, atuando principalmente nos seguintes temas: Metaheuristicas, Algoritmos Evolucionários e Redes Neurais. Entre as áreas de aplicação, destacam-se: Sáude (Câncer), Redes Sociais, Engenharia de Software, E-commerce e Tomada de Decisão

Comitês Técnicos / Program Committee

Comitê do programa / Program Committee

Adriana C. F. Alvim - Universidade Federal do Estado do Rio de Janeiro (UNIRIO)
Arilo Claudio Dias Neto - Universidade Federal do Amazonas (UFAM)
Auri Marcelo Rizzo Vincenzi - Universidade Federal de Goiás (UFG)
Cássio Leonardo Rodrigues - Universidade Federal de Goiás (UFG)
Celso G. Camilo-Junior - Universidade Federal de Goiás (UFG)
Eliane Martins - Universidade Estadual de Campinas (Unicamp)
Geraldo Robson Mateus - Universidade Federal de Minas Gerais (UFMG)
Gledson Elias - Universidade Federal da Paraíba (UFPB)
Gustavo Augusto Lima de Campos - Universidade Estadual do Ceará (UECE)
Jerffeson Teixeira de Souza - Universidade Estadual do Ceará (UECE)
Keiji Yamanaka - Universidade Federal de Uberlândia (UFU)
Leila Silva - Universidade Federal de Sergipe (UFS)
Márcio Eduardo Delamaro - Universidade Federal de Goiás (UFG)
Márcio de Oliveira Barros - Universidade Federal do Estado do Rio de Janeiro (UNIRIO)
Marcone Jamilson Freitas Souza - Universidade Federal de Ouro Preto (UFOP)
Maria Cláudia Figueiredo Pereira Emer - Universidade Tecnológica Federal do Paraná (UTFPR)
Mariela Inés Cortés - Universidade Estadual do Ceará (UECE)
Mel Ó Cinnéide - University College Dublin, Ireland
Pedro de Alcântara dos Santos Neto - Universidade Federal do Piauí (UFPI)
Phil McMinn - University of Sheffield, United Kingdom
Plínio de Sá Leitão Júnior - Universidade Federal de Goiás (UFG)
Ricardo Martins de Abreu Silva - Universidade Federal de Pernambuco (UFPE)
Rosiane de Freitas Rodrigues - Universidade Federal do Amazonas (UFAM)
Silvia Regina Vergilio - Universidade Federal do Paraná (UFPR)

Revisores Adicionais / Additinal Reviewers

Adriana Cesário de Faria Alvim - Universidade Federal do Estado do Rio de Janeiro (UNIRIO)
Arilo Dias Neto - Universidade Federal do Amazonas (UFAM)
Aurora Pozo - Universidade Federal do Paraná (UFPR)
Carla G. do Nascimento Macario - Embrapa
Cassio Leonardo Rodrigues - Universidade Federal de Goiás (UFG)
Daniela Cristina Cascini Kupsch - Centro Federal de Educação Tec. de Minas Gerais (CEFET-MG)
Eliane Martins - Universidade de Campinas (UNICAMP)
Fernando Henrique Ferreira - Universidade de São Paulo (USP)
Geraldo Robson Mateus - Universidade Federal de Minas Gerais (UFMG)
Gledson Elias - Universidade Federal da Paraíba (UFPB)
Gustavo Campos - Universidade Estadual do Ceará (UECE)
Jerffeson Souza - Universidade Estadual do Ceará (UECE)

Leila Silva - Universidade Federal de Sergipe (UFS)
Marcos Chaim - Universidade de São Paulo (USP)
Maria Claudia Emer - Universidade Tecnológica Federal do Paraná (UTFPR)
Mariela Cortés - Universidade Estadual do Ceará (UECE)
Márcio Barros - Universidade Federal do Estado do Rio de Janeiro (UNIRIO)
Pedro Santos Neto - Universidade Federal do Piauí (UFPI)
Plínio de Sá Leitão-Júnior - Universidade Federal de Goiás (UFG)
Rosiane de Freitas - Universidade Federal do Amazonas (UFAM)
Silvia Vergilio - Universidade Federal do Paraná (UFPR)

Comitê organizador / Organizing Committee

COORDENAÇÃO GERAL

Baldoino Fonseca - Universidade Federal de Alagoas (UFAL)
Leandro Dias da Silva - Universidade Federal de Alagoas (UFAL)
Márcio Ribeiro - Universidade Federal de Alagoas (UFAL)

COMITÊ LOCAL

Adilson Santos - Centro Universitário Cesmac (CESMAC)
Elvys Soares - Instituto Federal de Alagoas (IFAL)
Francisco Dalton Barbosa Dias - Universidade Federal de Alagoas (UFAL)

COORDENADORES DO WESB 2014

Auri Marcelo Rizzo Vincenzi - Universidade Federal de Goiás (UFG)
Celso Gonçalves Camilo Junior - Universidade Federal de Goiás (UFG)

Índice / Table of Contents

Configuração Baseada em Busca de Linha de Produto de Software: Resultados de um Mapeamento Sistemático

Rui Angelo Matnei Filho[1], **Silvia Regina Vergilio** [1]

[1]DINF – Universidade Federal do Paraná (UFPR)
Caixa Postal 19.081 – 81.531-980 – Curitiba – PR – Brasil

{ramfilho,silvia}@inf.ufpr.br

Resumo. *Uma Linha de Produto de Software (LPS) consiste em um conjunto de produtos que compartilham características comuns. A seleção de características para formação de um produto é conhecida como configuração de LPS e pode acontecer com diversos objetivos: a customização de produtos para um cliente, o teste, o gerenciamento e a evolução de LPS. Entretanto, a configuração de LPS é um problema complexo, visto que, o número de produtos gerados cresce exponencialmente em relação ao número de características. Observa-se que este problema pode ser modelado como um problema de otimização. Por este motivo, tem sido recentemente investigado no campo de SBSE (Search-Based Software Engineering), no qual soluções baseadas em busca têm sido propostas. Considerando este fato, o presente trabalho tem como objetivo apresentar resultados de um mapeamento sistemático sobre abordagens baseadas em busca para a configuração de LPS. Esses resultados incluem análise com relação ao propósito da configuração, algoritmos e métricas utilizadas, ano e tipo de publicação. Ao final foi possível a identificação de tópicos não investigados que podem levar a um aperfeiçoamento das abordagens existentes e a novas direções de pesquisa.*

Abstract. *A Software Product Line (SPL) is a set of products that share common features. The feature selection to derive a product is known as LPS configuration and it can have different purposes: customization of products to a customer, test, management and evolution. However, to configure a LPS is a complex problem, since the number of products exponentially grows with the number of features. This problem can be modeled as an optimization problem, and has been recently investigated in the SBSE (Search-Based Software Engineering) field. Considering such fact, this paper presents results of a mapping study on search-based approaches for LPS configuration. These results include analysis of the configuration purpose, used algorithms and metrics, publication year and type. As a conclusion, it was possible to identify some non-investigated topics that can lead to the improvement of existing approaches and to new research directions.*

1. Introdução

Uma Linha de Produto de Software (LPS) consiste em um conjunto de produtos que compartilham características comuns [Pohl et al. 2005]. A engenharia de LPS permite o desenvolvimento rápido, e de baixo custo, de uma vasta gama de produtos de software com alta qualidade, facilitanto o reuso.

Uma característica (feature) consiste em uma funcionalidade ou atributo do sistema visível ao usuário e que pode ou não estar presente para configurar um determinado produto [Pohl et al. 2005]. Características são fundamentais para representar as variabilidades de um sistema. Podem ser obrigatórias, opcionais, alternativas, e assim por diante, e são geralmente modeladas através de um diagrama de características (Feature Model (FM)). A seleção ou não de determinada característica para a formação de um produto é também conhecida como configuração de LPS. Esta seleção pode ser realizada com diversos objetivos, tais como a customização de produtos para um cliente, a geração de um conjunto de produtos para o teste da LPS, o gerenciamento e a evolução da LPS.

A configuração de LPS é um problema complexo da Engenharia de Software pois o número de combinações possíveis (produtos) cresce exponencialmente com relação ao número de características. Observa-se que o problema pode ser modelado como um problema de otimização a ser resolvido por algoritmos de busca. Por este motivo o problema de configuração de LPS tem sido recentemente tema de pesquisa no campo de SBSE (Search Based Software Engineering) [Harman et al. 2012]. Entretanto, este problema é influenciado por diversos fatores; existem diferentes critérios a serem satisfeitos pelos produtos; e há muitos desafios a serem investigados. Portanto, um mapeamento sistemático [Petersen et al. 2008] sobre o tema de configuração baseada em busca de LPS poderá mostrar o que tem sido feito/proposto pelos trabalhos existentes e apontar novas direções de pesquisa para melhor lidar com este problema.

Considerando este fato como motivação, este trabalho apresenta os principais resultados obtidos através de um mapeamento sistemático sobre configuração baseada em busca de LPS. Na literatura e nos resultados do mapeamento não foi encontrado nenhum trabalho relacionado reportando os trabalhos existentes sobre este tema.

O mapeamento foi conduzido seguindo os passos propostos por Petersen et. al. (2008) com o propósito de identificar os principais propósitos das abordagens existentes, algoritmos de otimização usados, assim como as principais funções de fitness e métricas utilizadas. Ainda é feita uma análise por ano e veículo de publicação, e a identificação dos principais pontos de melhoria e novas direções de pesquisa.

Esse trabalho está dividido como segue. Na Seção 2 a metodologia adotada no mapeamento é descrita. Na Seção 3 são apresentados os resultados obtidos que são discutidos na Seção 4. A Seção 5 conclui o trabalho e mostra possíveis desdobramentos futuros.

2. O processo de mapeamento

Nesse estudo foi adotado o processo definido por Petersen et. al. (2008) que inclui as seguintes principais atividades: (a) definição das questões de pesquisa; (b) busca por trabalhos relevantes; (c) classificação; e (d) extração de informações e mapeamento. A seguir as atividades são descritas em detalhes.

2.1. Questões de Pesquisa

O principal objetivo desse mapeamento é fornecer uma visão geral sobre os trabalhos existentes em configuração baseada em busca de LPS. Para atingir esse objetivo foram elaboradas as seguintes questões:

- *RQ1: Em que local e ano os trabalhos relacionados foram publicados?* Esta pergunta tem como objetivo identificar os principais veículos de publicação, periódicos e eventos, e também relacionar o número de publicações ao longo dos últimos anos.
- *RQ2 Com que objetivo é feita a configuração de produtos?* Como mencionado anteriormente, a seleção de produtos da LPS pode ser realizada com diferentes objetivos, customização, teste, evolução, etc.
- *RQ3: Quais os principais algoritmos utilizados?* Sabe-se que diferentes algoritmos de busca existem e o uso de um algoritmo específico pode ser mais apropriado em determinados casos. Uma subquestão a ser investigada, derivada desta está relacionada à função de avaliação (fitness) utilizada, quais métricas são consideradas e qual o tratamento dado ao problema, por exemplo multi-objetivo ou não.

2.2. Busca por trabalhos

Considerando as questões de pesquisa, definiu-se um conjunto de palavras-chave. O conjunto de palavras utilizado, escrito em inglês, foi o seguinte:

(“configuration") AND (“search-based") AND (“software product line")

A busca por trabalhos e a seleção dos mais relevantes foi realizada em cinco etapas, sendo elas: (i) a busca nos repositórios; (ii) exclusão dos trabalhos repetidos; (iii) seleção por título; (iv) seleção por abstract; e (v) seleção por critérios de inclusão/exclusão.

A string de busca escolhida foi utilizada na primeira etapa e foi a mesma para todas as bases. A busca foi encerrada no dia 21 de maio de 2014. Foram utilizadas oito bases acadêmicas como apresentado na Tabela 1. A última coluna desta tabela apresenta a quantidade de trabalhos encontrados na base relacionada. O número total de trabalhos obtidos na etapa (i) foi 269. O ano de publicação, repositório e título de cada trabalho foi incluído em uma planilha para facilitar o desenvolvimento das etapas posteriores.

Tabela 1. Bases acadêmicas utilizadas no mapeamento

Base	URL	# Trabalhos Encontrados
Science Direct	http://www.sciencedirect.com/	25
Scopus	http://www.scopus.com/	40
Web of Science	http://www.isiknowledge.com	0
IEEE Xplore	http://ieeexplore.ieee.org/	1
ACM Digital Library	http://dl.acm.org/	17
Springer	http://link.springer.com/	25
Google Scholar	http://scholar.google.com.br/	149
Citeseerx	http://citeseerx.ist.psu.edu/	12

Na segunda etapa 86 trabalhos repetidos foram descartados. A partir da análise dos títulos foram descartados mais 134 trabalhos que não possuíam o foco desejado nesse mapeamento (terceira etapa). Depois disso restaram 49 trabalhos. Na quarta etapa foram descartados os trabalhos irrelevantes após a análise do abstract o que resultou em 29 trabalhos de interesse. Na última etapa considerou-se outro importante fator do mapeamento sistemático, critérios de inclusão e exclusão. Os critérios adotados estão descritos na Tabela 2. Considerando esses critérios um conjunto final de 10 trabalhos foi selecionado.

Tabela 2. Critérios de Inclusão/Exclusão

Critérios de Inclusão:
• Textos em inglês; • Trabalhos publicados em jornais, conferências ou workshops; • Disponíveis em formato digital: PDF, DOC, HTML, etc; • Com foco em configuração (seleção de características) de LPS baseada em busca.
Critérios de Exclusão:
• Abstracts, dissertações de mestrado, teses de doutorado, livros, capítulos de livros; • Trabalhos publicados com acesso restrito; • Em outros idiomas que não inglês; • Trabalhos que não abordam LPS e sem foco em configuração (seleção de características) de LPS baseada em busca;

2.3. Classificação e Extração de Informações

De acordo com as questões de pesquisa e os principais interesses, os trabalhos foram classificados em categorias dentro de duas dimensões: (i) tipo de veículo de publicação; (ii) objetivo principal do trabalho, de acordo com a Tabela 3.

Tabela 3. Esquema de Classificação

Dimensões	**Categorias**
Tipo de Publicação	Periódico, Conferência ou Workshop
Objetivo Principal	Customização, Teste, Evolução

Através da leitura dos artigos, informações necessárias para a classificação dos trabalhos foram extraídas. Adicionalmente foram extraídas informações como: título, autores, nome do veículo de publicação, ano de publicação, algoritmos e funções de fitness.

2.4. Ameaças à Validade

A principal ameça à validade deste mapeamento está associada às questões de pesquisa e ao processo conduzido. As palavras chaves utilizadas podem não refletir o objetivo da pesquisa e ter deixado de fora alguns artigos. Entretanto, procurou-se utilizar diferentes bases de busca e os passos sugeridos por Petersen et al.

Outra ameaça está associada à confiabilidade da pesquisa. Outros esquemas de classificação poderiam ter sido utilizados. A aplicação dos critérios de inclusão/exclusão e a classificação dos trabalhos envolve aspectos subjetivos. Outras pessoas poderiam ter

realizado estas tarefas de maneira diferente. Para minimizar este problema, os autores procuraram ler os artigos de maneira integral, em casos de dúvida.

Com relação à analise dos dados, as ameaças podem ser consideradas mínimas, desde que foram utilizadas apenas análises qualitativas e quantitativas.

3. Resultados

Nessa seção são apresentadas as análises qualitativas e quantitativas dos dados extraídos dos trabalhos, visando responder às questões de pesquisa propostas. Para estas análises todos os 10 artigos encontrados foram considerados. Informações sobre estes artigos são apresentadas nas Tabelas 4 e 5.

Tabela 4. Local e Ano de Publicação

Referência	Título	Local de Publicação	Ano
JSS	2011	[Guo et al. 2011]	A genetic algorithm for optimized feature selection with resource constraints in software product lines
CoRR	2012	[Henard et al. 2012]	Bypassing the Combinatorial Explosion - Using Similarity to Generate and Prioritize T-wise Test Suites for Large Software Product Lines
CMSBSE	2013	[Karimpour and Ruhe 2013]	Bi-criteria genetic search for adding new features into an existing product line
CMSBSE	2013	[Sanchez et al. 2013]	Metrics on feature models to optimize configuration adaptation at run time
CMSBSE	2013	[Sayyad et al. 2013a]	Optimum feature selection in software product lines- Let your model and values guide your search
GECCO	2013	[Wang et al. 2013]	Minimizing test suites in software product lines using weight-based genetic algorithms
ICSE	2013	[Sayyad et al. 2013c]	On the value of user preferences in search-based software engineering- A case study in software product lines
ASE	2013	[Sayyad et al. 2013b]	Scalable product line configuration - A straw to break the camel's back
SPLC	2013	[Henard et al. 2013]	Multi-objective test generation for software product lines
JSS	2014	[White et al. 2014]	Evolving feature model configurations in software product lines

3.1. RQ1: Tipo e ano das publicações

Do conjunto total de 10 artigos, a maioria, quatro deles (40%), foram publicados em conferências, 3 em workshops e 3 em periódicos. A Figura 1 resume a porcentagem dos tipos de publicações.

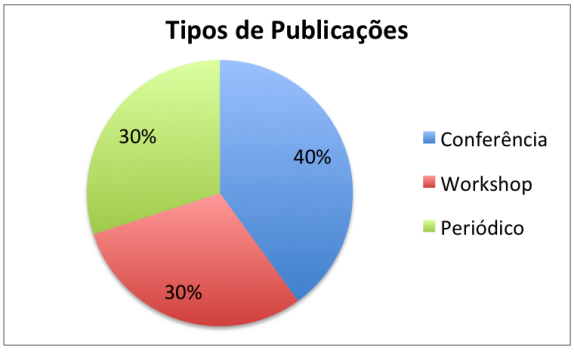


Figura 1. Percentual de trabalhos por tipo de publicação

Tabela 5. Detalhes de Cada Publicação

Objetivo	Referência	Algoritmos	Métricas
Evolução	[Karimpour and Ruhe 2013]	NSGA-II	Rint e Rvalue. Rvalue: média ponderada das variantes. Rint: média ponderada da integridade do produto
Evolução	[Sanchez et al. 2013]	Configuration Selection Algorithm (Baseado no Best-First Search Algorithm)	Consumo de memória, tempo de reconfiguração, acurácia, disponibilidade, latência, tempo de resposta em execução paralela, segurança
Customização	[Sayyad et al. 2013a]	NSGA-II, SPEA2, IBEA	Corretude, maior quantidade de características, menor quantidade de características que não foram utilizadas anteriormente, minimizar o número de defeitos, minimizar o custo
Customização	[Sayyad et al. 2013c]	IBEA, NSGA-II, ssNSGA-II, SPEA2, FastPGA, MO-Cell, MOCHC	Corretude, maior quantidade de características, menor quantidade de características que não foram utilizadas anteriormente, número de defeitos, custo
Customização	[Sayyad et al. 2013b]	NSGA-II, IBEA	Corretude, maior quantidade de características, menor quantidade de características que não foram utilizadas anteriormente, número de defeitos, custo
Customização	[Guo et al. 2011]	AG	Valor de uma solução, determinado pelo objetivo específico de domínio dividio pela Soma de todos recursos consumidos por uma característica incluída em um cromossomo da solução
Customização	[White et al. 2014]	Algoritmos para resolução CSP	Tamanho do Caminho, Custo do Caminho e Flexibilidade
Teste	[Wang et al. 2013]	AG baseado em função de agregação	TMP, FPC e FDC TMP: Porcentagem de minização; FPC: Cobertura de Pairwise, FDC: capacidade de detectar defeitos
Teste	[Henard et al. 2013]	AG	Cobertura Pairwise, Número de Produtos, Custo
Teste	[Henard et al. 2012]	Greedy, Near Optimal Prioritization	Cobertura T-wise

Os artigos foram publicados em diferentes eventos e periódicos da área de ciência da computação. Observa-se que a maioria dos eventos (4 dentre os 7 eventos da tabela) são eventos da área de SBSE, grande parte dos trabalhos está concentrada em apenas dois eventos da área de SBSE: GECCO e CMSBSE. Destaca-se o periódico Journal of System and Software (JSS) com dois trabalhos publicados.

Pode-se observar na Tabela 4 que o primeiro trabalho foi publicado em 2011. Desde então o interesse no assunto vem crescendo. Observa-se também que o maior número de artigos sobre o tema foi publicado no ano de 2013, como apresentado na Figura 2.

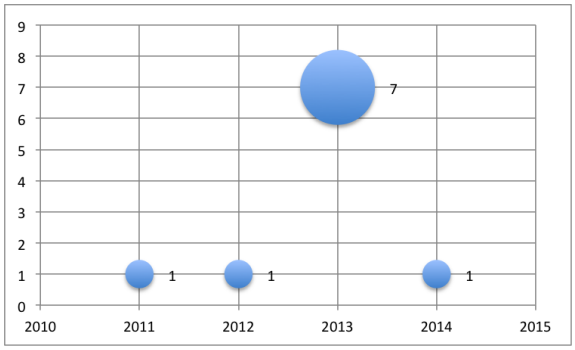


Figura 2. Número de Trabalhos por Ano

3.2. RQ2: Objetivo da seleção de produtos

Do total de 10 trabalhos selecionados, 5 trabalhos (50%), possuem como objetivo principal a customização de produtos para LPS para satisfazer algum critério determinado.

Os trabalhos apresentados por [Sayyad et al. 2013a, Sayyad et al. 2013b, Sayyad et al. 2013c] propõem a otimização da seleção de características para customização de produtos. Nesses trabalhos os autores visam a obter produtos obedecendo restrições entre as características e minimizando violações de regras no Diagrama de Características, custos e defeitos associados a uma característica. Os trabalhos utilizam a mesma função de fitness que consiste na maior corretude (menor quantidade de violações de regras), maior quantidade de características, menor quantidade de características que não foram utilizadas anteriormente, menor número de defeitos e menor custo. A representação dos Diagramas de Características é feita através de strings binárias onde o número de bits equivale ao número de características. Se o valor do bit for VERDADEIRO a característica é selecionada, caso contrário a característica é removida. São utilizados vários algoritmos genéticos destacando-se principalmente a superioridade do algoritmo IBEA sobre algoritmos mais conhecidos como NSGA-II e SPEA2.

Guo et.al. (2011) propõem em seu trabalho uma abordagem chamada GAFES que utiliza algoritmos genéticos para otimizar a seleção de características de uma LPS. A representação do Diagrama de Características segue o mesmo padrão adotado por [Sayyad et al. 2013a, Sayyad et al. 2013b, Sayyad et al. 2013c]. Os autores propõem um algoritmo baseado em algoritmo genético chamado FesTransform que repara uma seleção de característica que violou as restrições de um diagrama. A abordagem GAFES utiliza a combinação de uma seleção aleatória de características com o algoritmo FesTransform para obter uma população inicial. Ainda no contexto de customização, White et.al. (2014) apresentam uma abordagem que provê três contribuições no estudo de configuração multi-passo para LPS. Configuração Multi-Passo consiste em um problema que envolve a transição de uma configuração inicial, através de uma série de configurações intermediárias, para uma configuração final que atende a um conjunto de requisitos desejados. A representação do problema engloba entre os principais objetivos o custo de troca de configuração, o número máximo de passos no problema de configuração, os conjuntos de configuração no início e no fim da configuração, e é representada no formato CSP (Constraint Satisfaction-Problem).

Dois trabalhos têm como objetivo a evolução de LPS. O trabalho de Karimpour et.al. (2013) apresenta soluções para a evolução de LPS através da adição de novas características. Isso ocorre através de métricas que consideram valores atribuídos às características em equilíbrio com a integridade dos produtos. A representação utilizada para o problema utiliza uma string binária. Nesse caso, entretanto, para cada característica uma string do tamanho do número de produtos é adicionada. Assim, se a característica F1 possui uma string <1,0,0>, por exemplo, significa que têm-se três produtos e que apenas o primeiro deles possui a característica. O algoritmo utilizado é o NSGA-II. O contexto do trabalho de Sanchez et.al. (2013) consiste na geração de produtos em tempo real. Para isso os autores propõem um algoritmo para a seleção de uma configuração que otimize métricas de qualidade. Inicialmente os sistemas de configuração são representados por diagramas de características definidos em duas tuplas <S,D> onde S representa o conjunto de características selecionadas e D o conjunto de características não selecionadas.

Três trabalhos têm como objetivo a fase de teste de software. Eles abordam diferentes tarefas associadas a esta fase. O objetivo do trabalho de Henard et.al. (2012) é a priorização de casos de teste que consiste na organização dos casos de teste para o descobrimento mais rápido de defeitos. Esse trabalho possui como objetivo principal uma abordagem baseada em similaridade para a geração e seleção de produtos para grandes LPS. A minimização de casos de teste é tema do trabalho de Wang et.al. (2013) que utilizam algoritmos genéticos (AG) com funções de agregação dos seguintes fatores: número de testes, habilidade em revelar defeitos, cobertura. Diferentes configurações do AG são avaliadas considerando diferentes pesos na função de agregação. Outro trabalho de Henard et.al. (2013) também é baseado em AG com função de agregação para lidar com múltiplos objetivos conflitantes na seleção de testes para LPS.

3.3. RQ3: Quais os principais algoritmos utilizados?

De maneira geral a utilização de algoritmos genéticos nos trabalhos atinge os 70% (7 dentre 10). Isto demonstra a popularidade destes algoritmos. Dentre estes trabalhos, 4 utilizam o algoritmo multi-objetivo NSGA-II; 2 deles utilizam os algoritmos NSGA-II e SPEA2 paralelamente. Outro algoritmo muito utilizado em paralelo com o algoritmo NSGA-II, em 3 três trabalhos, é o IBEA (Indicator-Based Evolutionary Algorithm). Outros algoritmos genéticos como ssNSGA-II, FastPGA, MO-Cell, MOCHC, Weight-Based Genetic Algorithm (WBGA), Weight-Based Genetic Algorithm for Multi-objective Optimization (WBGA-MO) e Random-Weighted Genetic Algorithm (RWGA) também foram utilizados em uma quantidade menor de trabalhos.

Três trabalhos oferecem um tratamento multi-objetivo para o problema utilizando uma função de agregação. Além disso, alguns trabalhos utilizam outros tipos de algoritmos como Configuration Selection Algorithm [Sanchez et al. 2013] baseado no Best-First Search Algorithm, Greedy Prioratization e Near Optimal Priorization [Henard et al. 2012] e algoritmos de resolução CPS (Constraint Satisfaction Problem) [White et al. 2014] como Branch and Bound.

As funções de fitness utilizadas estão associadas a diferentes métricas. Quando o trabalho está relacionado à atividade de teste, defeitos revelados, cobertura de critérios tais como o pairwise, esforço e custo do teste relacionados ao número de casos de teste e consumo de memória são as mais utilizadas. Quando o objetivo consiste na customização de uma LPS, métricas como a quantidade de características corretas, quantidade de características não utilizadas anteriormente, número de defeitos e custo são comumente utilizadas.

4. Discussão

Nessa seção são resumidos os resultados obtidos nesse mapeamento sistemático e identificadas algumas tendências e oportunidades de pesquisa.

O esquema utilizado nos permite responder as questões propostas como segue:

- *RQ1: Em que local e ano os trabalhos relacionados foram publicados?* Os trabalhos têm sido publicados em dois eventos da área de SBSE: CMSBSE e GECCO. Outros eventos incluem ICSE, ASE, SPLC e os periódicos JSS e CoRR. As publicações ocorreram entre os anos de 2011 e 2014, destacando-se o ano de 2013 com sete trabalhos publicados.

- *RQ2: Qual o principal objetivo da seleção de produtos?* O principal objetivo dos trabalhos encontrados é a customização de LPS. Outros apresentados foram evolução, teste - minimização de conjuntos de teste e priorização de casos de teste.
- *RQ3: Quais os principais algoritmos utilizados?* O algoritmo mais utilizado é o NSGA-II, seguido pelos algoritmos SPEA2 e IBEA.

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

- Os principais veículos de publicação envolvem eventos das áreas de SBSE e de LPS;
- Considerando o ano em que os trabalhos foram publicados, observa-se que este é um tema atual de pesquisa;
- Dentre os trabalhos que possuem como objetivo o teste de LPS, poucos são os que consideram a satisfação de critérios de teste aplicados a este contexto. Este é um novo tema a ser explorado;
- A maioria dos trabalhos utiliza algoritmos genéticos, especialmente o NSGA-II. Dessa maneira, a aplicação de outros algoritmos bem como sua comparação com os algoritmos mais utilizados torna-se um importante tópico de pesquisa;
- Diferentes métricas são utilizadas nos mais diversos trabalhos, o que dificulta a comparação entre os resultados obtidos. Entretanto, isso mostra que o problema é multi-objetivo e diferentes combinações destas métricas podem ser consideradas para permitir a satisfação de diferentes condições associadas à configuração de LPS.

5. Considerações Finais

Este artigo apresentou resultados de um mapeamento sistemático de trabalhos sobre configuração baseada em busca de LPS. Os trabalhos realizam tal configuração com o propósito de customização de produtos, geração de um conjunto de produtos para o teste de LPS, e evolução de LPS. Observa-se que trabalhos futuros deverão explorar outras atividades tais como planejamento e projeto. O uso de outros algoritmos e métricas deverá também ser investigado.

Os 10 trabalhos encontrados foram analisados de acordo com o ano e veículo de publicação. Observou-se que este é um tema atual de pesquisa, e que o interesse é crescente, sendo que o maior número de trabalhos foi encontrado em 2013.

Espera-se que os resultados do mapeamento sistemático aqui apresentados possam servir como ponto de partida para novos estudos, motivação para novos mapeamentos ou ainda aprofundamento em pontos não tão estudados.

6. Agradecimentos

Os autores agradecem à Capes e ao CNPq pelo apoio financeiro.

Referências

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

Harman, M., Mansouri, S. A., and Zhang, Y. (2012). Search-based software engineering: Trends, techniques and applications. *ACM Comput. Surv.*, 45(1):11:1–11:61.

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

Henard, C., Papadakis, M., Perrouin, G., Klein, J., and Traon, Y. L. (2013). Multi-objective test generation for software product lines. In *Proceedings of the 17th International Software Product Line Conference*, SPLC '13, pages 62–71, New York, NY, USA. ACM.

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

Petersen, K., Feldt, R., Mujtaba, S., and Mattsson, M. (2008). Systematic mapping studies in software engineering. In *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, EASE'08, pages 68–77, Swinton, UK, UK. British Computer Society.

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

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

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

Sayyad, A., Ingram, J., Menzies, T., and Ammar, H. (2013b). Scalable product line configuration: A straw to break the camel's back. In *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, pages 465–474.

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

Wang, S., Ali, S., and Gotlieb, A. (2013). Minimizing test suites in software product lines using weight-based genetic algorithms. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, GECCO '13, pages 1493–1500, New York, NY, USA. ACM.

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

Applying Particle Swarm Optimisation for Solving the Next Release Problem

Arthur Rocha [1], **Leila Silva**[1], **Andre Britto**[1]

[1]Departamento de Computação – Universidade Federal de Sergipe (UFS)
São Cristovão – SE – Brasil

arthurstomp@gmail.com, leila@ufs.br, andre@ufs.br

Abstract. *The next release problem (NRP) is a problem in search-based software engineering. The goal of the problem is to maximise customer satisfaction, by choosing the requirements to be implemented in the next release of the software, from a set of dependent requirements, under the constraint of a predefined budget bound. The growing scale of requirements is a challenge in the context of NRP. In this paper we propose the use of a Particle Swarm Optimisation Algorithm to address the NRP. An experimental study, using classical instances provided by the research foundation Optimising Software by Computation from Artificial Intelligence (OSCAR), shows that the proposed approach has competitive performance when compared to algorithms commonly used in the NRP literature. Furthermore, when the complexity grows, PSO achieves best results in comparison with more complex approaches such as the Backbone-base Multilevel Algorithm (BMA).*

1. Introduction

An important task of the release planning of a software project is to select the best subset of requirements in such a way the software company can achieve maximum commercial profit, without overtaking the project budget.

To guide this decision the project manager should make a list of the requirements. This list is then ordered, based on the importance of the customer associated to each requirement. When planning the next release of the software, each requirement has also an estimated cost and the sum of the costs of a subset of requirements cannot exceed the project budget. Moreover, a requirement may depend on other requirements, generating a *requirement chain*. The cost of implementing the last element of this chain is the cost of implementing all requirements in the chain, including the last element. The project manager should decide which are the requirements to be implemented in the next release, considering the established ranking of importance, as well as the restriction on budget.

Nevertheless, as the number of requirements, customers and dependencies among requirements grow the human capability of dealing with this volume of data may introduce difficulties on finding the desired best subset of requirements. Ruhe and Saliu [Ruhe and Saliu 2005] have investigated this problem and suggested that when the complexity of the software grows, a decision-support tool is needed to make feasible the construction of the release plan.

Bagnall *et al.* (2001) have formalised this problem as a search based problem and named it as the Next Release Problem (NRP). The NRP has been largely

investigated in the context of search-based software engineering (SBSE) and several variants have been proposed considering both single-objective [Bagnall et al. 2001] [Xuan et al. 2012] [Jiang et al. 2010] [del Sagrado et al. 2010] [Yeltsin et al. 2013] and multi-objective [Zhang et al. 2007] [Durillo et al. 2009] approaches, focusing either on the requirements or on the customers, or even in both of them. In particular, recently Xuan *et al.* (2012) have addressed the growing in scale of requirements to be considered in the next release. They proposed a multilevel iterative algorithm, called Backbone-based Multilevel Algorithm (BMA), that performs reductions on the scale of the problem to construct the final set of customers. Experimental results suggested that this approach is better than direct solving approaches such as simulating annealing and genetic algorithms.

In this work we address the NRP by applying the Particle Swarm Optimisation (PSO) algorithm to find near-optimal solutions. We are not aware about the application of the heuristics PSO to the NRP. The purpose here is to investigate whether PSO may be considered to address the growing in scale of the NRP, without the inherent complexity of the BMA solution.

To validate the approach three classical instances of the NRP provided by the OSCAR lab have been used. The experiments have shown that PSO achieves superior results compared with other algorithms in the literature.

The remainder of this paper is organised as follows. In Section 2, the NRP problem is formalised. The PSO algorithm is described in Section 3. In section 4 the experiments and results achieved are discussed. Finally, in Section 5, we give some conclusions and directions for future work.

2. The Next Release Problem

In the requirements analysis phase of a software project it is necessary to select a subset of the candidate requirements for the next release, in order to maximise some sort of profit, for example, costumer satisfaction or development time, subjected to a budget bound. Each customer can request a subset of candidate requirements and may have an importance for the software company. Each requirement has a cost and may depend on other requirements to be implemented. The goal of the NRP is to select the requirements that will maximise the profit, keeping the cost under a budget bound.

This problem can be mapped to the well known 0-1 knapsack problem, [Cormen et al. 1990]. For the NRP, the weight limit of the knapsack is the project budget. The customers are the items of the solution; the importance of a customer is the value of the item. Each customer is represented by his set of requirements. Therefore, as it is a 0-1 knapsack problem, the set of requirements of each customer should be considered as a whole. Moreover, if any of these requirements is a terminal node of a requirement chain, all requirements of the chain must be included as part of the customer's set. The weight of an item (customer) is represented by the sum of the costs of all requirements associated to that customer. The goal of the problem is to maximise the satisfaction of more important customers, by giving priority in the implementation of their requirements, without violating the project budget. Thus, the solution of the NRP is a subset of customers, meaning that all of their requested requirements are in the next release of the software.

The NRP may be modelled as an acyclic digraph D, where vertices are require-

ments and edges represent the dependency between requirements. Let R be the set of all candidate requirements (vertices), $|R| = n$, and E the set of edges in D. An edge (r_i, r_j) means that the requirement r_i must be implemented before the implementation of the requirement r_j, $1 \leq i, j \leq n$, $i \neq j$. Thus if r_j is implemented in the next release, r_i must also be implemented to satisfy the dependency.

Assuming $r_s \in R$, $s \neq t$, a *requirement chain* in D, $P(r_s, r_t)$ is a maximal directed path from r_s to r_t, expressing that the implementation of r_t depends on the implementation of all requirements in the chain. Let $VP(r_s, r_t)$ be the set of vertices in $P(r_s, r_t)$, including the source vertex r_s and the target vertex r_t. As there may be several chains ending in r_t, say p chains, let $L(r_t)$ be the union of all $VP(r_s, r_t)$, that is, $L(r_t) = \bigcup_{j=1}^{p} VP_j(r_s, r_t)$.

Each requirement (vertex), $r_i \in R$ has an associated cost to be implemented, $c(r_i)$. Nevertheless, to satisfy the dependency relation, the implementation of a requirement implies the implementation of all requirements in $L(r_i)$. Thus, the final cost to implement a requirement r_i is $c(L(r_i)) = \sum_{r_j \in L(r_i)} c(r_j)$

Let S be the set of customers, $|S| = m$. Considering the digraph D, each customer s_u, $1 \leq u \leq m$ is represented by a subset R_u of R ($R_u \subseteq R$), comprising the requirements the customer s_u wants to be implemented in the next release. Thus, the cost to satisfy the customer s_u is given by $c(s_u) = \sum_{r_i \in R_u} c(L(r_i))$. Moreover, each customer s_u has a value $v(s_u)$ associated, which is an integer, representing the importance of s_u to the company. For a subset of customers $S^* \subseteq S$, $v(S^*) = \sum_{s_u \in S^*} v(s_u)$.

A solution of the NRP is a subset S' of S that maximises $v(S')$, among all $S' \subseteq S$, restricted to $\sum_{s_u \in S'} c(s_u) \leq B$, where B is the project budget.

For didactic purposes, we adopt here the representation of the solution introduced in [Xuan et al. 2012], where $S' \subseteq S$ is denoted as a set of ordered pair $X = \{(i, b) | (b = 1 \wedge s_i \in S') \vee (b = 0 \wedge s_i \notin S')\}$. By using this representation, the cost of a solution $c(S') = \sum_{s_u \in S'} c(s_u)$ is given by $c(X) = \sum_{(u,1) \in X} c(s_u)$. Analogously, $v(X) = \sum_{(u,1) \in X} v(s_u)$.

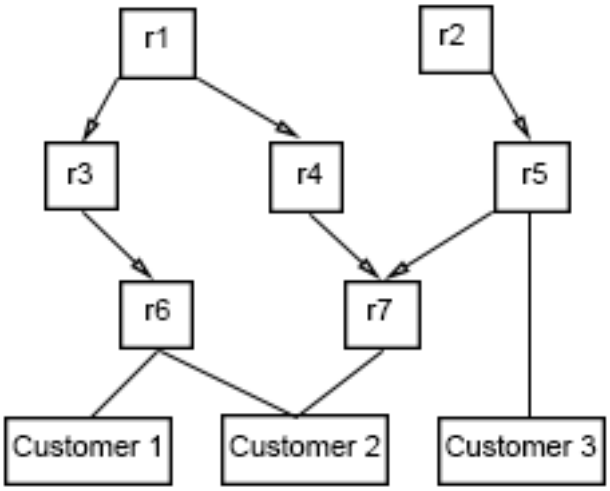


Figure 1. Example of NRP (extracted from [Bagnall et al. 2001]).

To illustrate the problem, consider Figure 1, extracted from [Bagnall et al. 2001], that presents a simple example, with set of requirements $R = \{r_1, r_2, r_3, r_4, r_5, r_6, r_7\}$ and

set of customers $S = \{s_1, s_2, s_3\}$.

The individual costs of requirements, as well as the value (importance) of customers are given in tables 1 and 2, respectively.

Table 1. Cost of requirements.

requirement(i)	1	2	3	4	5	6	7
cost(c_i)	10	6	7	1	4	6	1

Table 2. Value of customers.

customer	s_1	s_2	s_4
value	50	60	70

The dependencies among requirements are

$$E = \{(r_1, r_3), (r_1, r_4), (r_2, r_5), (r_3, r_6), (r_4, r_7), (r_5, r_7)\}.$$

The requested requirements by customers are $R_1 = \{r_6\}$, $R_2 = \{r_6, r_7\}$, $R_3 = \{r_5\}$. Thus, the final costs to satisfy the customers are $c(s_1) = c(r_1) + c(r_3) + c(r_6) = 23$, $c(s_2) = \sum_{i=1}^{7} c(r_i) = 35$ and $c(s_3) = c(r_2) + c(r_5) = 10$.The cost of a next release depends on which customers are in the solution. For this example, Table 3 expresses every possible solution.

Table 3. Costs and profits for the example of NRP.

	X	cost	profit
X_0	$\{(1,0),(2,0),(3,0)\}$	0	0
X_1	$\{(1,1),(2,0),(3,0)\}$	23	50
X_2	$\{(1,0),(2,1),(3,0)\}$	35	60
X_3	$\{(1,0),(2,0),(3,1)\}$	10	70
X_4	$\{(1,1),(2,1),(3,0)\}$	35	110
X_5	$\{(1,1),(2,0),(3,1)\}$	33	120
X_6	$\{(1,0),(2,1),(3,1)\}$	35	130
X_7	$\{(1,1),(2,1),(3,1)\}$	35	180

Observe that the cost of all requirements is 35. By considering a budget, B, equivalent to 0.7 of the total cost, $B = 24.5$, that are only two acceptable solutions (we consider a solution with no customers invalid), $X_1 = \{(1,1),(2,0),(3,0)\}$ and $X_3 = \{(1,0),(2,0),(3,1)\}$. As X_3 has a higher profit then X_1, then it is the better solution for this example.

3. Particle Swarm Optimisation Algorithm

PSO is a population meta-heuristic based on the collective behaviour of flock of birds. PSO explores a n-dimensional search space using set of solutions by updating generations. Each solution is a particle and this set is called swarm. The swarm moves through the search space in a cooperative search procedure. Each particle moves following simple rules, performed by a velocity operator [Kennedy et al. 2001].

The velocity operator is applied to every particle in the swarm and it is guided by a local and social component. The local component, called local best ($LBest$), represents the best position ever achieved by the particle. The social component, called global best ($GBest$), represents the best position ever achieved among neighbourhood particles. This neighbourhood can be a small set of particles or even the entire swarm [Eberhart and Kennedy 1995].

The movement of the particles is defined by Equation 1, presented in [Kennedy et al. 2001]. Each particle p_i, at a time step t, has a position $x(t) \in \mathbb{R}^n$, that represents a possible solution. The position of the particle, at time $t+1$, is obtained by adding its velocity, $v(t+1) \in \mathbb{R}^n$, to $x(t)$. The velocity of a particle p_i, defined by Equation 2, is based on the best position already fetched by the particle, $LBest$, and the best position already fetched by the set of neighbors of p_i, ($GBest$). Here, the neighbourhood was defined by the entire swarm, so the $GBest$ is the best position ever achieved by a particle in the swarm.

$$\overrightarrow{x}(t+1) = \overrightarrow{x}(t) + \overrightarrow{v}(t+1) \tag{1}$$

$$\overrightarrow{v}(t+1) = \overrightarrow{v}(t) + ((\varphi_1 \cdot (LBest - \overrightarrow{x}(t)) + (\varphi_2 \cdot (GBest - \overrightarrow{x}(t))) \tag{2}$$

The variables φ_1 and φ_2 in Equation 2 are coefficients that determine the influence of the particle's best position. The sum of these variables must not be greater the φ. These parameters must be defined by the user.

PSO was originally proposed to work with continuous solutions. However, as discussed in Section 2, the representation of the NRP solution here adopted, S', is denoted as a set of ordered pairs $X = \{(i,b)|(b = 1 \wedge s_i \in S') \vee (b = 0 \wedge s_i \notin S')\}$. For implementation purposes, a solution S' is a binary vector, where the position s_u is 1 if the u-th costumer is considered, otherwise is 0.

Therefore, some aspects must be considered to adapt the PSO to the NRP. First, the fitness function used in PSO for the NRP is defined to maximise the profit of the solutions. Second, since PSO was originally proposed to work with continuous solutions, to be applied to the NRP, the algorithm must be adapted to binary solutions. Here we use the approach of PSO for binary solutions discussed in [Kennedy et al. 2001]. In this approach, the particle position is a n-dimensional array of binary values, 0 and 1. Instead of being the amount of movement for each dimension, the velocity is the bias of the particle to be 0 or 1. For each position, if the velocity of the particle is high, it is more likely to select the value 1. If the velocity of the particle is low, then, it is more likely to select 0.

To perform this selection, the velocity of the particle must vary between 0 and 1. To achieve this, the same equation presented in Equation 2 is applied. However, it is used the sigmoid function [von Seggern 2006] to limit the values of the velocity. The sigmoid function, for the i-th particle on the j-th dimension is definied by Equation 3

$$s(v(t)_j) = \frac{1}{1 + \exp(-v(t)_j)} \tag{3}$$

Each particle must decide, whether the j-th dimension will be 0 or 1. As the decisions of every particle have to be stochastic, we use a random probabilistic threshold ρ, that is a array of random values between 0 and 1. So, the decision about what will be the next position of the i-th particle on the j-th dimension is defined by Equation 4.

$$if \rho_j < s(v(t)_j) \; then \; x(t+1)_j = 1 \; else \; x(t+1)_j = 0 \tag{4}$$

The pseudo-algorithm of the PSO adapted to the NRP is shown in Algorithm 1. The algorithm starts by initialising the particles with random start position and random start velocity. Then, the algorithm enters into a evolutionary loop. This loop stops when the stop criteria, defined by the user, occurs. In our algorithm, the stop criteria was a pre-defined number of iterations. Inside of the loop, each particle is considered. For each particle, the $LBest$ position and $GBest$ position must be updated if the current position ($particle.x(t)$) is better then the current $LBest$ and $GBest$ positions. Then the algorithm updates the velocity and the position of the particle. The new velocity ($particle.v_j(t+1)$) is updated following Equation 2. Futhermore, the velocity is bounded by a inferior and superior limits, again defined by the user. Finally, the particle position is updated according to Equation 4.

Algorithm 1 PSO through Binary Space

```
particles ← initialise particles
while stop criteria not found do
  for all particle ← particles do
    if fitness(particle.x(t)) > fitness(particle.LBest) then
      particle.LBest ← particle.x(t)
    end if
    if fitness(particle.x(t)) > fitness(GBest) then
      GBest ← particle.x(t)
    end if
    for all d ← dimensions do
      particle.v_j(t) = particle.v_j(t) + φ1(particle.LBest_j) − G(particle.x(t)_j)) +
      φ2(GBest_j − particle.x(t)_j)
      if v_j(t) ∈ (−V_max, V_max) then
        if ρ_j < s(particle.v_j(t)) then
          particle.x(t)_j = 1
        else
          particle.x(t)_j = 0
        end if
      end if
    end for
  end for
end while
```

4. Results and Discussion

In this section we evaluate the performance of our approach for nine NRP classical instances. Initially, we present the environment where we have executed our experiments.

Then, we describe how the classical NRP instances used in the experiments were generated, and we present results achieved by using the PSO heuristics. Finally, we discuss these results, comparing them with the results shown in [Xuan et al. 2012] for the BMA, GA and MSSA algorithms.

4.1. Experiment's details

Table 4. Generation Rules of the Classic NRP Instance Groups.

Instance group name	*nrp-1*	*nrp-2*	*nrp-3*
Requirements per level	20/40/80	20/40/80/160/320	250/500/700
Cost of requirements	1∼5/2∼8/ 5∼10	1∼5/2∼7/3∼9/ 4∼10/5∼15	1∼5/2∼8/ 5∼10
Max child requirements	8/2/0	8/6/4/2/0	8/2/0
Request of customers	1∼5	1∼5	1∼5
Customers	100	500	500
Profit of customers	10∼50	10∼50	10∼50

The algorithm has been evaluated by using classical instances of NRP, called *nrp-1*, *nrp-2* and *nrp-3* groups, introduced in [Bagnall et al. 2001]. Each group includes three instances, which consider distinct budget bounds, representing 30%, 50% and 70% of the total amount of requirements' costs. Table 4 shows the details of these groups of instances. All requirements are classified into three levels separated by the symbol " / ". A requirement in the second level may depend on some requirements in the first level, whereas a requirement in the third level may depend on some requirements in the first and second levels. An instance name is formed by the group name and the cost ratio. For example, *nrp-1-0.3* is an instance in the group *nrp-1* and the cost ratio 0.3.

The details of the instance *nrp-1-0.3* are discussed in what follows. There are three levels of requirements, with 20, 40, and 80 requirements in each level. The costs of requirements in the three levels range from 1 to 5, from 2 to 8, and from 5 to 10, respectively. A requirement in the first level has at most eight dependent requirements. Similarly, a requirement in the second level has at most two dependent requirements. There are 100 customers, each one representing from 1 to 5 requirements. In addition, each customer has an importance value between 10 and 50.

Table 5. Parameters of experiments.

Instance set name	p	n	V_{max}	V_{min}
nrp-1	200	200	4.0	-4.0
nrp-2	200	1000	7.0	-7.0
nrp-3	200	1000	7.0	-7.0

The parameters for the execution of the PSO algorithm were defined experimentally and are presented Table 5. The number of particles that will be interacting, p, the number of times that all the particles will move towards the global best, n, the maximum and minimum velocities of the particles, V_{max} and V_{min} and φ, the upper bound of the sum of random factors φ_1 and φ_2. The value of φ is a constant, 4.

The results of PSO algorithm have been obtained from 20 independent runs. As discussed earlier, PSO is here compared to BMA, GA and MSSA algorithms. The results of these algorithms are extracted from [Xuan et al. 2012] and were obtained from 10 independent runs. Here, it is considered the average and the best values of the fitness function.

4.2. Results

In Table 6 we present the results extracted from [Xuan et al. 2012] for the MSSA, GA and BMA algorithms, extended with the results we achieved using the PSO algorithm. Every line in Table 6 shows the name of the instance, its considered budget, the best result and the average result for the MSSA, GA, BMA and PSO algorithms. The cells highlighted in light gray indicate the highest average value and the cells highlighted in light dark indicate the highest best value, among all algorithms. A fairer comparison would be achieved by comparing the result of every execution of each algorithm, but these data is not available, so the comparison has been done by using the best result and average achieved in the experiments.

By considering the results of the first set of experiments, *nrp-1*, GA, BMA and PSO achieves similar performance, both on the average and on the best results. Observe that PSO outperforms the MSSA algorithm in every instance of the *nrp-1*. By comparing the BMA and the PSO algorithms, the BMA outperforms PSO at the first instance, *nrp-1-0.3*, on the average and the best results. By considering the *nrp-1-0.5* experiments, PSO achieves a better best value than the BMA, although this fact is not observed in the average result. At the last instance, *nrp-1-0.7*, both PSO and BMA achieve the same best value, but the BMA has a best average result.

By observing the second set of experiments, *nrp-2*, PSO outperforms significantly both GA and MSSA on the average and the best values. The BMA outperforms the PSO on the average, by a little difference in every instance of the *nrp-2*, but the PSO achieves a best value at the *nrp-2-0.5*.

Nevertheless, considering the last set of experiments, *nrp-3*, the PSO outperforms all the other algorithms on the average results, which suggest it is a more suitable approach for large NRP instances. By restricting to the best value, PSO outperforms BMA for the *nrp-3-0.3* and *nrp-3-0.5* instances, losing for a little difference when considering the *nrp-3-0.7* instance. Furthermore, PSO algorithm achieves the best average values for all instances considered.

In summary, considering the results of those three sets of experiments, PSO achieves similar results when compared with the BMA algorithm, which is the better than GA and MSSA algorithms as showed in [Xuan et al. 2012]. For the smaller instance, *nrp-1* and *nrp-2*, in general, BMA obtain the highest satisfaction, both on average and best values. However, the results of PSO algorithm are very close. We can highlight that PSO achieves the best results for the biggest instance, being suitable for high dimensional next release problems. Furthermore, PSO is simpler than the BMA, as the better is a multi-stage algorithm.

Table 6. Results of MSSA, GA, BMA and PSO considering the NRP classical instances

Instance	MSSA		GA		BMA		PSO	
Name	Best	Average	Best	Average	Best	Average	Best	Average
nrp-1-0.3	998	976.5	1187	1178.1	1201	1188.3	1197	1154.35
nrp-1-0.5	1536	1505.2	1820	1806.1	1824	1796.2	1827	1784.05
nrp-1-0.7	2301	2273.6	2507	2505.4	2507	2507.0	2507	2497.8
nrp-2-0.3	3220	3158.3	2794	2737.0	4726	4605.6	4578	4368.05
nrp-2-0.5	5229	5094.1	5363	5276.4	7566	7414.1	7678	7404.5
nrp-2-0.7	8002	7922.6	9018	8881.1	10987	10924.7	10917	10815.95
nrp-3-0.3	5147	5088.8	5852	5791.0	7123	7086.3	7330	7239.45
nrp-3-0.5	8725	8553.4	9639	9574.2	10897	10787.2	11018	10949.52
nrp-3-0.7	13600	13518.2	12454	12360.7	14180	10787.2	14178	14142.55

5. Conclusion

In this paper we present a preliminary investigation about the use of the PSO heuristics to solve large instances of the NRP and compare the results with some approaches in the literature.

Although several algorithms have been applied to solve the NRP, as far as we know, this is the first attempt of solving this problem using a PSO algorithm. The results of the experiments have shown that PSO seems to be a good alternative for solve complex instances of the NRP, even though a more precise statistical analyse is necessary before we can generalize the results achieved. Although PSO is a simpler approach compared with the BMA algorithm, it achieves similar results for instances of small size, and as the size grows, the PSO heuristic outperforms the BMA algorithm. Moreover, when compared with other approaches such as MSSA and GA, PSO is superior, in general.

Further works include make a statistical analysis with results of PSO and BMA algorithms to reinforce the results of the PSO, apply the PSO algorithm to real instances of NRP, in order to confirm these preliminary results, as well as investigate other possible variants of the PSO adopted here.. Moreover, we intend to develop a hybrid algorithm, by joining PSO with the BMA framework, to analyse a possible improvement in the results achieved. Finally, to investigate the use of the PSO in Multi-Objective NRP is also our next research task.

References

Bagnall, A. J., Rayward-Smith, V. J., and Whittley, I. M. (2001). The next release problem. *Information and Software Technology*, 43(14):883–890.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (1990). *Introduction to Algorithms*. McGraw-Hill Book Company.

del Sagrado, J., del guila, I. M., and Orellana, F. J. (2010). Ant colony optimization for the next release problem. *2st International Symposium on Search Based Software Engineering*, pages 67 – 76.

Durillo, J. J., Alba, E., Zhang, Y., and Nebro, A. J. (2009). A study of the multi-objective next release problem. *1st International Symposium on Search Based Software Engineering*, pages 49 – 58.

Eberhart, R. C. and Kennedy, J. (1995). A new optimizer using particle swarm theory. *Proceedings of the Sixth International Symposium on Micromachine and Human Science*, pages 39–43.

Jiang, H., Ren, Z., Zhang, J., Hu, Y., and Xuan, J. (2010). A hybrid aco algorithm for the next release problem. *IEEE Software*.

Kennedy, J., Eberhart, R. C., and Shi, Y. (2001). *Swarm intelligence*. Morgan Kaufmann Publishers.

Ruhe, G. and Saliu, M. O. (2005). The art and science of software release planning. *IEEE Software*, pages 47–53.

von Seggern, D. H. (2006). *CRC Standard Curves and Surfaces with Mathematica*. Chapman and Hall/CRC.

Xuan, J., Jiang, H., Ren, Z., and Luo, Z. (2012). Solving the large scale next release problem with a backbone based multilevel algorithm. *IEEE Transactions on Software Engineering*, 38(5):1195–1212.

Yeltsin, I., Paixo, M., and Souza, J. (2013). Uma adaptao de algoritmo gentico para o problema do proximo release com interdependncias entre requisitos. *Congresso Brasileiro de Software: Teoria e Prtica*, 4:22–31.

Zhang, Y., Harman, M., and Mansouri, S. A. (2007). The multi-objective next release problem. *GECCO*, pages 1129–1136.

Aplicação do padrão Mediator em uma abordagem de otimização de projeto de Arquitetura de Linha de Produto de Software

Giovani Guizzo[1], Thelma Elita Colanzi[2], Silvia Regina Vergilio[1]*

[1]DInf - Universidade Federal do Paraná (UFPR) – Curitiba, PR – Brasil

[2]DIN - Universidade Estadual de Maringá (UEM) – Maringá, PR – Brasil

{gguizzo,silvia}@inf.ufpr.br, thelma@din.uem.br

Resumo. *A aplicação de padrões de projeto, tais como o Strategy e Bridge, por meio de operadores de mutação no projeto baseado em busca de Arquitetura de Linha de Produto de Software (ALP) contribui para a obtenção de ALPs mais coesas e flexíveis e facilita a manutenção e evolução. Além disso, um estudo anterior mostra a viabilidade de aplicação destes padrões neste contexto. Considerando este fato, este trabalho introduz métodos automatizados para a aplicação do padrão Mediator em uma abordagem evolutiva multiobjetivo para a otimização de projeto de ALP. Estes métodos são utilizados por um operador de mutação para a obtenção de ALPs mais coesas, extensíveis e desacopladas durante o processo evolutivo. A aplicação do Mediator é exemplificada em uma ALP e as suas consequências mostram resultados promissores.*

Abstract. *The application of design patterns, such as Strategy and Bridge, through mutation operators in the search-based design of Software Product Line Architecture (PLA) contributes to obtain more cohesive and flexible PLAs, easing maintenance and evolution. In addition, a previous study shows the application viability of these patterns in this context. Considering this fact, this paper introduces automatic methods for the application of the Mediator design pattern in a multi-objective optimization approach for PLA design. These methods are used by a mutation operator to obtain PLAs which are more cohesive, extensible and decoupled during the evolutionary process. The application of Mediator is illustrated and its consequences show promising results.*

1. Introdução

Uma Linha de Produto de Software (LPS) pode ser definida como um conjunto de produtos que compartilham características comuns. Características são atributos de um sistema de software que afetam diretamente os usuários finais, satisfazem necessidades específicas de um segmento de mercado em particular e são desenvolvidas a partir de um conjunto comum de artefatos [van der Linden et al. 2007]. A Arquitetura de Linha de Produto de Software (ALP) é um artefato fundamental. Ela contém o projeto que é comum a todos os produtos derivados da LPS, ou seja, componentes para realizar todas as características

*Os autores agradecem à CAPES e ao CNPq pelo apoio financeiro.

comuns, e também as características variáveis, ou seja, contém variabilidades (o quê varia), as quais possuem pontos de variação (onde varia) e variantes (alternativas de projeto disponíveis para satisfazer uma variabilidade).

Dentre as atividades da engenharia de LPS, o projeto da ALP é uma tarefa difícil que pode se beneficiar com o uso de algoritmos de otimização. Isso se dá pela crescente complexidade dos sistemas e, em geral, pela existência de diferentes métricas de qualidade a serem consideradas pelos engenheiros, como por exemplo as que se referem à modularidade e extensibilidade de software. Neste sentido, Colanzi (2014) propõe a abordagem *Multi-objective Optimization Approach for PLA Design* (MOA4PLA) baseada em algoritmos multiobjetivos para otimizar o projeto de ALPs de modo a apoiar engenheiros no projeto e avaliação de ALP. A abordagem produz um conjunto de soluções com bons resultados para diferentes objetivos, como por exemplo extensibilidade e modularidade de ALPs. O foco dessa abordagem é o projeto de ALP, representado no diagrama de classes da UML, uma vez que esse tipo de modelo é bastante utilizado para modelar arquiteturas de software em um nível detalhado [Colanzi and Vergilio 2013].

A aplicação de padrões de projeto no contexto de LPS, como por exemplo os do catálogo GoF (*Gang of Four*) [Gamma et al. 1995], pode beneficiar o projeto de software [Colanzi and Vergilio 2012]. Esses padrões incluem soluções comuns provenientes de diversos projetos e que são amplamente utilizadas entre desenvolvedores. O uso de padrões pode favorecer alta coesão, baixo acoplamento e alta reusabilidade, inclusive em projetos de ALP. Padrões podem ainda contribuir para aumentar a extensibilidade de LPS [Oliveira-Junior et al. 2010]. Entretanto, observa-se na literatura uma falta de trabalhos sobre esse assunto. Isto deve-se ao fato de que o projeto de ALP baseado em busca é uma área recente de pesquisa. Outro motivo são os desafios a serem superados na aplicação automática de padrões de projeto. É necessário não somente conhecer padrões de projeto, mas também reconhecer e determinar padrões específicos de domínios baseando-se em requisitos de LPS [Colanzi 2014].

Considerando estes benefícios, e a falta de trabalhos relacionados que viabilizam a aplicação de padrões no projeto baseado em busca de ALP, em um trabalho anterior, Guizzo *et al.* (2013a) realizaram um estudo sobre a viabilidade de aplicar os padrões GoF no contexto da MOA4PLA e encapsular os resultados analíticos em algoritmos capazes de identificar e aplicar padrões de projeto em ALPs. Quatro padrões (*Strategy, Bridge, Mediator* e *Facade*) foram identificados como viáveis e dois deles (*Strategy* e *Bridge*) foram implementados e avaliados experimentalmente em [Guizzo et al. 2014]. Os resultados obtidos mostram que a aplicação do *Strategy* e do *Bridge* ajudou a MOA4PLA a encontrar uma maior diversidade de soluções, e em alguns casos, melhores resultados foram alcançados.

Estes resultados preliminares também serviram como motivação para implementar outros padrões que se mostraram também viáveis em [Guizzo et al. 2013b]. Portanto, o presente trabalho tem como objetivo propor métodos para verificar escopos propícios e aplicar automaticamente o padrão *Mediator* no contexto da MOA4PLA. Além disso, um estudo de caso da aplicação do padrão em uma LPS é apresentado, apontando as principais consequências encontradas como resultado. Um escopo propício para a aplicação de um padrão é um conjunto de elementos arquiteturais que podem e devem receber a aplicação do referido padrão, ou seja, em contextos onde o padrão se encaixa sem trazer anomalias

para a arquitetura. Se padrões de projeto forem aplicados de forma indiscriminada, a arquitetura pode ficar mais complexa e a aplicação pode não trazer benefícios, ou até mesmo prejudicar o projeto do software.

O padrão *Mediator* tem por objetivo intermediar a comunicação entre elementos, de modo a diminuir a complexidade e a interdependência entre eles [Gamma et al. 1995]. A maior vantagem deste padrão é o desacoplamento de elementos que possuem uma intercomunicação forte. No contexto deste trabalho, o *Mediator* pode contribuir para a obtenção, durante o processo evolutivo, de ALPs menos acopladas, mais coesas e mais extensíveis. Por esses motivos e pelo fato de já ter sido implementado, o *Mediator* foi escolhido para ser detalhado neste trabalho.

O artigo está organizado como segue. A Seção 2 apresenta a abordagem MOA4PLA para otimização de ALP. A Seção 3 descreve o operador de mutação introduzido em [Guizzo et al. 2013b, Guizzo et al. 2014]. A Seção 4 propõe os métodos de verificação e aplicação do *Mediator*. A Seção 5 contém um exemplo de aplicação do *Mediator* em uma LPS. A Seção 6 aborda trabalhos relacionados. A Seção 7 conclui o artigo e apresenta trabalhos futuros.

2. Abordagem de otimização multiobjetivo para projeto de ALP

Colanzi (2014) propôs a MOA4PLA, uma abordagem multiobjetivo de otimização de projeto de ALP, a qual foi utilizada neste trabalho. A abordagem recebe como entrada uma ALP modelada em um diagrama de classes contendo as variabilidades, pontos de variação e variantes da LPS. Para essa entrada, uma representação é instanciada de acordo com o metamodelo definido em [Colanzi and Vergilio 2013]. Tal instância representa os elementos arquiteturais como componentes, interfaces, operações e seus relacionamentos. Cada elemento é associado às características que ele realiza por meio de estereótipos da UML. Uma característica pode ser tanto comum a todos os produtos da LPS ou variável estando presente em apenas alguns destes produtos. Elementos variáveis são associados com variabilidades, pontos de variação e suas variantes.

Após transformada em uma representação, a ALP é otimizada utilizando algoritmos multiobjetivos. As funções de *fitness* disponíveis na MOA4PLA incluem métricas de coesão, de acoplamento e tamanho do projeto, além de métricas específicas de LPS, como extensibilidade de LPS, coesão baseada em características, e métricas que medem a difusão e a interação entre características. A MOA4PLA possui também operadores de mutação convencionais (Mover Método, Mover Atributo, Adicionar Classe, Mover Operação, e Adicionar Componente) e um operador que visa a melhorar modularização de características e extensibilidade da ALP (*Feature-driven Operator*) [Colanzi and Vergilio 2013].

Apesar da otimização e avaliação das soluções (projetos de ALP) serem totalmente automáticas, a decisão intelectual de seleção da melhor solução é delegada ao engenheiro ao final do processo evolutivo. Dessa forma, o propósito da MOA4PLA é justamente aumentar o número de soluções de projeto possíveis e ajudar o engenheiro a selecionar a ALP mais adequada ao seu propósito.

3. Proposta de aplicação automática de padrões na otimização de ALPs

No contexto da MOA4PLA, padrões de projeto são aplicados automaticamente por meio de um operador de mutação, primeiramente introduzido em [Guizzo et al. 2013b] e des-

crito resumidamente a seguir.

Para a definição do operador de mutação, alguns requisitos devem ser satisfeitos [Guizzo et al. 2013b]: i) o padrão deve ser aplicado em um escopo propício; ii) a aplicação do padrão deve ser coerente e não deve trazer nenhuma anomalia para a arquitetura (assim como assegurado em [Räihä et al. 2011]); iii) o padrão deve ser efetivamente aplicado em forma de mutação no processo evolutivo; e iv) o processo de identificação de escopos e aplicação do padrão deve ser totalmente automático, ou seja, aplicar o padrão sem nenhuma interferência do usuário.

O Algoritmo 1 apresenta o operador de mutação proposto em [Guizzo et al. 2013b]. Primeiramente, um padrão de projeto DP é aleatoriamente selecionado de um conjunto de padrões implementados (linha 4). Depois disso, o operador de mutação utiliza a função $f_s(A)$ para selecionar um escopo S da arquitetura A (linha 5). Uma possível implementação para a função $f_s(A)$ é a seleção aleatória de um número aleatório de classes e interfaces de A. Assim como a seleção aleatória de padrões de projeto, selecionar aleatoriamente os elementos arquiteturais mantém o aspecto aleatório da mutação no processo evolutivo. Entretanto, isso pode ser redefinido pelo usuário, sendo possível a criação de regras para a seleção de escopos e padrões.

Algoritmo 1: Pseudocódigo do operador de mutação

```
1 Entrada: A - Arquitetura a ser mutada; ρ_mutation - Probabilidade de mutação.
2 Saída: a arquitetura mutada A.
3 Início
4     DP ← seleção aleatória de um padrão de projeto em um conjunto de padrões aplicáveis;
5     S ← f_s(A);
6     se DP.verify(S) e ρ_mutation for alcançada então
7         DP.apply(S);
8     retorna A;
```

Depois disso, o método *verify()* verifica se o escopo S é um escopo propício para a aplicação do padrão de projeto DP (linha 6). Se o método de verificação retornar *verdadeiro* e a probabilidade de mutação $\rho_{mutation}$ for alcançada, então o método *apply()* é utilizado para aplicar o padrão de projeto DP no escopo S (linha 7). Os métodos *verify()* e *apply()* possuem implementação variável, específica para o padrão selecionado. Por fim a arquitetura mutada A é retornada. A próxima seção apresenta os métodos de verificação e aplicação do padrão *Mediator*.

Inicialmente, o operador de mutação foi aplicado utilizando somente os padrões *Strategy* e *Bridge* [Guizzo et al. 2013b, Guizzo et al. 2014], definindo-se os correspondentes métodos de verificação e aplicação. O presente trabalho estende o conjunto de padrões de projeto que podem ser aplicados usando o operador de mutação apresentado no Algoritmo 1 ao definir os métodos *verify()* e *apply()* para o padrão *Mediator*. Esses métodos são responsáveis por identificar os escopos propícios e efetivamente aplicar o *Mediator* em uma ALP. Eles são descritos na próxima seção.

4. Métodos de Verificação de Escopo e Aplicação do padrão Mediator

Esta seção descreve os métodos de verificação e de aplicação do padrão *Mediator*, juntamente com exemplos genéricos. Cada método de verificação *verify()* recebe um escopo S

como parâmetro e faz diversas verificações em seus elementos. Assim, cada método de verificação verifica se o escopo S é propício para a aplicação do padrão de projeto DP. Se sim, o método *apply()* de aplicação do padrão DP pode aplicá-lo em S. O método *apply()* é o método que faz a mutação propriamente dita, uma vez que ele aplica o padrão no escopo mudando, removendo e/ou adicionando elementos arquiteturais.

O método de aplicação de cada padrão primeiramente verifica se o padrão já está aplicado no escopo. Se sim, então o algoritmo corrige a estrutura do padrão caso haja alguma anomalia/irregularidade na estrutura do padrão. Caso contrário, uma nova instância do padrão é aplicada no escopo. O conceito de interesse (*concern*) foi utilizado nos métodos de verificação e aplicação do *Mediator* para identificar características relacionadas. Em LPS, uma característica pode ser considerada um interesse a ser realizado [Colanzi 2014]. Como mencionado anteriormente, na MOA4PLA as características estão associadas aos elementos arquiteturais por meio de estereótipos UML.

O conceito de evento de interesse foi utilizado para propor a implementação dos métodos do *Mediator*, seguindo uma estratégia semelhante à apresentada por [Gamma et al. 1995]. Cada objeto da classe "EventOfInterest" encapsula um evento de uma interação entre *colleagues*, a qual contém o *colleague* invocador, dados a serem utilizados na interação e a descrição do evento. Quando o evento ocorre, o *colleague* que dispara o evento (inicia a interação) utiliza a interface do *Mediator* para invocar o seu único método passando o objeto "EventOfInterest" criado. O *Mediator* utiliza esse objeto então para identificar quais *colleagues* ele deve interagir para completar a mediação.

O método *verify()* do *Mediator* verifica se: i) existe ao menos duas classes/interfaces com ao menos um interesse em comum; e ii) os elementos do item *i* possuem ao menos dois relacionamentos de uso entre si. Como o propósito do *Mediator* é justamente mediar as interações entre classes e interfaces, a exigência de classes/interfaces com uma mesma funcionalidade é dada pois assim a classe *Mediator* irá efetuar a mediação apenas para *colleagues* [Gamma et al. 1995]. *Colleagues* são elementos que interagem entre si, e que possuem funcionalidades semelhantes ou funcionalidades que compõem uma funcionalidade mais complexa.

A Figura 1.a) apresenta um exemplo de escopo propício para a aplicação do *Mediator*. Os *colleagues* são as classes "ClassA", "ClassB", "ClassC" e "ClassD" que possuem um mesmo interesse "X" e seis relacionamentos de uso entre si. Se utilizado o método de aplicação *apply()* do *Mediator* para efetuar uma mutação no escopo da Figura 1.a), o resultado esperado é o apresentado na Figura 1.b).

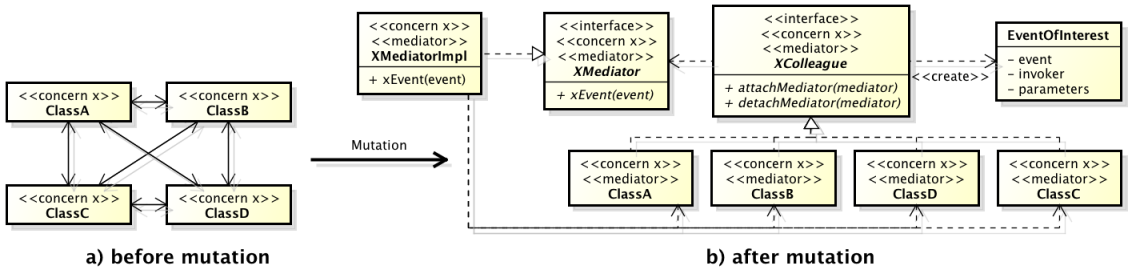


Figura 1. Exemplo de mutação utilizando o padrão *Mediator*

O pseudocódigo do método de aplicação do padrão *Mediator* é apresentado no Algoritmo 2. Para atingir o resultado da Figura 1.b), o método de aplicação cria a classe "EventOfInterest" caso ainda não exista uma (linha 3). Em seguida (linhas 4 à 11), para cada interesse associado a ao menos dois elementos que possuem ao menos dois relacionamentos entre eles, o método cria (caso ainda não exista): i) uma interface *Mediator*, "XMediator" no exemplo; ii) uma classe *Mediator*, "XMediatorImpl no exemplo; e iii) uma interface *Colleague* , "XColleague" no exemplo.

Algoritmo 2: Pseudocódigo do método *apply()* do padrão *Mediator*

```
1  Entrada: S - Escopo a ser mutado
2  Início
3      EoI ← Procura na arquitetura ou cria a classe "EventOfInterest";
4      E[] ← {e ∈ E : e usa e' ∨ e' usa e};
5      I[] ← {i ∈ E.interesses() : |{e ∈ E : i ∈ e.interesses()}| > 1};
6      para cada i ∈ I faça
7          EI[] ← {e ∈ E : (e usa e' ∨ e' usa e : i ∈ e.interesses() ∧ i ∈ e'.interesses())};
8          se |EI.relacionamentos()| ≥ 2 então
9              MI ← Procura na arquitetura ou cria a interface Mediator;
10             MC ← Procura na arquitetura ou cria a classe Mediator;
11             CI ← Procura na arquitetura ou cria a interface Colleague;
12             Faz com que os elementos de EI implementem CI;
13             Adiciona método "iEvent(eventOfInterest)" em MI e MC;
14             Adiciona métodos "attachMediator(mediator)" e "dettachMediator(mediator)"
               em CI;
15             Faz com que CI use MI e EoI;
16             Faz com que MC use todos os elementos de EI;
17             Remove relacionamentos entre elementos de EI;
18             para cada v ∈ EI.variabilidades() faça
19                 se v.variantes() ⊂ EI então Move v para CI;
20             Adiciona estereótipo «mediator» em CI, MI, MC e ∀e ∈ EI;
```

O próximo passo é fazer com que os *colleagues* implementem todas as interfaces *Colleague* dos seus interesses (linha 12) e criar um método a ser chamado quando um evento de interesse ocorrer (linha 13), "xEvent" no exemplo. Esse método recebe um objeto "EventOfInterest" e é declarado na interface e classe *Mediator*. Um método para adicionar e outro para remover (linha 14) um *Mediator* (respectivo ao seu interesse) é criado nas interfaces *Colleagues* (omitidos dos subelementos na Figura 1.b)). Esses métodos determinam qual *Mediator* os *colleagues* devem usar. Em seguida a interface *Colleague* passa a utilizar sua respectiva interface *Mediator* e a classe *Mediator* passa a utilizar diretamente cada um de seus respectivos *colleagues*. Ao contrário dos *colleagues*, a classe "XMediatorImpl" deve acessar os *colleagues* diretamente (linha 16), pois ela deve identificar quais classes devem ser utilizadas. Se um elemento possui apenas interesses que possuem *Mediators*, então esse elemento parará de ser utilizado por quaisquer outros elementos e esses elementos passarão a utilizar a interface *Mediator* (linha 17).

Em adição, se o escopo sendo mutado possui variabilidades, então o algoritmo move todas as variabilidades que possuem como variante apenas *colleagues* de uma hierarquia comum de *colleagues*, para a respectiva interface *Colleague* (linhas 18 e 19). Por fim, as interfaces *Colleague* e os elementos das hierarquias *Mediator* são rotulados com o estereótipo «mediator» para uma posterior identificação de padrões aplicados (linha 20).

Apesar de não ser aplicado diretamente em variabilidades da LPS, quando o *Mediator* é aplicado em escopos que possuam variabilidades, as variabilidades associadas aos *colleagues* são movidas para a interface *Colleague*. Isso previne que a estrutura da variabilidade fique incoerente e permite uma maior flexibilidade da mesma, uma vez que para adicionar ou remover novas variantes basta incluir os elementos no papel de *colleagues* (fazê-los implementar a interface *Colleague*). Isso ajuda a aumentar a extensibilidade da ALP e a diminuir o acoplamento entre os elementos.

5. Exemplo de Uso do Operador Proposto

Esta seção tem por objetivo apresentar a aplicabilidade do *Mediator* por meio do operador de mutação apresentado na Seção 3, detalhando como o resultado da mutação foi obtido e quais as consequências da sua aplicação. Para isso, foi utilizada a LPS *Microwave Oven Software* [Gomaa 2004], a qual é destinada a controlar fornos microondas. A mutação de exemplo foi obtida em uma das execuções do operador para o qual as seguintes entradas de dados foram informadas: A = a ALP *Microwave Oven Software*, e $\rho_{mutation} = 0,1$.

Primeiramente um padrão deve ser selecionado aleatoriamente. No cenário apresentado, o padrão *Mediator* foi selecionado. O próximo passo consiste em selecionar um escopo S de A utilizando a função $f_s(A)$. A Figura 2 ilustra os elementos arquiteturais aleatoriamente selecionados pela função $f_s(A)$ e armazenados no escopo S.

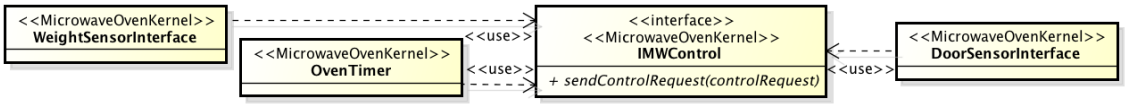


Figura 2. Escopo selecionado para a mutação pela função $f_s(A)$

Depois do escopo ter sido selecionado, verifica-se se ele é um escopo propício à aplicação do *Mediator* ou não, de acordo com o método de verificação apresentado na Seção 4. Os elementos "WeightSensorInterface", "OvenTimer", "IMWControl" e "DoorSensorInterface" são caracterizados como *colleagues* pois possuem um mesmo interesse em comum "MicrowaveOvenKernel" e interagem por meio de relacionamentos de uso. Como esses elementos possuem mais que dois relacionamentos entre si, então este é um escopo propício para a aplicação do *Mediator*. Deste modo, o método $Mediator.verify(S)$ retorna $verdadeiro$ para este escopo.

Supondo que a probabilidade de mutação seja atingida, no próximo passo o escopo recebe a aplicação do *Mediator* por meio da execução do método $Mediator.apply(S)$, cujo resultado é apresentado na Figura 3.

O método *apply()* do padrão *Mediator* primeiramente criou a classe "EventOfInterest". Em seguida, a interface "MicrowaveOvenKernelColleague" foi criada para interagir com o *Mediator*. Os *colleagues* "WeightSensorInterface", "OvenTimer", "IMWControl" e "DoorSensorInterface" passaram a implementar essa interface após a mutação. Os métodos *attachMediator()* e *dettachMediator()* foram criados. A interface "MicrowaveOvenKernelMediator" e a classe "MicrowaveOvenKernelMediatorImpl" foram criadas para mediar a interação dos elementos. Essa classe passou a utilizar todos os *colleagues* após a mutação e os relacionamentos entre esses *colleagues* foram removidos.

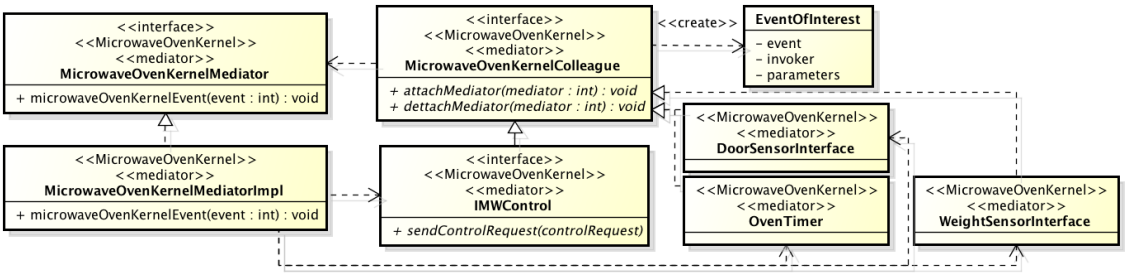


Figura 3. Escopo mutado com a aplicação do padrão de projeto *Mediator*

No exemplo apresentado nesta seção, após a aplicação do *Mediator*, o acoplamento entre os elementos diminuiu e a coesão aumentou, uma vez que os *colleagues* não precisam mais gerenciar as consequências de seus eventos. Além disso, a extensibilidade foi beneficiada, uma vez que para adicionar um novo elemento que precise interagir com os *colleagues*, basta fazer com que este implemente a interface *Colleague* e passe a ser utilizado pela classe *Mediator*. Para a remoção de um desses elementos, basta removê-lo da estrutura e remover a interação entre ele e a classe *Mediator*, não importando quantos outros *colleagues* estavam interagindo com ele por meio do *Mediator*. Essas melhorias no projeto da ALP impactam positivamente nas funções de *fitness* da MOA4PLA [Colanzi 2014] (mencionadas na Seção 2), já que tais funções levam em consideração a extensibilidade de ALP e métricas convencionais como coesão e acoplamento. Por outro lado, a aplicação do *Mediator* pode aumentar a complexidade da ALP e dificultar a manutenção.

Ao aplicar o padrão *Mediator* juntamente com outros padrões de projeto neste processo, ou até mesmo com os operadores de mutação convencionais da MOA4PLA [Colanzi 2014], a sua estrutura pode ser desconfigurada e a instância do padrão aplicada na arquitetura pode deixar de existir. Entretanto, mesmo que a instância do padrão seja desfeita, a arquitetura pode continuar semanticamente coerente e sem anomalias. A utilização de restrições para combinação de padrões é um assunto de trabalhos futuros.

6. Trabalhos Relacionados

Uma abordagem semi-automática para aplicar padrões de projeto do catálogo GoF em sistemas utilizando refatoração de código é proposta em [Cinnéide and Nixon 2001]. Entretanto, essa abordagem delega ao projetista a decisão de quais padrões aplicar e onde aplicá-los, focando apenas em remover do projetista o processo tedioso e propenso a erros de reorganização de código.

Räihä *et al.* (2011) aplicam os padrões *Facade*, *Adapter*, *Strategy*, *Template Method* e *Mediator* na síntese de arquiteturas de software com algoritmos genéticos. Os padrões são aplicados por operadores de mutação de modo a construir soluções facilmente modificáveis durante o processo evolutivo. A proposta dos autores garante uma aplicação automática coerente e válida, porém isso não significa que os padrões estão sendo aplicados em escopos propícios. Além disso, a representação da arquitetura adotada não comporta características específicas de ALPs, não sendo compatível com a representação necessária para o funcionamento da MOA4PLA [Colanzi and Vergilio 2013].

Para aplicar padrões em LPS, Keepence e Mannion (1999) desenvolveram um método que usa padrões de projeto para modelar variabilidades em modelos orientado

a objetos. Os autores definiram três padrões para modelar variantes mutuamente exclusivas, variantes inclusivas e variantes opcionais. Entretanto, não é possível identificar automaticamente escopos para a aplicação destes padrões usando apenas diagrama de classes. Outro fator é que os autores definiram a aplicabilidade dos padrões com base apenas em elementos específicos de LPS. Além disso, alguns padrões do catálogo GoF oferecem os mesmos tipos de soluções em um nível mais abstrato, como por exemplo o *Strategy* [Gamma et al. 1995].

Ziadi *et al.* (2003) propõem uma abordagem baseada no uso do padrão *Abstract Factory* para derivar modelos de produtos a partir de ALPs modeladas com UML. A abordagem inclui a especificação de uso do padrão e um algoritmo usando OCL (*Object Constraint Language*) para derivar os modelos. Apesar de ter sido comprovada como uma boa solução para modelar variabilidades, não é possível identificar, através desta abordagem, escopos propícios para a aplicação automática do padrão, principalmente considerando o projeto de LPS baseado em busca.

Ainda que estes trabalhos envolvam a aplicação de padrões em ALPs, eles não se preocupam em identificar escopos propícios à aplicação automática de padrões.

7. Conclusão

Este trabalho introduziu métodos para a identificação de escopos propícios e para a aplicação automática do padrão *Mediator* em LPS. Esses métodos são utilizados como complemento pelo operador de mutação proposto em [Guizzo et al. 2013b] e na abordagem MOA4PLA [Colanzi 2014]. O padrão é aplicado apenas quando um escopo da ALP é propício para a sua aplicação. Deste modo, evita-se a introdução de anomalias de projeto e mantém-se a integridade da arquitetura. Resultados experimentais satisfatórios com os padrões *Strategy* e *Bridge* em um trabalho anterior [Guizzo et al. 2014] mostraram as vantagens de aplicar padrões de projeto em ALPs com uma abordagem baseada em busca, o que motivou a implementação do *Mediator* neste trabalho.

No exemplo apresentado, a aplicação do *Mediator* propiciou a obtenção de uma solução de projeto mais extensível e desacoplada. Tais melhorias são detectadas pelas métricas de software que compõem as funções de *fitness* da MOA4PLA. Como destacado no exemplo, a aplicação deste padrão em ALPs contribui para o gerenciamento de variantes, de modo que estas sejam facilmente adicionadas e removidas da arquitetura. Esse aprimoramento pode ser detectado pela métrica de extensibilidade de ALP [Oliveira-Junior et al. 2010]. Apesar de aplicar padrões de projeto automaticamente, a MOA4PLA delega ao engenheiro a seleção da solução mais adequada aos seus objetivos.

Neste sentido, como trabalhos futuros, pretende-se avaliar empiricamente a aplicabilidade do padrão *Mediator* em ALPs reais em combinação com outros padrões (como o *Strategy* e o *Bridge*) assim como avaliado em [Guizzo et al. 2014]. Pretende-se também estender o trabalho para outros padrões do catálogo GoF e permitir a interação do engenheiro durante o processo evolutivo, de forma que ele possa interferir no processo bloqueando partes mal projetadas ou sugerindo melhorias em partes do projeto de ALP de uma forma semi-automática. Por fim, outra possibilidade é a criação de *minipatterns* [Cinnéide and Nixon 2001] para ajudar na reutilização de transformações de modelos.

Referências

[Cinnéide and Nixon 2001] Cinnéide, M. and Nixon, P. (2001). Automated software evolution towards design patterns. In *4th Intl Workshop on Principles of Software Evolution*, pages 162–165.

[Colanzi 2014] Colanzi, T. E. (2014). *Uma abordagem de otimização multiobjetivo para projeto arquitetural de linha de produto de software*. Tese de doutorado, Universidade Federal do Paraná, Curitiba, PR.

[Colanzi and Vergilio 2012] Colanzi, T. E. and Vergilio, S. R. (2012). Applying Search Based Optimization to Software Product Line Architectures: Lessons Learned. In *Proceedings of the 4th SSBSE*, volume 7515, pages 259–266, Riva del Garda.

[Colanzi and Vergilio 2013] Colanzi, T. E. and Vergilio, S. R. (2013). Representation of Software Product Lines Architectures for Search-based Design. In *CMSBSE Workshop of International Conference on Software Engineering (ICSE)*.

[Gamma et al. 1995] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*, volume 206 of *Addison-Wesley Professional Computing Series*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Gomaa 2004] Gomaa, H. (2004). *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures*. Addison-Wesley Professional.

[Guizzo et al. 2013a] Guizzo, G., Colanzi, T. E., and Vergilio, S. R. (2013a). Applying design patterns in product line search-based design: feasibility analysis and implementation aspects. In *Proceedings of the 32nd SCCC*. ACM.

[Guizzo et al. 2013b] Guizzo, G., Colanzi, T. E., and Vergilio, S. R. (2013b). Otimizando arquiteturas de LPS: uma proposta de operador de mutação para a aplicação automática de padrões de projeto. In *IV WESB*, pages 90–99, Brasília. SBC.

[Guizzo et al. 2014] Guizzo, G., Colanzi, T. E., and Vergilio, S. R. (2014). A Pattern-Driven Mutation Operator for Search-Based Product Line Architecture Design. In *the 6th SSBSE*, Fortaleza. Springer.

[Keepence and Mannion 1999] Keepence, B. and Mannion, M. (1999). Using patterns to model variability in product families. *IEEE Software*, (August):102–108.

[Oliveira-Junior et al. 2010] Oliveira-Junior, E. A., Gimenes, I. M. S., and Maldonado, J. C. (2010). Systematic management of variability in UML-based software product lines. *Journal of Universal Computer Science*, 16(17):2374–2393.

[Räihä et al. 2011] Räihä, O., Koskimies, K., and Mäkinen, E. (2011). Generating software architecture spectrum with multi-objective genetic algorithms. In *2011 Third (NaBIC)*, pages 29–36. IEEE.

[van der Linden et al. 2007] van der Linden, F., Schimid, K., and Rommes, E. (2007). *Software Product Lines in Action*. Springer-Verlag Berlin Heidelberg, Berlin, Germany.

[Ziadi et al. 2003] Ziadi, T., Jézéquel, J., and Fondement, F. (2003). Product Line Derivation with UML. Technical report.

Análise da Aplicação de Variantes do Algoritmo NSGA-II no Problema do Planejamento da Próxima Versão de um Software com Grande Número de Requisitos

Rodrigo Otoni, André Britto, Leila Silva

Departamento de Computação – Universidade Federal de Sergipe (UFS)
São Cristóvão, SE – Brasil – CEP 49100-000

rodrigo7491@gmail.com, {andre,leila}@ufs.br

Resumo. O *Next Release Problem* (NRP) é um problema em *Search Based Software Requirement* (SBSR). O objetivo deste problema é a maximização da satisfação dos clientes de um software através da implementação dos requisitos por estes solicitados, respeitando o limite orçamentário de desenvolvimento. O crescimento no número de requisitos é um desafio no contexto do NRP. Neste trabalho é analisado o comportamento de variantes do algoritmo NSGA-II em instâncias do NRP de larga escala extraídas da literatura. Os resultados obtidos mostram que uso de variantes é desejável, uma vez que todas as variantes testadas superam o NSGA-II.

Abstract. The Next Release Problem (NRP) is a problem in Search Based Software Requirement (SBSR). The goal of this problem is to maximize the satisfaction of the software customers by implementing the requirements requested by them, without violating the project budget. The growing number of requirements is a challenge in the context of the NRP. In this paper, we analyze the behavior of variants of the NSGA-II algorithm in large scale instances of the NRP. The results achieved are better when compared with the ones of NSGA-II algorithm, suggesting that the use of variants is a promising approach.

1. Introdução

Uma tarefa importante no planejamento das versões de um software é a seleção de quais requisitos serão implementados na próxima versão, de forma a maximizar o valor comercial do mesmo, respeitando o limite orçamentário de desenvolvimento. Para guiar este processo é necessário listar todos os requisitos que podem vir a ser desenvolvidos na próxima versão. A cada requisito é associado um custo de implementação e um conjunto de clientes que desejam que este requisito seja implementado na próxima versão. Os requisitos podem depender de outros requisitos e cada cliente tem uma importância relativa para a empresa desenvolvedora, expressa por um valor numérico. Com base nas informações coletadas, os projetistas devem, então, escolher um subconjunto de requisitos a ser implementados na próxima versão do software, visando priorizar as metas dos clientes considerados mais importantes, respeitando o orçamento disponível.

Com o crescimento do número de requisitos, clientes e relações de dependência entre requisitos, a tarefa de seleção de requisitos pode se tornar bastante difícil, sendo desejável o uso de algum método de apoio à decisão. Este problema foi formalizado como um problema de busca por Bagnall *et al.* (2001) e foi nomeado *Next Release*

Problem (NRP). Os autores relacionaram o NRP ao problema da mochila binária [Cormem *et al.* 2009], estabelecendo a seguinte comparação: o limite de peso da mochila é o limite orçamentário; os itens são os clientes; o valor dos itens é a importância dos clientes e o custo de cada item é o custo de implementação de todos os requisitos solicitados pelo cliente associado ao item, juntamente com o custo de todos os pré-requisitos necessários para a implementação dos requisitos solicitados. O objetivo do problema é, então, a seleção de um subconjunto de clientes, os quais terão suas solicitações completamente atendidas, visando maximizar a satisfação dos clientes mais importantes, respeitando a restrição financeira.

Por se tratar de um problema NP-difícil [Bagnall *et al.* 2001], o NRP vem sendo vastamente investigado no contexto de *Search Based Software Requirements* (SBSR), área que visa a solução de problemas de Engenharia de Requisitos através da aplicação de algoritmos de busca. Várias variantes que lidam tanto com a modelagem mono-objetivo [Ruhe e Saliu 2005][Van den Akker *et al.* 2007][Xuan *et al.* 2012], quanto com a modelagem multi-objetivo [Zhang *et al.* 2007][Durillo *et al.* 2009][Zhang e Harman 2010][Zhang *et al.* 2011][Cai e Wei 2013] foram propostas para o NRP. Problemas de otimização multi-objetivo possuem mais de uma função objetivo para otimizar. Nestes problemas, as funções objetivo que são otimizadas são conflitantes, logo não há somente uma melhor solução, mas um conjunto com as melhores soluções. A abordagem multi-objetivo é desejável para o NRP, dado que os objetivos (maximização da satisfação dos clientes e minimização do custo) são naturalmente conflitantes e quanto mais possíveis configurações da próxima versão forem analisadas, melhor será a decisão efetuada pelos projetistas.

A primeira abordagem multi-objetivo do NRP foi proposta por Zhang *et al.* (2007). Neste trabalho os autores citam a natureza multi-objetivo do problema e sugerem uma variante que considera dois objetivos: a satisfação dos clientes, que deve ser maximizada, e o custo total de implementação dos requisitos, o qual deve ser minimizado. Foram realizados testes comparativos com algumas meta-heurísticas nesta formulação, sendo que o algoritmo NSGA-II [Deb *et al.* 2002] se mostrou mais eficiente que as demais técnicas investigadas.

Um trabalho posterior, feito por Durillo *et al.* (2009), comparou o desempenho dos algoritmos MOCell e NSGA-II em várias instâncias do MONPR (Multi-Objective NRP). Os resultados obtidos mostraram que o NSGA-II se comporta melhor, tanto em relação à convergência ao fronte de Pareto ideal, quanto à dispersão, quando as instâncias se tornam maiores, sendo considerado o algoritmo com melhor escalabilidade. Outro resultado interessante obtido neste trabalho foi a descoberta de que, em geral, boas soluções são compostas de alguns requisitos que provêem grande satisfação, juntamente com vários requisitos de baixo custo, o que inviabiliza o uso de técnicas mais simples, como algoritmos gulosos.

O impacto de relações entre requisitos em uma abordagem multi-objetivo é discutido no trabalho de Zhang e Harman (2010), no qual se propõe a análise do impacto das relações de pré-requisição, co-requisição, exclusão e interligação de valor e custo. Os autores propõem uma modificação baseada em arquivo do algoritmo NSGA-II e a compararam com o algoritmo NSGA-II original [Deb *et al.* 2002]. Os resultados dos experimentos mostraram que o algoritmo modificado se comportou de forma superior

quando submetido a instâncias com relações entre requisitos, e também que a relação de co-requisição é a que apresenta maior dificuldade em relação às demais relações.

Além das abordagens citadas, pode-se encontrar trabalhos na literatura com outras modelagens para o MONRP, como, por exemplo, o trabalho de Zhang *et al.* (2011), no qual se considera cada cliente como um objetivo a ser maximizado, sendo o custo tratado como uma restrição, e também diversos métodos para obtenção de melhores resultados para este problema, como o algoritmo híbrido proposto no trabalho de Cai e Wei (2013).

Um dos focos atuais de pesquisa na área de Engenharia de Requisitos é o tratamento de grande número de requisitos. Em geral os métodos de busca estudados apresentam baixo desempenho à medida que instâncias maiores do NRP são consideradas. Um dos trabalhos recentes sobre o NRP de larga escala é o de Xuan *et al.* (2012), que propõe um algoritmo mono-objetivo para aplicação neste tipo de instância.

Este trabalho se propõe a investigar novas variantes do algoritmo NSGA-II aplicadas ao NRP multi-objetivo de larga escala. Os trabalhos relacionados mencionados anteriormente indicam que o algoritmo NSGA-II é o mais indicado para o NRP multi-objetivo, sendo, então, desejável se observar o comportamento do NSGA-II nos problemas de larga escala. Além disso, é avaliada a influência do uso de modificações desse algoritmo na busca por melhores soluções. As modificações utilizadas consistem de duas variantes retiradas da literatura, que são novos algoritmos para promover dispersão, método SEED [Zheng *et al.* 2012] e método NSGA-II$_R$ [Fortin e Parizeau 2013], e diferentes métodos de inicialização da população.

O NSGA-II e suas variantes são analisados em cinco instâncias de teste do NRP apresentadas em Xuan *et al.* (2012). Para medir qual algoritmo obteve o melhor resultado é utilizada medida hipervolume. O hipervolume serve como guia para identificar qual algoritmo multi-objetivo obteve melhor resultado [Beume *et al.* 2009]. Os algoritmos são comparados através de testes estatísticos.

O restante deste artigo está organizado da seguinte forma: a seção 2 trata da formalização do modelo do NRP utilizado, a seção 3 descreve os algoritmos investigados, a seção 4 apresenta os experimentos realizados e os resultados obtidos e, por fim, na seção 5 são apresentadas as considerações finais e direções de trabalhos futuros.

2. Formalização do Problema

A seleção de requisitos a serem implementados na próxima versão é uma fase importante do desenvolvimento de qualquer projeto de software. Nesta etapa deve-se selecionar o subconjunto de requisitos que gera a maior satisfação dos clientes, respeitando o limite orçamentário.

A cada cliente está associado um conjunto de requisitos que se deseja que sejam implementados, e uma importância relativa para o desenvolvedor do software por ele utilizado. Além disso, cada requisito pode possuir dependências de outros requisitos.

Na formalização adotada, um cliente só fica satisfeito quando todo o subconjunto de requisitos por ele solicitado é implementado. A única relação entre requisitos considerada é a de pré-requisição, a qual é mapeada através de um dígrafo, como descrito a seguir.

O problema consiste de dois objetivos, a satisfação dos clientes, que deve ser maximizada, e o custo de implementação, que deve ser minimizado, sendo que estes objetivos são naturalmente conflitantes. A meta é, então, identificar um conjunto de possíveis soluções localizadas no fronte de Pareto ideal, visando auxiliar os projetistas de software na definição dos requisitos a serem implementados na próxima versão, pela análise de várias configurações, com diferentes faixas de custo e satisfação dos clientes.

O problema pode ser descrito como a seguir. Seja D um dígrafo acíclico, no qual os vértices representam requisitos e as arestas representam relações de dependência. Seja R o conjunto de todos os requisitos, $|R| = n$, e A o conjunto de todas as arestas de D. Uma aresta (r_i, r_j) significa que o requisito r_i deve ser implementado antes do requisito r_j, $1 \leq i, j \leq n$, $i \neq j$. Logo, caso r_j seja implementado na próxima versão, r_i também o será para satisfazer esta dependência.

Supondo $r_s, r_t \in R$, $s \neq t$, uma `cadeia de requisitos` em D, $P(r_s, r_t)$ é um caminho orientado maximal de r_s a r_t, o qual expressa a necessidade da implementação de todos os requisitos na cadeia para a posterior implementação de r_t. Seja $VP(r_s, r_t)$ o conjunto de todos os vértices em $P(r_s, r_t)$. Como podem haver x cadeias terminando em r_t, temos que a união de todos os $VP(r_s, r_t)$, denominada $L(r_t)$, é dada por $L(r_t) = \bigcup_{j=1}^{x} VP_j(r_s, r_t)$.

Cada requisito tem um custo associado de implementação, $c(r_i)$. Como a implementação de r_i implica obrigatoriamente na implementação de $L(r_i)$ para satisfação da relação de dependência, temos que o custo final de implementação de r_i é $c\big(L(r_i)\big) = \sum_{r_j \in L(r_i)} c(r_j)$.

Seja S o conjunto de todos os clientes, $|S| = m$. Considerando o dígrafo D, cada cliente S_u, $1 \leq u \leq m$, é representado por um subconjunto R_u de R ($R_u \subseteq R$), que representa os requisitos que o cliente S_u deseja que sejam implementados. O custo para satisfazer o cliente S_u é dado, então, por $c(S_u) = \sum_{r_i \in R_u} c(L(r_i))$. Temos ainda que cada cliente S_u tem um valor $v(S_u)$ a ele associado, que representa sua importância para o desenvolvedor do software por este usado. Para um subconjunto S^* de clientes ($S^* \subseteq S$), temos que $v(S^*) = \sum_{S_u \in S^*} v(S_u)$.

O objetivo do problema é, então, encontrar um subconjunto $S' \subseteq S$ que maximiza a satisfação total dos clientes e minimiza o custo de implementação da próxima versão, ou seja, S' deve otimizar as funções f_1 e f_2 como definidas a seguir.

$$\text{Maximize } f_1 = \sum_{S_u \in S'} v(S_u)$$

e

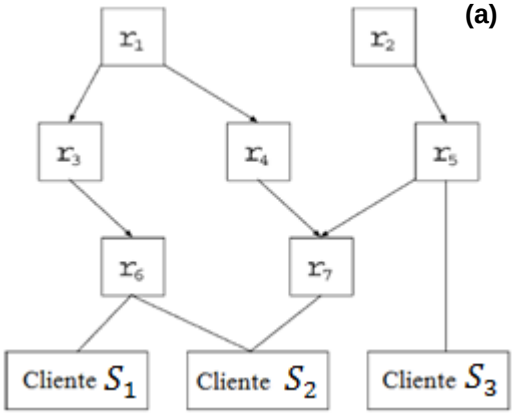
$$\text{Minimize } f_2 = \sum_{S_u \in S'} c(S_u)$$

Como o processo de minimização de uma função f é o mesmo de maximização de $-f$, neste trabalho objetiva-se a maximização das funções f_1 e $-f_2$.

Dado que as funções f_1 e f_2 são conflitantes por natureza, uma resposta ao problema consiste de um conjunto de soluções não-dominadas, ou seja, de soluções para as quais não se pode obter melhoria no valor de uma função sem a degradação do valor da outra função. Os projetistas, em posse de tais possíveis soluções, podem escolher a que mais lhes parecer viável, tendo em vista os recursos financeiros disponíveis.

Para ilustração do problema, considere a figura 1, a qual representa uma pequena instância do NRP, com o conjunto de requisitos $R = \{r_1, r_2, r_3, r_4, r_5, r_6, r_7\}$ e o conjunto de clientes $S = \{S_1, S_2, S_3\}$.

(a)



(b)

Req.	Custo (c_i)
r_1	10
r_2	6
r_3	7
r_4	1
r_5	4
r_6	6
r_7	1

(c)

Cliente	Valor $v(S_i)$
S_1	50
S_2	60
S_3	70

Figura 1. (a) Exemplo de instância do NRP (adaptado de [Bagnall *et al*, 2001]). (b) Custo dos requisitos da instância. (c) Valor dos clientes da instância.

As relações de dependência em D são definidas a seguir.

$$A = \{(r_1, r_3), (r_1, r_4), (r_2, r_5), (r_3, r_6), (r_4, r_7), (r_5, r_7)\}$$

Os requisitos solicitados por cada cliente são $R_1 = \{r_6\}$, $R_2 = \{r_6, r_7\}$ e $R_3 = \{r_5\}$. Logo, o custo de satisfação de cada cliente é $c(S_1) = c\big(L(r_6)\big) = 23$, $c(S_2) = c\big(L(r_6)\big) + c\big(L(r_7)\big) = 35$, $c(S_3) = c\big(L(r_5)\big) = 10$. O valor das funções f_1 e f_2 depende de quais clientes são selecionados. A tabela 3 apresenta todas as possíveis soluções deste exemplo.

Observando-se a tabela 1, pode-se notar que as soluções não-dominadas são {}, $\{S_3\}$, $\{S_1, S_3\}$ e $\{S_1, S_2, S_3\}$, uma vez que, nestas soluções, o valor de uma função não pode ser melhorado sem a degradação do valor da outra.

Tabela 1. Possíveis soluções para a instância da figura 1

Clientes	-	S_1	S_2	S_3	S_1, S_2	S_1, S_3	S_2, S_3	S_1, S_2, S_3
f_1	0	50	60	70	110	120	130	180
$-f_2$	0	-23	-35	-10	-35	-33	-35	-35

3. Algoritmos Investigados

Este trabalho visa a investigar o comportamento do algoritmo NSGA-II, proposto por Deb *et al.* (2002), e duas variações do mesmo, propostas por Zheng *et al.* (2012), e Fortin e Parizeau (2013), no contexto do NRP.

O *Non-Dominated Sorting Genetic Algorithm II* (NSGA-II) é um algoritmo evolutivo multi-objetivo bastante utilizado, tanto por sua habilidade para obtenção de valores satisfatórios de convergência para o fronte de Pareto ideal e de dispersão de soluções, quanto por sua complexidade relativamente baixa de tempo, $O(mn^2)$, e espaço, $O(n^2)$, onde m é o número de objetivos e n é o tamanho da população utilizada.

O algoritmo opera da seguinte forma: inicialmente é gerada uma população inicial aleatória, de tamanho n, e, através dos operadores genéticos de *crossover* e

mutação, também é gerada uma população de filhos de mesmo tamanho. Em seguida, entra-se em processo iterativo por geração, ao final do qual a solução é encontrada. Neste processo, o primeiro passo é juntar a população principal da geração atual t, P_t, com a sua população de filhos, Q_t, formando $R_t = P_t + Q_t$. O algoritmo, então, executa a operação *fast-non-dominated*-sort em R_t, a qual ordena os indivíduos deste conjunto em frontes decrescentes, de acordo com o critério de não-dominação, ou seja, todas as soluções não-dominadas de R_t ficam no fronte 1, F_1, todas as soluções não-dominadas de $R_t - F_1$ ficam no fronte 2, F_2, e assim por diante. Posteriormente, os frontes passam pela operação de *crowding-distance-assingnment*, a qual atribui um valor de dispersão relativa para os indivíduos, e são unidos para gerar a população principal da próxima geração, P_{t+1}. Caso a união do último fronte necessário, F_i, gere um número de indivíduos maior que n em P_{t+1}, são selecionados apenas os $n - |P_{t+1}|$ indivíduos mais dispersos de F_i para a união com P_{t+1}. Após a população P_{t+1} estar completa, são utilizados os operadores genéticos de seleção e mutação para geração da população de filhos Q_{t+1}, terminando assim, uma iteração do processo iterativo por gerações. Ao final do processo iterativo, o algoritmo retorna, então, os indivíduos não-dominados encontrados, ou seja, os indivíduos do fronte F_1.

O NSGA-II tem se mostrado um algoritmo bastante robusto, porém, uma característica negativa do mesmo é a inabilidade da operação *crowding-distantce-assignment* de manter uma dispersão satisfatória de soluções. Visando sanar este problema, Zheng *et al.* (2012), desenvolveram um método para manutenção de diversidade denominado *Sphere Excluding Evolutionary Algorithm* (SEED).

O SEED altera a forma como os $n - |P_{t+1}|$ indivíduos do último fronte selecionado para junção com P_{t+1} são escolhidos. Inicialmente, os indivíduos do último fronte são ordenados de acordo com os seus valores extremos, definidos em relação as x funções objetivos destes.

$$Valor\ Extremo = Maximo\{|f_1|, \dots, |f_x|\}$$

Posteriormente, são selecionados os indivíduos com os maiores valores extremos para compor P_{t+1}, até que $|P_{t+1}| = n$. Neste processo seletivo, quando um indivíduo é selecionado, todos aqueles distantes de um raio r (um parâmetro do algoritmo) em relação a todos os objetivos são excluídos da lista de valores extremos e são alocadas em uma lista de indivíduos candidatos. Caso a lista de valores extremos fique vazia e ainda ocorra $|P_{t+1}| < n$, os indivíduos da lista de candidatos são alocados de acordo com a dispersão dos mesmos.

Outro trabalho que propõe melhoria na manutenção da dispersão de soluções no NSGA-II, denominada NSGA-II_R, foi desenvolvido por Fortin e Parizeau (2013). Neste trabalho os autores tratam a influência da ocorrência de indivíduos com valores idênticos nas funções de avaliação sobre a dispersão dos mesmos.

Segundo os autores, a ocorrência de indivíduos com soluções idênticas pode comprometer tanto a convergência quanto a dispersão do algoritmo. Para contornar este problema propõe-se a alteração da forma de como os indivíduos são escolhidos em três partes do algoritmo: a computação da dispersão, o método de seleção para *crossover* e a escolha dos $n - |P_{t+1}|$ indivíduos do último fronte para formação de P_{t+1}.

A alteração proposta consiste de, ao invés do método de escolha possuir como conjunto de possibilidades o conjunto de indivíduos envolvidos no processo, as

possibilidades se restringirem às soluções que estes representam, sendo que, caso haja indivíduos com a mesma solução, eles serão representados apenas uma vez no conjunto de possibilidades, e, caso tal solução seja escolhida, um dos indivíduos com esta solução é selecionado aleatoriamente.

Além das modificações relacionadas à melhoria da dispersão das soluções, também investiga-se aqui a importância da população inicial no resultado final obtido pelo algoritmo. Visando tal fim, é proposto a geração induzida de indivíduos com valores da função de custo em intervalos determinados, visando uma população inicial mais diversa. Este método é comparado com geração aleatória clássica de indivíduos, a qual é utilizada no NSGA-II original.

4. Resultados

Nesta seção é avaliado o desempenho do NSGA-II, bem como de algumas variantes do mesmo, considerando inclusive o impacto de dois métodos de inicialização da população, o aleatório e o intervalar. As diferentes combinações das variantes com os métodos de inicialização são analisadas em diferentes instâncias do NRP multi-objetivo de larga escala.

4.1. Configurações dos Experimentos

Os experimentos foram conduzidos usando-se as cinco instâncias clássicas apresentadas no trabalho de Xuan *et al.* (2012), denominadas nrp-1, nrp-2, nrp-3, nrp-4 e nrp-5. A tabela 2 apresenta o número de requisitos e de clientes de cada instância. Além da quantidade de requisitos e de clientes, as instâncias também variam em relação a outros fatores, como, por exemplo, os intervalos dos valores de custo dos requisitos e importância dos clientes. Mais informações sobre as instâncias podem ser encontradas em Xuan *et al.* (2012).

Tabela 2. Características das instâncias utilizadas

Instâncias	nrp-1	nrp-2	nrp-3	nrp-4	nrp-5
Qnt. de Requisitos	140	620	1500	3250	1500
Qnt. de Clientes	100	500	500	750	1000

A tabela 3 apresenta um sumário dos algoritmos utilizados, a saber, o NSGA-II sem modificações, e as duas variantes, SEED e NSGA-II$_R$. Para cada um destes algoritmos são utilizados dois métodos de inicialização: a geração aleatória (A), e a geração intervalar (I). Neste último método, os indivíduos da população são intencionalmente gerados visando que os valores de suas funções de custo estejam nos intervalos de 0-33%, 33-66% e 66-100% do custo total da instância.

Tabela 3. Algoritmos utilizados

Algoritmos	Variação	Inicialização
NSGA-II-A	-	Aleatória
NSGA-II-I	-	Intervalar
NSGA-II-SEED-A	SEED	Aleatória
NSGA-II-SEED-I	SEED	Intervalar
NSGA-II$_R$-A	NSGA-II$_R$	Aleatória
NSGA-II$_R$-I	NSGA-II$_R$	Intervalar

As configurações utilizadas foram semelhantes para todos os algoritmos e foram as mesmas para todas as instâncias. Adotou-se a codificação binária dos cromossomos, sendo utilizado o torneio binário como método de seleção, o *crossover* de corte de um ponto como método de *crossover* e a mutação aleatória como método de mutação. A chance de *crossover* foi de 0,8 e a taxa de mutação foi de $1/n$, onde $n = |R|$. O tamanho da população foi de 500 indivíduos e a quantidade de gerações foi de 400, totalizando 200.000 execuções da função de avaliação. A modificação proposta por Zheng *et al.* (2012) acrescenta um parâmetro a mais, o raio da esfera, no qual foi utilizado o valor 300.

Foram feitas 30 execuções de cada algoritmo em cada instância. A comparação dos resultados foi feita utilizando o hipervolume gerado pelos frontes de Pareto obtidos a cada execução. O hipervolume é uma métrica de maximização que analisa tanto a convergência quanto a diversidade das soluções encontradas [Beume *et al.* 2009]. Para identificar qual algoritmo obteve o melhor resultado foi utilizado o teste de Friedman [Derrac *et al.* 2011], o qual é um teste estatístico não paramétrico utilizado para a comparação de múltiplos conjuntos de dados. Nesse teste as observações entre os blocos podem ser classificadas. Em nossa análise ele foi utilizado com 5% de nível de significância. O pós-teste foi efetuado através de funções da ferramenta R [R Core Team 2012] e indica se há diferença estatística entre os diferentes blocos do conjuntos de dados.

4.2. Resultados e Discussão

A tabela 4 apresenta a média e o desvio padrão (em parênteses) dos valores de hipervolume obtidos por cada algoritmo, sendo que as melhores soluções, de acordo com o teste de Friedman, estão destacadas.

Tabela 4. Média e desvio padrão do hipervolume das 30 execuções de cada algoritmo para cada instância

Algoritmos	nrp-1	nrp-2	nrp-3	nrp-4	nrp-5
NSGA-II-A	1,48E+06 (8,95E+0)	3,82E+07 (2,36E+05)	7,31E+07 (9,74E+05)	2,39E+08 (3,44E+06)	6,61E+07 (7,45E+05)
NSGA-II-I	1,50E+06 (3,80E+0)	3,82E+07 (1,83E+05)	7,79E+07 (8,80E+05)	2,66E+08 (3,63E+06)	7,40E+07 (8,06E+05)
NSGA-II-SEED-A	1,51E+06 (7,00E+0)	3,91E+07 (1,61E+05)	7,42E+07 (1,12E+06)	2,53E+08 (5,81E+07)	6,65E+07 (7,84E+05)
NSGA-II-SEED-I	1,52E+06 (7,52E+0)	3,91E+07 (1,50E+05)	7,93E+07 (7,04E+05)	2,72E+08 (2,68E+06)	7,49E+07 (6,24E+05)
NSGA-II_R-A	1,49E+06 (1,31E+0)	3,90E+07 (1,57E+05)	7,33E+07 (8,96E+05)	2,50E+08 (5,65E+07)	6,56E+07 (7,92E+05)
NSGA-II_R-I	1,52E+06 (9,34E+0)	3,90E+07 (3,90E+07)	7,88E+07 (6,44E+05)	2,69E+08 (2,97E+06)	7,44E+07 (7,36E+05)

Pode-se observar que na instância nrp-1 as melhores soluções foram encontradas pelos algoritmos NSGA-II-SEED-I e NSGA-II_R-I. Esses algoritmos obtiveram um valor de hipervolume significativamente maior que os demais, de acordo com o teste de Friedman. Para o nrp-2, novamente, os algoritmos que utilizam diferentes métodos de dispersão obtiveram o melhor resultado, independentemente do método de inicialização. Nessas duas instâncias, o algoritmo NSGA-II puro (NSGA-II-A) não atingiu o melhor resultado.

Nas maiores instâncias, nrp-3, nrp-4 e nrp-5, as variantes NSGA-II-I, NSGA-II-SEED-I e NSGA-II_R-I apresentaram os melhores valores de hipervolume, de acordo com o teste de Friedman. Nessas instâncias de larga escala, diferentemente dos problemas anteriores, o método de inicialização intervalar se destacou e foi responsável pela geração dos melhores resultados, independente das variações dos métodos de dispersão. Novamente, o NSGA-II puro não obteve o melhor resultado em nenhuma instância estudada.

Tendo em vista os resultados, pode-se concluir que, dentre as modificações analisadas, a utilização do método intervalar de geração da população inicial foi o que apresentou maior impacto na qualidade das soluções, em especial nas instâncias maiores, sendo também o método com melhor escalabilidade. As variantes do NSGA-II relativas ao método de dispersão apresentam um desempenho superior ao do NSGA-II original, em especial nas instâncias menores. O algoritmo NSGA-II original não apresenta um bom comportamento se comparado aos demais, não chegando a se destacar em nenhuma das instâncias, sendo, então, desejável a utilização de uma modificação do mesmo para obtenção de melhores resultados.

5. Considerações Finais

Neste trabalho foi analisado o comportamento de variantes do algoritmo NSGA-II no contexto do NRP de larga escala. As variantes utilizadas visam promover uma maior dispersão dos indivíduos da população, com objetivo de melhorar a qualidade do conjunto de soluções finais. Também foi analisado o impacto da utilização de diferentes métodos de geração da população inicial.

Os resultados obtidos mostram que o método de inicialização da população é o fator que mais influencia a qualidade das soluções obtidas em instâncias de larga escala. Deve-se destacar que o NSGA-II puro apresentou o pior desempenho em relação aos demais algoritmos testados, sendo desejável a utilização de uma variação do mesmo para obtenção de melhores resultados.

Como trabalhos futuros, propõem-se o aprofundamento do estudo comparativo, com a realização de mais testes e a utilização de outras métricas, como o *spread*, e o teste de outros algoritmos multi-objetivo no contexto do NRP de larga escala, como a versão multi-objetivo do algoritmo PSO (*Partcile Swarm Optimization*).

Agradecimentos

Este trabalho foi parcialmente apoiado pelo Instituto Nacional de Ciência e Tecnologia para Engenharia de Software (INES), financiado pelo CNPq, processo 573964/2008-4.

Referências

Bagnall, A. J., Rayward-Smith, V. J. e Whittley, I. M. (2001). The Next Release Problem. *Information and Software Technology*, vol. 43.

Beume, N., Fonseca, C. M., Ibáñez, M. L., Paquete, L. e Vahrenhold J. (2009). On the Complexity of Computing the Hypervolume Indicator. *IEEE Transactions on Evolutionary Computation*, vol. 13, nº. 5.

Cai, X. e Wei, O. (2013). A Hybrid of Decomposition and Domination Based Evolutionary Algorithm for Multi-Objective Software Next Release Software. *10th IEEE International Conference on Control and Automation*.

Cormem, T. H., Leiserson, C. E., Rivest, R. L. e Stein C. (2009), *Introduction to Algorithms*, McGraw-Hill Book Company, 3rd edition.

Deb, K., Pratap, A., Agarwal, S. e Meyarivan, T. (2002). A Fast an Elitist MultiObjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, vol. 6, nº. 2.

Derrac, J., García, S., Molina, D. e Herrera, F. (2011). A Pratical Tutorial on the Use of Nonparametric Statistical Tests as a Methodology for Comparing Evolutionary and Swarm Intelligence Algorithms. *Swarm and Evolutionary Computation*, vol. 1, nº. 1.

Durillo, J. J., Alba, E., Zhang, Y. e Nebro, A. J. (2009). A Study of the Multi-Objective Next Release Problem. *1st International Symposium on Search Based Software Engineering*.

Fortin, F. e Parizeau, M. (2013). Revisiting the NSGA-II Crowding-Distance Computation. GECCO '13, Proceedings of the *15th Annual Conference on Genetic and Evolutionary Computation*.

R Core Team. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, 2012. ISBN 3-900051-07-0.

Ruhe, G. e Saliu, M. O. (2005). The Art and Science of Software Release Planning. *IEEE Software*, vol. 22, nº. 6.

Van den Akker, M., Brinkkemper, S., Diepen, G. e Versendaal, J. (2007). Software Product Release Planning Through Optimization and What-If Analysis. *Information and Software Technology*, vol. 50.

Xuan, J., Jiang, H., Ren, Z. e Luo, Z. (2012). Solving the Large Scale Next Release Problem with a Backbone-Based Multilevel Algorithm. *IEEE Transactions on Software Engineering*, vol. 38, nº. 5.

Zhang, Y., Harman, M. e Mansouri, S. A. (2007). The Multi-Objective Next Release Problem. GECCO '07, Proceedings of the *9th Annual Conference on Genetic and Evolutionary Computation*.

Zhang, Y. e Harman, M. (2010). Search Based Optimization of Requirements Interaction Management. *2nd International Symposium on Search Based Software Engineering*.

Zhang. Y., Harman, M., Finkelstein, A. e Mansouri, S. A. (2011). Comparing the Performance of Metaheuristics for the Analysis of Multi-Stakeholder Tradeoffs in Requirements Optimization. *Information and Software Technology*, vol. 53.

Zheng, J., Shen, R. e Zou, J. (2012). Enhancing Diversity for NSGA-II in Evolutionary Multi-Objective Optimization. *8th International Conference on Natural Computation*.

Multi-Objective Test Case Selection: A Hybrid Particle Swarm Optimization and Harmony Search Algorithm

Luciano S. de Souza[1,2], Ricardo B. C. Prudêncio[2], Flavia de A. Barros[2]

[1]Federal Institute of Education Science and Technology of the North of Minas Gerais
IFNMG – Pirapora (MG), Brazil

[2]Center of Informatics (CIn)
Federal University of Pernambuco (UFPE)
Recife (PE), Brazil

luciano.souza@ifnmg.edu.br, {rbcp,fab}@cin.ufpe.br

Abstract. *Automatic Test Case (TC) selection is an important tool to enhance the quality of the software programs. TC selection can be treated as an optimization problem, aiming to find a subset of TCs which optimizes one or more objective functions (i.e., selection criteria). In our work, we developed a new hybrid strategy for TC selection which consider two objectives simultaneously: maximize branch coverage while minimizing execution cost (time). This hybrid strategy was implemented using a multi-objective techniques based on Particle Swarm Optimization (PSO) and Harmony Search (HS) algorithm. The experiments were performed on the space and flex programs from the SIR repository, attesting the feasibility of the proposed hybrid strategy.*

1. Introduction

Almost everything we use today has an element of software in it. There are several situations where software failure is simply unacceptable. Hence, an organization that develops any form of software must put in every effort to drastically reduce, and preferably, eliminate any defects [Desikan and Ramesh 2006]. However, testing activities are very expensive. In this scenario, automation seems to be the key solution for improving the efficiency and effectiveness of the testing process as well as reduce the costs.

The related literature presents two main approaches for testing: White Box (structural) or Black Box (functional) testing. Structural testing investigates the behavior of the software directly accessing its code. Functional testing, in turn, investigates whether the software functionalities are responding/behaving as expected without using any knowledge about the code [Young and Pezze 2005]. In both approaches, the testing process relies on the (manual or automatic) generation and execution of a Test Suite (TS), that is, a set of Test Cases. A Test Case (TC), consists of "a set of inputs, execution conditions, and a pass/fail conditions" [Young and Pezze 2005]. The execution of the TS aims to provide a suitable coverage of an adopted test adequacy criterion (e.g., code coverage, functional requirements coverage) in order to satisfy the test goals. In this scenario, both automatic generated or manually created TS may be large. In general, they try to cover all possible test scenarios in order to accomplish a good coverage of the adequacy criterion. Although it is desirable to fully satisfy the test goals, the execution of large suites is a very expensive task, demanding a great deal of the available resources (time and people) [Harold et al. 1993].

It is possible to identify in large test suites two or more TCs covering the same requirement/piece of code (i.e., they are redundant concerning that requirement/piece of code). Thus, it is possible to reduce the suite in order to fit the available resources. This task of reducing a test suite based on a given selection criterion is known as *Test Case selection*.

When performing TC Selection there may be several TC combinations to consider. Hence, a promising way to deal with this problem is by using search based optimization techniques. These techniques aim to search for a subset of TCs which optimizes a given function (i.e., the given selection criterion).

Regarding search based Test Case Selection we can cite in literature both single objective and multi-objective approaches (see [Mansour and El-Fakih 1999, Ma et al. 2005, Yoo and Harman 2007, Yoo and Harman 2010, de Souza et al. 2010, de Souza et al. 2011, de Souza et al. 2013, Yoo et al. 2011b, Yoo et al. 2011a]). Since, in general, there are more than one test goal to consider during the testing process, the use of multi-objective search based techniques is encouraged. For example, in [de Souza et al. 2011] (our previous work) we investigated the multi-objective TC selection considering both the functional requirements coverage (quality) and the execution effort (cost) of the selected subset of TCs as objectives of the selection process.

Despite of the good results obtained in [de Souza et al. 2011] on a case study, further improvements could be performed. In this work, we developed a new hybrid algorithm (the BMOPSO-CDRHS) by merging: (1) the BMOPSO-CDR algorithm and, (2) the Harmony Search algorithm. Harmony Search (HS) algorithm [Geem et al. 2001] has drawn more and more attention from search based community due to its excellent characteristics such as easy implementation and good optimization ability.

In order to verify the performance of the proposed algorithm we performed experiments using two programs from the Software-artifact Infrastructure Repository (SIR) programs [Do et al. 2005], and compared the results with three other algorithms: (1) the former version (BMOPSO-CDR); (2) the Non-Dominated Sorting Genetic Algorithm (NSGA-II) [Deb et al. 2000]; and the Multi-Objective Binary Harmony Search Algorithm (MBHS) [Wang et al. 2011]. Each implemented algorithm was applied to the multi-objective TC selection problem, providing to the user a set of solutions (test suites) with different values for the objective functions: code's branch coverage and execution cost. The user may then choose the solution that best fits the available resources.

Finally, it is important to highlight that, the literature lacks works applying PSO techniques into the multi-objective TC selection problem (besides the works of the current authors). Furthermore, to the best of our knowledge, the HS algorithm was not yet investigated in the context of Test Case Selection. Hence, this is an promising study area, and this work aims to explore it a little further by creating a new hybrid multi-objective strategy for test case selection.

2. Multi-Objective Test Case Selection by Using the Search Based PSO Algorithm

In the current work, we propose the use of the search based Particle Swarm Optimization algorithm (PSO) to solve the multi-objective TC selection problem. In contrast to single-

objective problems, Multi-Objective Optimization (MOO) aims to optimize more than one objective at the same time.

An MOO problem considers a set of k objective functions $f_1(x), f_2(x), ..., f_k(x)$ where x is an individual solution for the problem being solved. The output of an MOO algorithm is usually a population of *non-dominated* solutions considering the objective functions. Formally, let x and x' be two different solutions. We say that x dominates x' (denoted by $x \preceq x'$) if x is better than x' for at least one objective function and x is not worse than x' for any objective function. x is said to be *not dominated* if there is no other solution x_i in the current population, such that $x_i \preceq x$. The set of non-dominated solutions in the objective space returned by an MOO algorithm is known as Pareto frontier [Coello et al. 2007].

The PSO algorithm is a population-based search approach, inspired by the behavior of birds' flocks [Kennedy and Eberhart 1995] and has shown to be a simple and efficient algorithm compared to other search techniques, including for instance the widespread Genetic Algorithms [Eberhart and Shi 1998]. The basic PSO algorithm starts its search process with a random population (also called swarm) of *particles*. Each particle represents a candidate solution for the problem being solved and it has four main attributes:

1. the position ($\mathbf{t}$) in the search space (each position represents an individual solution for the optimization problem);
2. the current velocity ($\mathbf{v}$), indicating a direction of movement in the search space;
3. the best position ($\hat{\mathbf{t}}$) found by the particle (the memory of the particle);
4. the best position ($\hat{\mathbf{g}}$) found by the particle's neighborhood (the social guide of the particle).

For a number of iterations, the particles fly through the search space, being influenced by their own experience $\hat{\mathbf{t}}$ and by the experience of their neighbors $\hat{\mathbf{g}}$. Particles change position and velocity continuously, aiming to reach better positions and to improve the considered objective functions. In this work, the particle's positions were defined as binary vectors representing candidate subsets of TCs to be applied in the software testing process. Let $T = \{T_1, \ldots, T_n\}$ be a test suite with n test cases. A particle's position is defined as $\mathbf{t} = (t_1, \ldots, t_n)$, in which $t_j \in \{0, 1\}$ indicates the presence (1) or absence (0) of the test case T_j within the subset of selected TCs.

As said, two objective functions were adopted: branch coverage and execution cost. The branch coverage (function to be maximized) consists of the ratio (in percentage) between the amount of code branches covered by a solution $\mathbf{t}$ and the amount of branches covered by T. Formally, let $B = \{B_1, \ldots, B_k\}$ be a given set of k branches covered by the original suite T. Let $F(T_j)$ be a function that returns the subset of branches in B covered by the individual test case T_j. The branch coverage of a solution $\mathbf{t}$ is given by:

$$B\_Coverage(\mathbf{t}) = 100 * \frac{|\bigcup_{t_j=1}\{F(T_j)\}|}{k} \quad (1)$$

In eq. (1), $\bigcup_{t_j=1}\{F(T_j)\}$ is the union of branches subsets covered by the selected test cases (i.e., T_j for which $t_j = 1$).

The execution cost (function to be minimized) represents the amount of time required to execute the selected suite. Formally, each test case $T_j \in T$ has a *cost score* c_j. The total cost of a solution **t** is given by:

$$Cost(\mathbf{t}) = \sum_{t_j=1} c_j \tag{2}$$

It is not the purpose of this work to enter into a discussion concerning which objectives are more important for the TC selection problem. Irrespective of arguments about their suitability, branch coverage is a likely candidate for assessing quality of a TS, and execution time is one realistic measure of cost.

Finally, the proposed algorithms deliver a Pareto frontier regarding the objective functions $B\_Coverage$ and $Cost$. A Pareto frontier is the set of non-dominated solutions in the objective space returned by the algorithm [Coello et al. 2007].

2.1. The BMOPSO-CDR algorithm

The Binary Multi-objective Particle Swarm Optimization with Crowding Distance and Roulette Wheel (BMOPSO-CDR) was firstly presented in [de Souza et al. 2011]. It uses an External Archive (EA) to store the non-dominated solutions found by the particles during the search process. It is summarized as follows:

1. Randomly initialize the swarm, evaluate each particle according to the considered objective functions and then store in the EA the particles' positions that are non-dominated solutions;
2. WHILE stop criterion is not verified DO
   (a) Compute the velocity **v** of each particle as:

$$\mathbf{v} \leftarrow \omega\mathbf{v} + C_1 r_1(\hat{\mathbf{t}} - \mathbf{t}) + C_2 r_2(\hat{\mathbf{g}} - \mathbf{t}) \tag{3}$$

   where ω is the inertia factor; r_1 and r_2 are random values in [0,1]; C_1 and C_2 are constants. The social guide ($\hat{\mathbf{g}}$) is defined as one of the non-dominated solutions stored in the EA, selected by Roulette Wheel.
   (b) Compute the new position **t** of each particle for each dimension t_j as:

$$t_j = \begin{cases} 1, \text{ if } r_3 \leq sig(v_j) \\ 0, \text{ otherwise} \end{cases} \tag{4}$$

   where r_3 is a random number in [0,1] and sig(v_j) is defined as:

$$sig(v_j) = \frac{1}{1 + e^{-v_j}} \tag{5}$$

   (c) Use the mutation operator as proposed by [Coello et al. 2004];
   (d) Evaluate each particle and update the solutions stored in the EA;
   (e) Update the particle's memory $\hat{\mathbf{t}}$;
3. END WHILE and return the current EA as the Pareto frontier.

2.2. The BCMOPSO-CDRHS algorithm

The Harmony Search algorithm (see [Geem et al. 2001]) is inspired by the the musical process of searching for a perfect harmony. It imitates the musician seeking to find pleasing harmony determined by an aesthetic standard, just as the optimization process seeks to find a global optimal solution determined by an objective function [Wang et al. 2010]. The harmony in music is analogous to the optimization solution vector, and the musician's improvisations are analogous to local and global search schemes in optimization techniques [Afkhami et al. 2013].

Basically, the HS algorithm starts by creating random harmonies (solutions) and storing them into a matrix called Harmony Memory (HM). The HM is used, during all the optimization process, to store the best harmonies found by the algorithm. After the initialization of the HM, the improvisation begins and it is controlled by three operators[1]: (1) Harmony Memory Considering Operator (HMCO), (2) Pitch Adjusting Operator (PAO), and (3) Random Selection Operator (RSO). (following the nomenclature of [Wang et al. 2011]).

The HMCO and RSO are controlled by the Harmony Memory Considering Rate (HMCR), which is a value between 0 and 1 that balances the exploration and exploitation when performing the improvisation. In turn, the Pitch Adjustment Rate (PAR) controls the pitch adjusted (analogous to a local search mechanism). At the end of the improvisation, if the new harmony obtained after applying the operators is better than the worst harmony in the HM, it will be stored into the HM and the worst harmony is removed. This process continues until a stop criterion is reached.

In order to create the hybrid BCMOPSO-CDRHS, we adapted the Discrete Harmony Search algorithm from [Wang et al. 2010] using the individual selection strategy and the parallel update strategy (see [Wang et al. 2010] for more details). In our work, the HM corresponds to the EA and, since there is no need to initialize the HM (as the HS will be performed at each iteration after the PSO), we introduced the HS improvisation process after the step (e) of the main loop (2) of BMOPSO-CDR as:

1. Create NGC harmonies[2] (solutions) **t**.
2. For EACH dimension of each harmony DO

$$t_j = \begin{cases} t_{kj}, \text{ if } r_1 \leq HMCR \\ RSO, \text{ otherwise} \end{cases} \tag{6}$$

$$RSO = \begin{cases} 1, \text{ if } r_2 \leq 0.5 \\ 0, \text{ otherwise} \end{cases} \tag{7}$$

where t_j is j-th element of the new harmony; r_1 and r_2 are random values in the interval [0,1]; $k \in \{1, 2, \ldots, HMS\}$ is chosen randomly.

   (a) If the element of the new harmony came from HM (i.e., if $r_1 \leq HMCR$) then

$$t_j = \begin{cases} G_j, \text{ if } r_3 \leq PAR \\ t_j, \text{ otherwise} \end{cases} \tag{8}$$

[1]Using the nomenclature presented in [Wang et al. 2011]

[2]NGC (new generating candidate) is the number of harmonies that will be created before uppdate the HM. This corresponds to the parallel update strategy (see [Wang et al. 2010]).

where r_3 is a random values in the interval [0,1]; G_j is the j-th element of the best solution stored in HM. Since HM corresponds to EA, we do not have only a best single solution. Hence, we use the Roulette Wheel in order to select **G** that will be the same used for all candidate harmonies.

3. Update the HM (EA) by adding the non-dominated created harmonies and by removing the dominated solutions from HM.
   This overall improvisation process is repeated 20 times[3].

3. Experiments and Results

This section presents the experiments performed in order to evaluate the search algorithms implemented in this work. In addition to the aforementioned algorithms, we also implemented the well known NSGA-II algorithm [Deb et al. 2000], and the only (to the best of our knowledge) proposed Multi-Objective Binary Harmony Search (MBHS) algorithm [Wang et al. 2011]. These algorithms were implemented in order to compare whether the our proposed algorithms are competitive as multi-objective search based optimization techniques. The experiments were performed using the *space* and *flex* programs from the Software-artifact Infrastructure Repository (SIR) [Do et al. 2005]. The *space* program, from the European Space Agency, is an interpreter for an array definition language (ADL), containing 15,297 lines of code. The *space* program has several test suites, hence we choose one of the suites with most code coverage (with 567 test cases). In turn, the *flex* program is lexical analyzer utility from Unix, with 6,199 and a TS containing 160 test cases (we choose the largest suite in this program).

For each TC, since there were no information about the execution cost of the TCs, we computed the execution cost information by using the Valgrind profiling tool [Nethercote and Seward 2003]. The Valgrind executes the program binary code in an emulated virtual CPU. The cost of each test case was measured by counting the number of virtual instruction codes executed by the emulated environment. These counts allows to argue that they are directly proportional to the cost of the TC execution. Additionally, for the same reasons, we computed the branch coverage information by using the profiling tool *gcov* from the GNU compiler *gcc*.

3.1. Metrics

In our experiments, we evaluated the Pareto frontiers obtained by the algorithms, for each test suite, according to three different metrics usually adopted in the literature:

- Hypervolume (HV): it computes the size of the dominated space, which is also called the *area under the curve*. A high value of hypervolume is desired in MOO problems;
- Generational Distance (GD): it computes how far, on average, one Pareto set (called PF_{known}) is from the true Pareto set (called as PF_{true});
- Coverage (C): it indicates the amount of the solutions within the non-dominated set of the first algorithm which dominates the solutions within the non-dominated set of the second algorithm.

[3]This value was found by trial and error and further formal investigation will be performed in order to verify its influence.

The GD metric requires that the PF_{true} be known. For more complex problems (with bigger search spaces), as the *space* and *flex* programs, it is impossible to know PF_{true} a priori. In these cases, a reference Pareto frontier (called here $PF_{reference}$) can be constructed and used to compare algorithms regarding the Pareto frontiers they produce [Yoo and Harman 2010]. The reference frontier represents the union of all found Pareto frontiers, resulting in a better set of non-dominated solutions. Additionally, the C metric reported in this work refers to the coverage of the optimal set $PF_{reference}$, over each algorithm, indicating the amount of solutions of those algorithms that are dominated, e. g. that are not optimal.

3.2. Algorithms Settings

All the algorithms were run for a total of 200,000 objective function evaluations. The BMOPSO-CDR and the hybrid BMOPSO-CDRHS algorithms used 20 particles, mutation rate of 0.5, ω linearly decreases from 0.9 to 0.4, constants C_1 and C_2 1.49, maximum velocity of 4.0 and EA's size of 200 solutions. These values are the same used in [de Souza et al. 2011] and represent generally used values in the literature. Regarding BMOPSO-CDRHS and MBHS algorithms, we used HMCR of 0.9 and PAR of 0.03 (as suggested in [Wang et al. 2011]). In order to perform a more fair comparison we added the parallel update strategy into the MBHS, and for both BMOPSO-CDRHS and MBHS algorithms we used a NGC of 20 harmonies.

The NSGA-II algorithm, in turn, used a mutation rate of 1 / population size, crossover rate of 0.9 and population size of 200 individuals. As the NSGA-II and MBHS algorithms does not use an external archive to store solutions, we decided to use the population size/HMS of 200 solutions to permit a fair comparison. This way, all the algorithms are limited to a maximum of 200 non-dominated solutions.

3.3. Results

After 30 executions of each TC selection algorithm, the values of branch coverage and execution cost observed in the Pareto frontiers were normalized since they are measured using different scales. All the evaluation metric were computed and statistically compared using the Wilcoxon rank-sum test with an α level of 0.95. The Wilcoxon rank-sum test is a nonparametric hypothesis test that does not require any assumption on the parametric distribution of the samples.

In the context of this paper, the null hypothesis states that, regarding the observed metric, two different algorithms produce equivalent Pareto frontiers. The α level was set to 0.95, and significant *p-values* suggest that the null hypothesis should be rejected in favor of the alternative hypothesis, which states that the Paretor frontiers are different. Table 1 presents the average values of the adopted metrics, as well as the observed standard deviations. Additionally, asterisks (*) were used to indicate whenever two values are considered statistically equal (i.e. the null hypothesis was not rejected).

From Table 1, we can observe that the BMOPSO-CDRHS outperformed the other selection algorithms for all the collected metrics. It is possible to observe, from the HV metric, that the BMOPSO-CDRHS dominates bigger objective space areas when compared to the others. Furthermore, the GD values obtained by the algorithm shows that its Pareto frontiers have better convergence to the optimal Pareto frontier (represented by

Table 1. Mean value and standard deviation for the metrics

Metric	Suite	BMOPSO-CDR	BMOPSO-CDRHS	NSGA-II	MBHS
HV	space	0.804 (0.008)	0.949 (0.003)	0.872 (0.011)	0.947 (0.002)
	flex	0.734 (0.003)	0.847 (0.003)	0.820 (0.012)	0.832 (0.003)
GD	space	0.020 (0.004)	8.4E-4 (1.6E-4)	0.002 (8.2E-4)	0.001 (2.4E-4)
	flex	0.022 (0.002)	0.001 (2.9E-4)	0.002 (7.7E-4)	0.004 (4.5E-4)
C	space	1.0* (0.0)	0.966 (0.037)	0.994* (0.029)	0.997* (0.007)
	flex	1.0* (0.0)	0.969 (0.064)	0.996* (0.015)	1.0* (0.0)

the $PF_{reference}$). Finally, the Coverage metric indicates that the BMOPSO-CDRHS algorithm was the least dominated algorithm by the optimal Pareto set, hence several of its solutions are within the optimal frontier, and was the only one different from the others.

In addition to aforementioned results, we also state that the hybrid mechanism indeed improved the BMOPSO-CDR algorithm, and that the BMOPSO-CDRHS selection algorithm is a competitive multi-objective algorithm.

4. Conclusion

In this work, we propose the creation of an hybrid algorithm by adding the Harmony Search algorithm into the binary multi-objective PSO for structural TC selection. The main contribution of the current work is to investigate whether this hybridization can improve the multi-objective PSO (BMOPSO-CDR), from [de Souza et al. 2011], for selecting structural test cases considering both branch coverage and execution cost. We highlight that hybrid binary multi-objective PSO with Harmony Search was not yet investigated in the context of TC selection. Besides, the developed selection algorithms can be adapted to other test selection criteria and are not limited to two objective functions. Furthermore, we expect that the good results can also be obtained on other application domains.

The hybrid algorithm (BMOPSO-CDRHS) outperformed the BMOPSO-CDR, MBHS and NSGA-II for all metrics. Hence, we can conclude that by creating the hybrid the former algorithm was improved and the hybrid algorithm is a competitive multi-objective search strategy.

As future work, we can point the investigation of other hybrid strategies, and perform the same experiments on a higher number of programs in order to verify whether the obtained results are equivalent to those presented here, and also whether these results can be extrapolated to other testing scenarios other than from the SIR repository. Also, we will investigate whether the performance of PSO, on TC selection problem, is impacted by changing its parameters.

Acknowledgment

This work was partially supported by the National Institute of Science and Technology for Software Engineering (INES www.ines.org.br), CNPq, CAPES, FACEPE and FAPEMIG.

References

Afkhami, S., Ma'rouzi, O. R., and Soleimani, A. (2013). A binary harmony search algorithm for solving the maximum clique problem. *International Journal of Computer Applications*, 69.

Coello, C., Pulido, G., and Lechuga, M. (2004). Handling multiple objectives with particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, 8(3):256–279.

Coello, C. A. C., Lamont, G. B., and van Veldhuizen, D. A. (2007). *Evolutionary Algorithms for Solving Multi-Objective Problems*, volume 5. Springer.

de Souza, L. S., Miranda, P. B. C., Prudêncio, R. B. C., and Barros, F. d. A. (2011). A multi-objective particle swarm optimization for test case selection based on functional requirements coverage and execution effort. In *In Proc. of the 23rd Int. Conf. on Tools with AI (ICTAI 2011)*.

de Souza, L. S., Prudêncio, R. B., de A. Barros, F., and da S. Aranha, E. H. (2013). Search based constrained test case selection using execution effort. *Expert Syst. with App.*, 40(12):4887 – 4896.

de Souza, L. S., Prudêncio, R. B. C., and Barros, F. d. A. (2010). A constrained particle swarm optimization approach for test case selection. In *In Proc. of the 22nd Int. Conf. on SW Eng. and Knowledge Eng.*

Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T. (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. In *Parallel Problem Solving from Nature PPSN VI*, volume 1917 of *LNCS*.

Desikan, S. and Ramesh, G. (2006). *Software Testing: Principles and Practice*. Pearson Education Canada.

Do, H., Elbaum, S., and Rothermel, G. (2005). Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. *Empirical Softw. Engg.*, 10(4):405–435.

Eberhart, R. C. and Shi, Y. (1998). Comparison between genetic algorithms and particle swarm optimization. *LNCS*, 1447:611–616.

Geem, Z. W., Kim, J. H., and Loganathan, G. (2001). A new heuristic optimization algorithm: harmony search. *Simulation*, 76(2):60–68.

Harold, M. J., Gupta, R., and Soffa, M. L. (1993). A methodology for controlling the size of a test suite. *ACM Trans. Softw. Eng. Methodol.*, 2(3):270–285.

Kennedy, J. and Eberhart, R. C. (1995). Particle swarm optimization. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, pages 1942–1948.

Ma, X.-Y., Sheng, B.-K., and Ye, C.-Q. (2005). Test-suite reduction using genetic algorithm. *Lecture Notes in Computer Science*, 3756:253–262.

Mansour, N. and El-Fakih, K. (1999). Simulated annealing and genetic algorithms for optimal regression testing. *Journal of Software Maintenance*, 11(1):19–34.

Nethercote, N. and Seward, J. (2003). Valgrind: A program supervision framework. In *In Third Workshop on Runtime Verification (RVï¿½03)*.

Wang, L., Mao, Y., Niu, Q., and Fei, M. (2011). A multi-objective binary harmony search algorithm. In *Advances in Swarm Intelligence*, pages 74–81. Springer.

Wang, L., Xu, Y., Mao, Y., and Fei, M. (2010). A discrete harmony search algorithm. In *Life System Modeling and Intelligent Computing*, pages 37–43. Springer.

Yoo, S. and Harman, M. (2007). Pareto efficient multi-objective test case selection. In *Proceedings of the 2007 Int. Symp. on SW Test. and Analysis*, pages 140–150.

Yoo, S. and Harman, M. (2010). Using hybrid algorithm for pareto efficient multi-objective test suite minimisation. *J. Syst. Softw.*, 83:689–701.

Yoo, S., Harman, M., and Ur, S. (2011a). Highly scalable multi objective test suite minimisation using graphics cards. In *Proceedings of the Third international conference on Search based software engineering*, SSBSE'11, pages 219–236, Berlin, Heidelberg. Springer-Verlag.

Yoo, S., Nilsson, R., and Harman, M. (2011b). Faster fault finding at google using multi objective regression test optimisation. In *8th European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 11), Szeged, Hungary*.

Young, M. and Pezze, M. (2005). *Software Testing and Analysis: Process, Principles and Techniques*. John Wiley & Sons.

SBSTFrame: uma proposta de framework para o teste de software baseado em busca

Bruno N. Machado, Andre A. Lôbo de Oliveira, Beatriz P. Martins, Celso G. Camilo-Junior, Cassio L. Rodrigues, Plínio de Sá Leitão Júnior, Auri M. R. Vincenzi

[1]Instituto de Informática – Universidade Federal Goiás (UFG)
Caixa Postal 131 - CEP 74001-970 – Goiânia – GO – Brasil

{bruno.nunes,andreoliveira,beatrizmartins,celso,cassio,plinio,auri}@inf.ufg.br

Resumo. *O teste de software é um importante componente no ciclo de vida de desenvolvimento de software, implicando diretamente na alta qualidade do produto de software. Alguns problemas detectados durante a fase de teste de software não são possíveis de serem otimizados apenas com as técnicas tradicionais da Engenharia de Software. Contudo, é possível realizar a modelagem matemática desses problemas e tentar otimizá-los por meio das técnicas de busca. Assim, o presente trabalho objetiva o desenvolvimento de um framework capaz de facilitar a aplicação de técnicas de busca em problemas da área de Teste de Software. As atuais funcionalidades do framework são apresentadas em um estudo de caso. Para uma análise inicial dos benefícios, analisou-se o esforço de programação dado pela quantidade de linhas de código reduzida. Ao final, percebe-se uma praticidade, facilidade e agilidade da proposta que resulta em uma redução de custo para o usuário.*

Abstract. *The software testing is an important component of lifecycle of the software development, that directly implies in the high quality of software product. Some detected problems in the software testing phase are not possible to be optimized only with the tradition techniques of the Software Engineering. By the way, it is possible to do the mathematical modelling of these problems and try to optimize them through the search techniques. Thus, the present study aims to develop a framework capable to facilitate the application of the search techniques in problems of the Testing Software. The actual functionalities of the framework are demonstrated in a case study. For an analysis of initial befefits, analyzed the programing effort given by quantify lines code reduced. The end, there is a perceived practicality, facility and agility of the poposal that results in a lower cost to user.*

1. Introdução

O teste de software é um importante componente no ciclo de vida de desenvolvimento de software, implicando diretamente na alta qualidade do produto de software [Tayamanon et al. 2011]. Testar um software consiste em executá-lo na tentativa de revelar o maior número possível de defeitos [Myers and Sandler 2004], com o objetivo de aumentar a qualidade do produto em questão. Alguns problemas detectados durante a fase de teste de software não são possíveis de serem resolvidos apenas com as técnicas tradicionais da Engenharia de Software. Contudo, é possível realizar a modelagem matemática desses problemas e resolvê-los por meio da otimização matemática, apoiado pelo

uso de metaheurísticas [Freitas and Maia 2010]. Nesse contexto, um campo denominado *Search-based Software Engineering* (SBSE) trata da resolução de problemas de Engenharia de software por meio de técnicas de otimização. Devido a quantidade de problemas possíveis de serem modelados e resolvidos por meio de técnicas de otimização na área de teste de software, surgiu a *Search-based Software Testing* (SBST), como um subcampo da SBSE.

Há uma grande quantidade de ferramentas que apoiam a aplicação de técnicas de teste de software tradicionais. Por outro lado, também há diversos *frameworks* que apoiam a implementação de técnicas de busca. Entretanto, são poucas as ferramentas disponíveis para aplicação de abordagens especializadas em SBST. Por isso, o presente trabalho propõe o desenvolvimento de um framework, chamado SBSTFrame, capaz de integrar esses dois tipos de ferramentas, facilitando assim, a aplicação de técnicas de busca em problemas do Teste de Software.

O *framework* possui três elementos principais, são eles: problema, solução e resultados. Cada elemento, possui seu conjunto de parâmetros a serem selecionados e/ou definidos pelo usuário. A configuração dos três compõe o experimento a ser executado e analisado. Alguns níveis e critérios de teste, bem como algumas técnicas de busca já são abrangidas pelo *framework*, e são demonstradas em um estudo de caso, seção 4.

A respeito dos resultados, observou-se a baixa dependência de recursos de software, a facilidade no uso do *framework* proposto para configuração de um experimento (problema, solução e resultados). Destaca-se também, a reutilização de recursos já existentes, como ferramentas de teste de software e *frameworks* que apoiam a implementação de técnicas de busca. Além disso, a grande quantidade de código que é abstraída perante o usuário, e a quantidade de operações que são realizadas dentro do *framework* de forma transparente para o usuário.

Este trabalho está organizado da seguinte forma: seção 2 apresenta a Search-Based Software Testing (SBST), destacando alguns dos seus problemas de otimização bem como algumas ferramentas que apoiam o seu emprego. Na seção 3 a proposta é apresentada. Já seção 4 traz um estudo de caso ilustrando o uso do *framework* proposto e a seção 5 descreve uma análise da abordagem proposta. Por fim a seção 6 discorre sobre as conclusões do trabaho.

2. Search-Based Software Testing (SBST)

A Engenharia de Software, como uma disciplina de engenharia, é uma área na qual existem problemas que podem ser resolvidos matematicamente [Harman 2006]. Nesse contexto, surgiu um campo denominado *Search-based Software Engineering* (SBSE) que trata da resolução de problemas de Engenharia de Software por meio de técnicas de otimização matemática. Tais problemas podem ser caracterizados pela busca da solução em um espaço com elevada quantidade de possibilidades e alta complexidade para o alcance das soluções.

O Teste de Software consiste em uma das ”áreas chave”da Engenharia de Software [IEEE Computer Society 2014]. Existem muitos problemas presentes nas etapas do Teste de Software possíveis de serem modelados e resolvidos com técnicas de otimização matemática. Dada a grande quantidade e representatividade desses problemas, surgiu a *Search-Based Software Testing* (SBST) como uma subárea da SBSE

[Freitas and Maia 2010]. Os primeiros trabalhos nessa área surgiram em 1976, porém apenas em 1990 essa linha passou a ganhar força, e mais recentemente, uma maior gama de quantidade de trabalhos tem sido publicados em SBST [McMinn 2011]. Desde então, da mesma forma que a SBSE, as pesquisas em SBST complementam as técnicas convencionais da Engenharia de Software já utilizadas no desenvolvimento de Sistemas em problemas cujas soluções são obtidas de forma não determinísticas.

As metaheurísticas são técnicas de otimização frequentemente empregadas no campo da SBST. Pode-se citar as técnicas de busca locais e globais: a) *Hill-Climbing* [Blum and Roli 2003]; b) *Simulated Annealing* [Kirkpatrick et al. 1983]; c) *Greedy Randomized Adaptive Search Procedure* (GRASP)[BoussaïD et al. 2013]; e; d) Algoritmos Genéticos [Holland 1992].

2.1. Problemas de Otimização no Teste de Software

Considerando a grande quantidade de informações possíveis para realização do teste nas suas diferentes fases, níveis e critérios, por exemplo, o grande domínio de entrada de um programa; boa parte das áreas do Teste de Software são passíveis de otimização. Todavia, apenas a aplicação de técnicas de otimização nessas áreas do Teste de Software não resolve os problemas, é necessário que essas técnicas sejam aprimoradas.

Por isso, encontrar soluções de qualidade nessas áreas de teste em tempos praticáveis, não tem sido uma tarefa fácil, uma vez que lidam, na maioria das vezes, com um escopo de problemas que podem ser reduzidos à classe dos problemas NP-completos. Pode-se citar, os problemas de geração, seleção, minimização e priorização de casos de teste. Como exemplo, apresenta-se a formulação matemática para os problemas de seleção e geração, dada por [Yoo and Harman 2012] e [Korel 1990], respectivamente:

1. Seleção do Conjunto de Teste
   - Dado: o programa, P, a versão modificada de P, P' e o conjunto de teste T.
   - Problema: encontrar um subconjunto $T' \in T$ que teste P.
2. Geração de dados de teste
   - Dado: $P =< n_{k1}, ..., n_{kn} >$, o caminho de um programa e o domínio de entrada D de P.
   - Problema: encontrar uma entrada x para o programa P tal que $x \in D$ para que P seja satisfeito em seus diferente caminhos.

Assim, diversas atividades de teste tem sido alvo da aplicação das técnicas de SBST [McMinn 2011], merecendo citar: a) *Teste Temporal* [Wegener et al. 1997, Wegener and Grochtmann 1998]; b) *Teste Funcional* [Bühler and Wegener 2008]; c) Teste Estrutural [Miller and Spooner 1976, Tracey et al. 1998]. Outras pesquisas tem obtido bons resultados com a aplicação de abordagens evolucionárias, destacando-se os Algoritmos Genéticos (AGs) [Yoo and Harman 2007, Hemmati and Briand 2010, Farzat 2010]. Todavia, ainda existem muitos desafios a serem superados e o desenvolvimento de novas técnicas de SBST, bem como o aprimoramento das existentes, podem colaborar para automatização de muitas atividades de teste e assim alcançar soluções de qualidade e em um tempo factível.

2.2. Ferramentas de SBST

O uso de ferramentas que apoiem as atividades do processo de teste tem sido um importante aliado para avaliação dos produtos de software, contribuindo no ganho de tempo,

produtividade, confiabilidade e qualidade em cada etapa do ciclo de vida do teste. A adoção dessas ferramentas, em conjunto com as técnicas de automação de testes podem aumentar consideravelmente o número de defeitos encontrados, acelerar a execução dos testes, diminuir o esforço nas atividades de teste e aumentar a qualidade do produto de software. O JUnit, a Proteum, a MuJava, o Selenium, são exemplos de ferramentas de teste, que dão apoio a automação do teste em diversas fases, níveis e critérios de teste.

Como discutido na seção 2 muitos problemas presentes nas etapas do Teste de Software não são possíveis de serem resolvidos com as técnicas tradicionais, porém são passíveis de serem modelados e resolvidos com técnicas de otimização matemática. Assim, surgiu a necessidade de ferramentas especializadas em abordagens que fazem uso de técnicas de otimização matemática, como as metaheurísticas. Há poucas ferramentas voltadas para esse tipo de abordagem, dentre elas, serão discutidas aqui as ferramentas *EvoSuite* e *EvoTest*.

Em [Fraser and Arcuri 2011] é apresentada a ferramenta EvoSuite, uma ferramenta que gera casos de teste de forma automática para classes escritas na linguagem Java. O EvoSuite aplica uma abordagem hibrida capaz de gerar e otimizar conjuntos de casos de teste a fim de satisfazer um critério de cobertura. Para produção de conjuntos de casos de teste, EvoSuite sugere possíveis oráculos pela adição de conjuntos pequenos e eficientes de declarações que concisamente resumem o comportamento atual; essas declarações permitem ao desenvolvedor detectar variações do comportamento esperado e capturar o comportamento atual, a fim de proteger contra futuros defeitos quebrando este comportamento.

Já em [Gross et al. 2009] é realizado uma avaliação a respeito do *framework EvoTest*, este voltado para teste de software de caixa branca evolucionário, essa abordagem visa encontrar casos de testes que cubram maior parte do código, a fim de maximizar a chance de identificar erros no código. O objetivo desta avaliação foi apresentar o estado atual do *framework* e apresentar os pontos críticos do mesmo. Foi apontado como uma das grandes deficiências, a falta de suporte a realização de testes em funções que possuam como parâmetro, variáveis do tipo ponteiro ou array. Além disso, tal ferramenta não suporta a avaliação de mais de um método em simultâneo.

Em ambas as ferramentas avaliadas, observou-se que estas são restritas há algum nível e/ou critério de teste. O EvoSuite, por exemplo, apoia apenas um critério de teste, o teste de mutação. Já o EvoTest, apoia apenas um nível de teste, o teste de unidade. Além disso, ambas são restritas a uma linguagem de programação, o EvoSuite trabalha apenas com programas escritos em Java, enquanto que o EvoTest, com programas escritos em C. Isso mostra que, além de existirem poucas ferramentas que deem apoio a aplicação de abordagens SBST, estas poucas, são restritas em algum aspecto, dificultando assim a aplicação de tais abordagens. Nesse sentido, evidencia-se a necessidade de uma ferramenta que seja capaz de apoiar diversos níveis e critérios de teste.

3. O Framework Proposto

Existe uma grande variedade de ferramentas para aplicação das mais variadas técnicas de teste de software. Da mesma forma, há uma grande variedade de *frameworks* que facilitam o desenvolvimento de técnicas de busca. Todavia, são poucas as ferramentas disponíveis para aplicação de abordagens especializadas em SBST. Além disso, tais fer-

ramentas exigem um conhecimento avançado das duas áreas que a SBST envolve: a) na área da atividade/técnica do Teste de Software que se deseja otimizar e; b) na área da técnica de busca necessária para realizar a otimização.

Nesse contexto, o objetivo desse trabalho centra-se na integração desses dois tipos de ferramentas para facilitar o aplicação de técnicas de busca em problemas da área de Teste de Software. Assim, a ideia é que o *framework* proposto exerça o papel de uma camada superior às ferramentas do teste de software e dos *frameworks* das técnicas de busca, a fim de que a comunicação entre elas seja realizada de maneira mais transparente para o usuário.

Extensibilidade e flexibilidade são requisitos essenciais à arquitetura de qualquer sistema que se deseja propor. Diante disso, foi pensado em três elementos principais para o *framework*: a) *Problema*; b) *Solução* e; c) *Resultados*. A Figura 1 apresenta duas visões do *framework*: a) em forma de caso de uso e; b) em forma de diagrama de atividade.

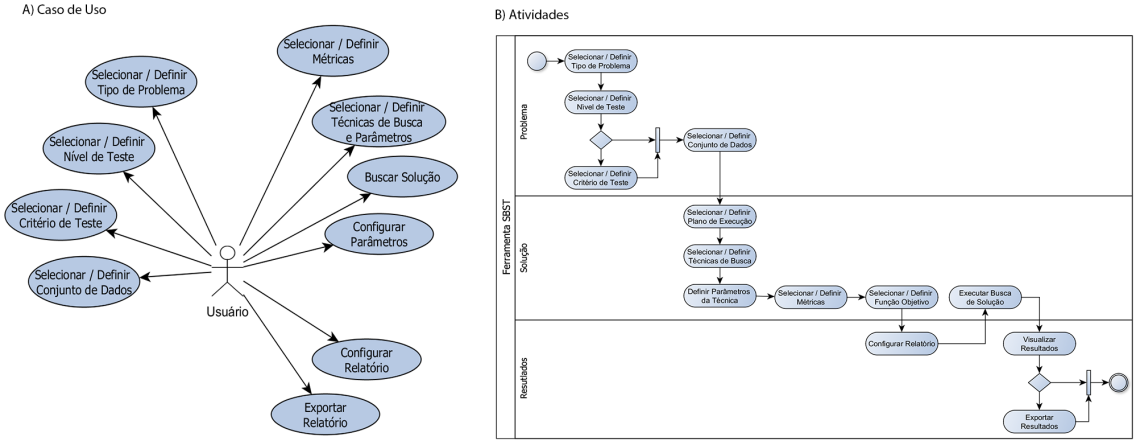


Figura 1. Diagramas do *framework* proposto: a) Caso de Uso e b) Atividade.

O *Problema* é um elemento que permite ao usuário as ações de ”Selecionar”(caso o problema que o usuário queira otimizar já esteja definido pelo *framework*) ou ”Definir”(caso o problema que o usuário queira otimizar não esteja definido pelo *framework*):

1. *Selecionar / Definir - Tipo de Problema*: determina o(s) tipo(s) do(s) problema(s) (ex.: minimização de conjunto, seleção, priorização ou geração de dados de teste);
2. *Selecionar / Definir - Nível de Teste*: determina o nível de teste do(s) problema(s);
3. *Selecionar / Definir - Critério de Teste*: determina o critério de teste que se deseja otimizar;
4. *Selecionar / Definir - Conjunto de Dados*: determina as informações do problema. Varia conforme o problema, por exemplo, caso o problema seja um problema de seleção, a nível de unidade e correspondente ao Teste de Mutação, o *Conjunto de Dados* seria constituído por P, P' e T, sendo o programa original, o conjunto de mutantes e o conjunto de casos de teste, respectivamente..

A *Solução* é um elemento que permite ao usuário as ações de “Selecionar”(caso o elemento da *Solução* que o usuário queira utilizar já esteja definido), “Definir”(caso esse elemento não esteja definido), “Configurar”(caso seja necessário atribuir valores/atributos específicos para a solução e realização do experimento) e “Buscar”(caso o usuário queira iniciar o processo de busca):

1. *Selecionar / Definir - Técnicas de Busca*: determina a(s) técnica(s) de busca utilizada(s) no experimento;
2. *Selecionar / Definir - Métricas*: determina a(s) métrica(s) que possibilitam avaliar a(s) solução(ões)s;
3. *Configurar - Parâmetros*: determina os parâmetros do experimento, da técnica de busca e do problema;
4. *Buscar - Conjunto de Dados*: dispara a busca pela(s) solução(ões).

O *Resultado* é um elemento que permite ao usuário as ações de "Configurar"e "Exportar":

1. *Configurar - Relatório*: determina a(s) métricas(s) e atributo(s) para geração do(s) relatório(s);
2. *Exportar*: determina a(s) extensão(ões) do(s) relatório(s).

Pode-se notar que os elementos do *framework* proposto podem ser aplicados nas diferentes atividades de teste passíveis de otimização, uma vez que tais elementos não estão presos a nenhuma fase, nível ou critério específico. Dessa forma, o *framework* pode ser considerado como um *framework* da SBST, uma vez que pode tratar atividades de teste de uma maneira *Plug and Play*, podendo ser extensível. Além disso, outra característica, consiste na flexibilidade do seu uso, podendo ser usada para otimizar atividades de teste ou para avaliar/desenvolver/aprimorar técnicas de busca.

O *framework* está sendo desenvolvido na linguagem de programação Java em conjunto com seu JDK (*Java SE Development Kit*). Optou-se por esta linguagem, por ser portável entre os principais sistemas operacionais do mercado. Portanto, para utilização do *framework*, basta que o usuário possua o JDK 7 update 9, ou superior, instalado na sua máquina.

Com o desenvolvimento do *framework*, espera-se que a aplicação de técnicas de busca em problemas de teste de software seja mais eficiente, na seção 2.1 é apresentado alguns dos problemas que serão beneficiados pelo *framework*. Além disso, espera-se mostrar a capacidade de aplicação dos métodos na resolução de problemas de otimização.

4. Estudo de Caso

Para melhor entender o fluxo de funcionamento do *framework* proposto (Figura 1), suponha o seguinte cenário: "*Um usuário com pouca experiência em Teste de Software precisa realizar a seleção de um subconjunto de casos de teste para realização do teste unitário de um determinado programa. Ele tem em mãos um conjunto muito grande de casos de teste, o desafio é conseguir selecionar um subconjunto, T', que represente todo conjunto, T, de casos de teste, assim T' torna-se equivalente a T, com isso, apenas com um subconjunto, o usuário conseguirá representar todo o conjunto de casos de testes. Além de selecionar um T' capaz de representar T, é necessário garantir que T' atende ao critério de cobertura de todos os casos de teste.*". Este cenário caracteriza-se como um problema de seleção de casos de teste, ideal para utilização de técnica de SBST, conforme seção 2.1.

De acordo com fluxo proposto na Figura 1, para buscar um subconjunto de casos de teste e com boa qualidade, inicialmente o usuário do cenário descrito precisa configurar o *Problema* (passo 1). O fluxo do passo 1 consiste em : a) selecionar / definir o tipo de problema (passo 1.1): seleção de casos de teste; b) selecionar / definir o nível de teste

(passo 1.2): teste de unidade; c) selecionar / definir o critério de teste (passo 1.3): critério de teste de mutação; e d) selecionar / definir o conjunto de dados (passo 1.4): P, P' e T, sendo os caminhos para o programa original, o conjunto de mutantes e o conjunto de casos de teste, respectivamente. Considerando que os casos de teste estejam escritos no formato JUnit 4.0, o *framework* pode utilizar de uma base de *Ferramentas de Teste de Software* para reconhecer esse formato de forma transparente ao usuário. Com isso, o problema está configurado. A Figura 2 apresenta como foi feita esta configuração no código, utilizando o *framework* aqui proposto.

```
problema.setTipoProblema(TipoProblema.SELECAO);
problema.setNivelTeste(NivelTeste.UNIDADE);
problema.setCriterioTeste(CriterioTeste.TESTE_MUTACAO);
problema.setConjuntoDados(CAMINHO_PROGRAMA_ORIGINAL, CAMINHO_PROGRAMA_MUTANTE, CAMINHO_CASO_TESTE);
```

Figura 2. Configuração do Problema

O *Problema* configurado serve como entrada para o elemento *Solução* (passo 2). Nesse passo, configura-se o elemento com: a) selecionar / definir o plano de execução que será utilizado na realização da busca (passo 2.1): número de técnicas; b) selecionar / definir técnica de busca que será aplicada (passo 2.2): algoritmo genético. A seleção da técnica é apoiada por *Frameworks de Metaheurísticas*, como o *Wachtmaker*, por exemplo, considerados pelo *framework* proposto. Dessa forma, o *framework* consiste em uma camada superior aos *frameworks*, simplificando e especializando o seu uso para otimizar problemas de SBST.

Após selecionada / definida a técnica de busca, c) selecionar / definir os parâmetros específicos para execução da técnica (passo 2.3): tamanho da população, tamanho do indivíduo, no contexto da técnica escolhida, algoritmo genético; d) definir as métricas a serem coletadas na realização da busca (passo 2.4): número de execuções de cada experimento, cobertura de código, score de mutação. A definição das métricas é essencial para avaliação do resultado da busca realizada. Tal passo, considera como entrada o elemento *Problema* e qual ou quais técnicas selecionadas ou definidas foram configuradas, uma vez que diferentes critérios ou algoritmos de busca, por exemplo, podem exigir diferentes métricas para serem avaliadas. Com isso finaliza-se a configuração do elemento *Solução*.e) definir a função objetivo (passo 2.5): maior *score* de mutação. A figura 3 apresenta como foi configurada a solução do problema no código.

```
solucao.setPlanoExecucao(NUMERO_DE_EXECUCOES,NUMERO_DE_TECNICAS);
solucao.tecnica[0].setTecnicaDeBusca(TecnicasDeBusca.ALGORITMO_GENETICO);
solucao.tecnica[0].setParametrosTecnica(TAMANHO_POPULACAO, NUMERO_DE_GERACOES, TAMANHO_POPULACAO, TAXA_DE_CRUZAMENTO);
solucao.tecnica[0].setMetricas(SCORE_MUTACAO, COBERTURA_DE_CÓDIGO);
solucao.tecnica[0].setFuncaoObjetivo(FUNCAO_OBJETIVO);
```

Figura 3. Configuração da Solução

Na sequência, o elemento *Resultados* é a próxima configuração (passo 3), com : a) configuração do relatório, no qual serão instanciadas quais as informações serão apresentadas no relatório, definir os comparativos a serem realizados (passo 3.1): resultados padrão, dados estatísticos e melhor subconjunto de caso de teste. A configuração do relatório depende das métricas que foram definidas, visto que o resultado destas serão fomento para as informações do(s) relatório(s). A figura 4 traz a linha de código necessária para configuração do relatório a ser gerado para este experimento.

```
resultados.ConfigurarRelatorio(RESULTADO_PADRAO, DADOS_ESTATISTICOS, MELHOR_SUBCONJUNTO_CASO_DE_TESTE);
```

Figura 4. Configuração dos Resultados

Realizadas todas as configurações passo 1, 2 e 3 o usuário deverá executar a busca de solução (passo 2.5), que consiste na execução da técnica selecionada sobre o problema. Após execução serão apresentados os relatórios gerados para visualização dos resultados, previamente configurados (passo 3.2). Tais relatórios poderão ser analisados pelo usuário para realização da análise da(s) solução(s) encontrada(s), no caso, a avaliação da qualidade do subconjunto de casos de teste. Além disso, esses relatórios poderão ser exportados ou não (passo 3.3).

5. Resultados e Discussões

Considerando o estudo de caso proposto na Seção 4, a análise da abordagem proposta considera os seguintes itens: a) dependências de recursos de software para utilização do *framework*; b) facilidade na configuração do experimento (problema, solução e resultados); c) reutilização de componentes já existentes.

Em termos de dependência de outros recursos de software, o uso do *framework* proposto tem como pré-requisito apenas a instalação do JDK, o que lhe dá uma maior portabilidade. Já no que diz respeito a facilidade na configuração do experimento, pode-se dizer que o *framework* abstrai diversas operações. O usuário não precisou interagir com a ferramenta MuJava para interpretar os arquivos que continham o conjunto de dados de teste, por exemplo. Outra operação abstraída foi a interação com o *framework watchmaker* para implementação da técnica de busca, neste cenário, o algoritmo genético.

Destaca-se também a reutilização de recursos de software já existentes, como o *framewok watchmaker*, um *framework* orientado a objetos para implementação de algoritmos evolucionários em Java, facilitando a implementação das técnicas de busca. O *Jfreechart* é outro recurso de software reutilizado pelo *framework*, para fornecer uma visualização gráfica do(s) resultado(s) do(s) experimento(s).

Além disso, pode-se evidenciar a quantidade de linhas de código que são abstraídas perante o usuário. Como apresentado nas Figuras 2, 3 e 4, são necessárias apenas 10 linhas de código para configuração do experimento (problema, solução e resultados). São abstraídas 300 linhas de código necessárias para implementação da técnica de busca, 239 linhas de código para comunicação com a ferramenta MuJava na interpretação do conjunto de dados de teste, 30 linhas para comunicar com a biblioteca Jfreechart para confecção dos gráficos, 122 linhas para configuração e execução dos experimentos e 60 linhas para configuração e apresentação dos resultados, totalizando 721 linhas de código abstraídas e transparentes para o usuário, para esse exemplo.

6. Conclusões e Trabalhos Futuros

Neste trabalho fora apresentado a importância do Teste de Software e a necessidade de modelar os problemas da área de forma matemática, para que assim, possa-se otimizar as atividades de teste, por meio do uso de técnicas de busca, como por exemplo, o uso de metaheurísticas. Nessa linha, evidenciou-se a escassez de ferramentas que apoiassem a

aplicação de abordagens especializadas em SBST, a experimentação de técnicas de busca, visando a otimização das atividades de teste de sofware.

Sendo assim, foi apresentado a proposta do desenvolvimento de um *framework*, chamado SBSTFrame, que apoiasse a aplicação das abordagens SBST e a experimentação dessas abordagens, exercendo assim o papel de uma camada superior às ferramentas do teste de software e dos *frameworks* das técnicas de busca já existentes, realizando a comunicação entre eles de maneira transparente para o usuário.

Apresentado-se um exemplo de utilização do *framework* em um cenário fictício para ilustrar como o usuário deverá interagir com o *framework*. Foi realizada a configuração deste cenário dentro de um IDE, seguindo o fluxo de funcionamento do *framework*. Com isso, evidencio-se a abstração de linhas de código que é feita perante o usuário.

Atualmente o *framework* atende à problemas de seleção, teste de unidade e ao critério teste de mutação. Em termos de técnicas de busca, o *framework* traz o algoritmo genético canônico, algoritmo de seleção aleatória e o algoritmo de ilhas implementados utilizando o *framework watchmaker*. Todavia a proposta da ferramenta não se restringe a apenas este cenário, sendo assim, como trabalhos futuros destaca-se, a extensão da ferramenta para otimizar outras fases, níveis e critérios de teste de sofwtare. Criar novas interfaces para: a) acoplar frameworks de SBST, como EvoSuite, Evotest, como uma camada abaixo da ferramenta proposta para abstrair a complexidade envolvida em se trabalhar com os mesmos; b) inserir novos frameworks de metaheurísticas para fornecer mais algoritmos, disponibilizando abordagens distribúidas, multiobjetivo, dentre outros; c) disponibilizar o *framework* como Web Service, a fim de disseminar a aplicação de abordagens especializadas em SBST; d) ampliar o número de problemas suportados pelo SBSTFrame.

Referências

Blum, C. and Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308.

BoussaïD, I., Lepagnot, J., and Siarry, P. (2013). A survey on optimization metaheuristics. *Inf. Sci.*, 237:82–117.

Bühler, O. and Wegener, J. (2008). Evolutionary functional testing. *Comput. Oper. Res.*, 35(10):3144–3160.

Farzat, F. (2010). Test Case Selection Method for Emergency Changes. *2nd International Symposium on Search Based Software Engineering*, pages 31–35.

Fraser, G. and Arcuri, A. (2011). Evosuite: Automatic test suite generation for object-oriented software. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ESEC/FSE '11, pages 416–419, New York, NY, USA. ACM.

Freitas, F. and Maia, C. (2010). Otimização em Teste de Software com Aplicação de Metaheurísticas. *Revista de Sistemas de . . .* , 5:3–13.

Gross, H., Kruse, P., Wegener, J., and Vos, T. (2009). Evolutionary white-box software test with the evotest framework: A progress report. In *Software Testing, Verification*

and Validation Workshops, 2009. ICSTW '09. International Conference on, pages 111–120.

Harman, M. (2006). Search based software engineering. In *International Conference on Computational Science (4)*, pages 740–747.

Hemmati, H. and Briand, L. (2010). An Industrial Investigation of Similarity Measures for Model-Based Test Case Selection. *2010 IEEE 21st International Symposium on Software Reliability Engineering*, pages 141–150.

Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems*. MIT Press, Cambridge, MA, USA.

IEEE Computer Society, Pierre Bourque, R. E. F. (2014). *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0*. IEEE Computer Society Press, Los Alamitos, CA, USA, 3rd edition.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *SCIENCE*, 220(4598):671–680.

Korel, B. (1990). Automated software test data generation. *Software Engineering, IEEE Transactions on*, 16(8):870–879.

McMinn, P. (2011). Search-Based Software Testing: Past, Present and Future. *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*, pages 153–163.

Miller, W. and Spooner, D. (1976). Automatic generation of floating-point test data. *Software Engineering, IEEE Transactions on*, SE-2(3):223–226.

Myers, G. J. and Sandler, C. (2004). *The Art of Software Testing*. John Wiley & Sons.

Tayamanon, T., Suwannasart, T., Wongchingchai, N., and Methawachananont, A. (2011). Tmm appraisal assistant tool. In *Proceedings of the 2011 21st International Conference on Systems Engineering*, ICSENG '11, pages 329–333, Washington, DC, USA. IEEE Computer Society.

Tracey, N., Clark, J., Mander, K., and McDermid, J. (1998). An automated framework for structural test-data generation. In *Proceedings of the 13th IEEE International Conference on Automated Software Engineering*, ASE '98, pages 285–, Washington, DC, USA. IEEE Computer Society.

Wegener, J. and Grochtmann, M. (1998). Verifying timing constraints of real-time systems by means of evolutionary testing. *Real-Time Syst.*, 15(3):275–298.

Wegener, J., Sthamer, H., Jones, B. F., and Eyres, D. E. (1997). Testing real-time systems using genetic algorithms. *Software Quality Control*, 6(2):127–135.

Yoo, S. and Harman, M. (2007). Pareto Efficient Multi-objective Test Case Selection. *Proceedings of the 2007 international symposium on Software testing and analysis - ISSTA '07*, pages 140–50.

Yoo, S. and Harman, M. (2012). Regression testing minimization, selection and prioritization: A survey. *Softw. Test. Verif. Reliab.*, 22(2):67–120.

Um Algoritmo Genético no Modelo de Ilhas para Seleção de Casos de Teste na Análise de Mutantes

Gleibson W. Silva Borges, Beatriz P. Martins, André A. Lôbo de Oliveira, Celso G. Camilo-Junior, Auri M. R. Vincenzi, Plínio de Sá Leitão Júnior

[1]Instituto de Informática – Universidade Federal de Goiás (UFG)
Caixa Postal 131 – 74.001-970 – Goiânia – GO – Brasil

{gleibsonborges,beatrizmartins,andreoliveira,celso,auri,plinio}@inf.ufg.br

Resumo. *O custo computacional é um dos grandes problemas de aplicabilidade do Teste de Mutação, pois a sua realização pode envolver conjuntos de grande cardinalidade, considerando os casos de teste e os programas mutantes gerados. Este artigo situa-se na área dos algoritmos genéticos paralelos que objetivam a seleção de subconjuntos de casos de teste em apoio à redução de custos no Teste de Mutação. Diante disso, propõe-se o Algoritmo Genético no Modelo de Ilhas (AG-MI) para realizar a seleção de casos de teste de maneira mais eficiente. A experimentação compara os resultados do algoritmo proposto com outros dois algoritmos aplicados em um benchmark real. Os resultados revelam uma melhoria no escore de mutação e uma menor quantidade de cálculos da abordagem proposta.*

Abstract. *The computational cost is a major problem for the applicability of Mutation Testing because its implementation can involve sets of large cardinality, considering test cases and mutants programs generated. This article is situated in the field of parallel genetic algorithms that aim to select test cases subsets to support cost reduction in Mutation Testing. Therefore, we propose the Genetic Algorithm on Island Model (GA-IM) to perform the selection of test cases more efficiently. The experimentation compares the proposed algorithm results with two others algorithms applied to a real benchmark. The results show an improvement in the score mutation and a smaller amount of computation of the proposed approach.*

1. Introdução

A atividade de Teste de Software desempenha um papel muito importante no aprimoramento de sistemas computacionais. Segundo Papadakis, Malevris e Kallia (2010), as atividades de Validação e Verificação (V&V) consomem cerca de 50% à 60% do custo total no ciclo de vida de um software. Frente à alta cardinalidade do domínio de entrada de um programa, o Teste de Software se torna importante no intuito de melhorar a qualidade do produto ao revelar seus defeitos.

O Teste de Software utiliza várias abordagens para a realização do teste nas suas mais variadas fases, níveis e critérios. Todavia, existem muitos problemas da área que não são resolvidos por abordagens determinísticas em um tempo factível. O domínio de entrada de um programa pode abranger uma infinidade de valores possíveis, sendo, assim, impossível de testar todas as possibilidades [Delamaro, Maldonado, Jino 2007].

Nesse contexto, surge um campo de pesquisa denominado Sear*ch Based Software Testing (SBST)* que faz uso de metaheurísticas para buscar soluções em cenários de teste onde não se conhece métodos eficientes.

Dentre as técnicas de teste existentes, a Análise de Mutantes (ou Teste de Mutação) é um critério de teste baseado em defeitos, reconhecido por sua grande capacidade de revelar defeitos. Todavia, apresenta dois grandes problemas [Jia e Harman 2011]: i) alto esforço humano para a detecção dos mutantes equivalentes e; ii) alto custo computacional para execução dos programas mutantes. Portanto, a SBST pode ser aplicada para aprimorar a utilização desse critério de teste.

A presente pesquisa propõe o AG-MI paralelo (*multithreading*) para seleção automatizada de um bom subconjunto de casos de teste. Espera-se empregar a estratégia de subdividir tarefas grandes em tarefas pequenas para que possam ser resolvidas paralelamente e com maior eficácia. O espaço de busca passa a ser explorado por subpopulações chamadas ilhas, sendo cada ilha formada por subconjuntos de casos de teste chamados indivíduos, os quais podem ser trocados entre as ilhas para promover uma melhor evolução [Silva 2005].

A estrutura deste artigo se apresenta como segue: a Seção 2 abrange uma revisão acerca da Análise de Mutantes e faz uma análise sobre trabalhos correlatos à abordagem proposta. Na Seção 3 a abordagem proposta é apresentada e a Seção 4 descreve a realização dos experimentos. Por fim, a Seção 5 expõe as conclusões e trabalhos futuros.

2. Revisão

2.1. Análise de Mutantes

DeMillo, Lipton, e Sayward (1978), publicaram o primeiro artigo sobre o Teste de Mutação, apresentando a ideia da técnica que está fundamentada em dois pilares: i) a hipótese do programador competente - programadores experientes escrevem programas muito próximos do correto; ii) efeito de acoplamento – defeitos complexos de serem descobertos, isto é, defeitos que causam disturbios maiores no desempenho do sistema, estão relacionados a defeitos simples, aqueles em que pouquissimos Casos de Teste não conseguiriam encontrá-los.

De acordo com A. Oliveira, C. Camilo-Junior e A. Vincenzi (2013), o Teste de Mutação utiliza um conjunto de programas mutantes *P'*, para verificar a adequação de um conjunto de teste *T*. Esse conjunto de mutantes *P'* é gerado por meio de operadores que inserem desvios sintáticos no programa original *P*. A ideia é avaliar o desempenho de *T* sobre *P'* tendo como base avaliativa *P*. Diz-se que um programa mutante é "morto" quando as execuções de um determinado caso de teste sobre o programa original e o mutante geram saídas diferentes.

A métrica que avalia a qualidade dos casos de teste, chamada escore de mutação, é representada por um número real que varia de 0.0 a 1.0 calculado conforme Equação 1. Assim, um subconjunto de casos de teste mais adequado é aquele que "mata" uma maior quantidade de mutantes.

$$ms(P,T) = \frac{\mathrm{DM(P,T)}}{\mathrm{M(P)} - EM(P)} \quad (1)$$

Onde:
- *ms(P, T)*: escore de mutação do conjunto de testes T;
- *P*: programa original;
- *T*: total de casos de teste;
- *DM* (*P*, *T*): total de programas mutantes mortos por *T*;
- *M(P)*: número de mutantes do programa *P*.
- *EM* (*P*): total de mutantes equivalentes.

2.2. Trabalhos Correlatos

O Modelo de Ilhas é uma versão paralela dos Algoritmos Genéticos que mantêm a estrutura do AG canônico, porém divide o conjunto de soluções candidatas do problema em vários subconjuntos menores. Durante a etapa de seleção, estes grupos podem evoluir individualmente e em paralelo, visando melhorar a qualidade dos indivíduos.

Quanto à utilização do paralelismo, assim como neste trabalho, Silva (2005) trata da implementação de um algoritmo genético paralelo utilizando o modelo de ilhas, onde populações aleatórias são geradas de maneira independente em cada uma das ilhas, as quais ficam restritas a processadores específicos, objetivando a disponibilização de uma ferramenta de alto desempenho que faz uso de paralelismo.

Também no contexto dos algoritmos paralelos, Costa (2012) procurando compreender melhor a dinâmica entre os processos evolucionário e migratório no Modelo de Ilhas e utilizando-se do conceito das Topologias de Migração de indivíduos, discorre um estudo sobre a facilidade que as Topologias Dinâmicas oferecem para reconfigurar o fluxo com que indivíduos migram a cada execução, os autores tentam alcançar uma economia de tempo distribuindo o esforço computacional, beneficiando-se de uma configuração paralela.

Quanto aos correlatos à área de Teste de Mutação, Oliveira (2013) propôs um Algoritmo Genético Coevolucionário para seleção automática de bons subconjuntos de casos de teste e bons subconjuntos de mutantes para o Teste de Mutação. Nessa proposta, usa-se mais de uma população (coevolução competitiva) para aprimorar a busca. Como resultado, consegue-se reduzir o custo computacional, sem reduzir a eficácia da solução.

Louzada (2012) também faz uso de técnicas heurísticas para seleção de Casos de Teste, este propõe o uso de um Algoritmo Genético Elitista (GA), como uma ferramenta para a geração e seleção de dados de teste aplicados no Teste de Mutação para diferentes benchmarks. Seus resultados indicam um bom desempenho do algoritmo usado nos benchmarks.

Para a seleção de casos de teste, em específico, não foram localizadas referências com a utilização do Modelo de Ilhas no contexto da Análise de Mutantes. Assim, o presente trabalho objetiva explorar o uso do Modelo para aprimorar a realização da Análise de Mutantes. A Seção 3 descreve a abordagem proposta.

3. Abordagem Proposta

Observando ambientes relativamente isolados na natureza, como ilhas, percebeu-se que algumas espécies nativas possuem interessantes adaptações às particularidades do ambiente em que vivem.

A idéia é fazer uso do paralelismo, o qual se propõe como uma estratégia muito utilizada em computação, para obter resultados com mais agilidade em tarefas complexas, visto que nessa estratégia, uma tarefa grande pode ser dividida em tarefas menores e posteriormente serem resolvidas. Os algoritmos genéticos possuem uma estrutura computacional altamente paralelizável, assim, alguns métodos de paralelização usam uma única população, enquanto outros dividem a população em várias subpopulações relativamente isoladas.

Na escolha desta abordagem, observou-se 4 pontos positivos em seu uso: i) aumento de desempenho (redução de tempo) no processamento; ii) capacidade de resolver grandes desafios computacionais; iii) utilização de um sistema distribuído na resolução de tarefas; iv) e obtenção de ganhos de performance [Silva2005].

A possibilidade de várias subpopulações concorrentes poderem gerar uma melhor solução (Figura 1), levou à construção do AG-MI que aproveita bem essa vantagem. O algoritmo seqüencial fica dividido numa série de ilhas (populações de casos de teste), que evoluem paralelamente e comunicam entre si as melhores soluções.

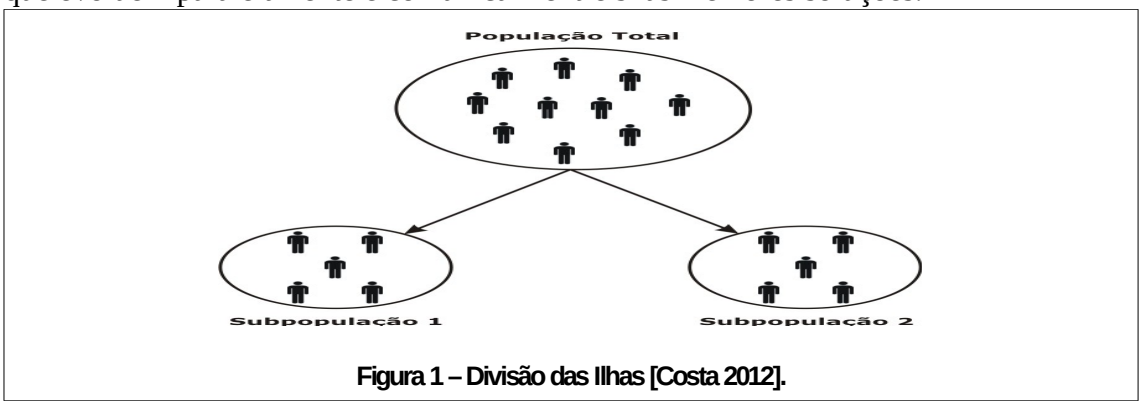


Figura 1 – Divisão das Ilhas [Costa 2012].

Este modelo mantém, entretanto, a estrutura seqüencial do algoritmo genético, apenas sendo adicionados os operadores para fazer a migração dos elementos entre as diferentes ilhas. As populações iniciais são geradas de modo idêntico à sua versão seqüencial, mas o modelo de ilhas introduz um operador de migração, além das três etapas básicas (seleção, cruzamento e mutação), que é usado para enviar indivíduos de uma subpopulação para outra. A Figura 2 ilustra os passos do algoritmo conforme segue:

1. *Inicializa* as ilhas a partir de uma população inicial de cardinalidade alta, isto é um domínio de entrada com muitos indivíduos;
2. *Seleciona* os melhores indivíduos dentre as ilhas;
3. *Calcula-se* a aptidão da subpopulação macro selecionada;
4. *Verifica* se o limite de gerações pré-definido foi atingido (finaliza o processo, se não segue para *Seleção do AG)*;

5. *Operador de Seleção* é lançado (seleciona os pais para o cruzamento);
6. *Operador de Cruzamento* gera os filhos conforme operador de cruzamento;
7. *Operador de mutação* realiza a mutação conforme operador específico.

O processo de Migração desempenha um papel muito importante para o AG-MI, visto que é responsável por promover a diversificação das características dos indivíduos em cada ilha. Já a frequência de migração não deve ser muito alta devido aos elevados custos de comunicação neste tipo de sistema. Com isso o operador de migração precisa ser executado apenas quando existir a necessidade de renovação de alguma subpopulação.

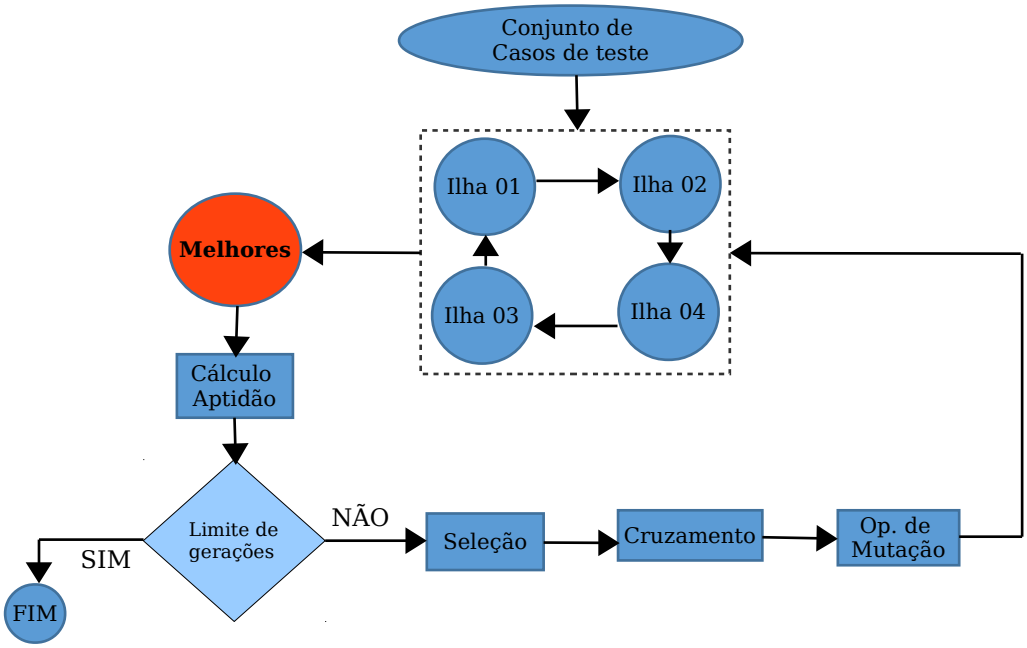


Figura 2 – Fluxograma do AG com Modelo de Ilhas (AG-MI)

No intuito de diversificar a evolução de cada ilha, são utilizadas topologias de migração (Figura 3), que definem que um indivíduo será migrado de uma ilha à outra sempre que uma época e/ou geração pré-definida for contemplada. Isso garante que o processo de evolução individual se mantenha diversificado.

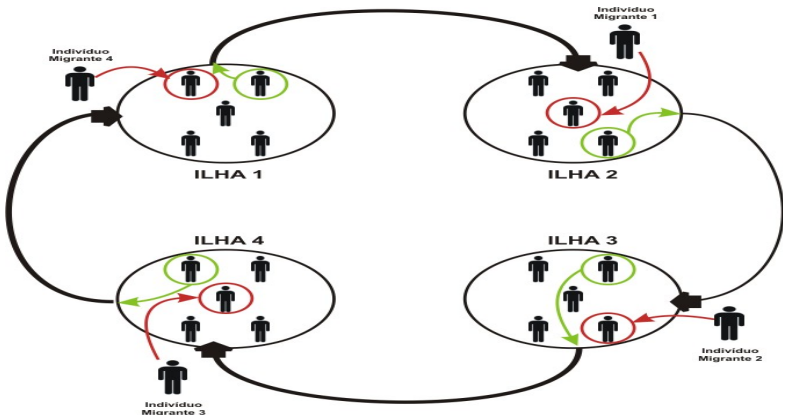


Figura 3 – Estratégia de migração com Topologia em anel [Costa 2012].

A presente pesquisa propõe o AG-MI para automatizar a seleção de bons subconjuntos de casos de teste para o Teste de Mutação. Tal proposta é baseada em dois aspectos principais:

1. Alta cardinalidade de mutantes e casos de teste: uma estratégia para redução do custo computacional da execução de mutantes consiste em selecionar um subconjunto de casos de teste com alto escore de mutação. Assim, uma boa cobertura no teste pode ser obtida com baixo custo para sua realização. Todavia, selecionar esse subconjunto pode não ser uma tarefa fácil;
2. A diversidade genética: é a característica central do AG-MI. As subpopulações objetivam alcançar tal diversidade para minimizar a posssibilidade do algoritmo genético ficar preso numa região de ótimo local do espaço de busca.

4. Estudos Experimentais

4.1. Algoritmos e Benchmarks

A experimentação comparou o algoritmo proposto (AG-MI) com outros dois algoritmos:

1. AG: algoritmo genético canônico que realiza a seleção com apenas uma população. Calcula-se a aptidão dos indivíduos conforme a Equação 1. O melhor indivíduo (melhor subconjunto de casos de teste) das gerações é a saída do algoritmo;
2. AL: consiste no algoritmo de abordagem aleatória. O algoritmo pode ser descrito em 3 passos: 1) gera-se indivíduos na quantidade de vezes do número de gerações multiplicada pelo tamanho da população (parâmetros utilizados no AG e AG-MI); 2) calcula-se a aptidão dos indivíduos conforme Equação 1; 3) escolhe-se o melhor indíviduo entre todos os analisados.

O programa *cal*, escrito na linguagem C, foi utilizado como benchmark da experimentação já com os dados sobre Mutantes e Casos de Teste definidos. Esse programa é um utilitário Linux em que o usuário fornece como entrada o ano e/ou o mês, por linha de comando, e o programa retorna: a) o calendário do ano, quando se fornece apenas um parâmetro e; b) o calendário de um mês referente a um ano específico, quando se fornece dois parâmetros. Para geração dos programas mutantes foi utilizada a ferramenta Proteum [Delamaro 1996], que é um ambiente de Teste de Mutação integrado que une ferramentas de apoio à especificação e teste de programas baseado em Mutação, além disso visa aplicar a mutação a nível de integração.

O benchmark além de possuir uma baixa probabilidade de defeitos em seu código, devido ao seu uso intenso, foi amplamente utilizado em estudos anteriores. Por exemplo, as pesquisas de Wong (1993) e de Vincenzi (1998). Por isso, o cal foi o programa utilizado como benchmark na presente pesquisa e possui os seguintes dados: a) total de 4622 mutantes; b) 344 mutantes equivalentes e; c) um total de 2000 Casos de Teste.

4.2. Parâmetros dos algoritmos

Para todos os algoritmos foram fixados os seguintes parâmetros: a) operador de seleção: torneio com 2 competidores; b) taxa de cruzamento: 95%; c) taxa de mutação: 5%; d) elitismo: 1 indivíduo; e) tamanho da população: 100 e; f) condição de término: 5 minutos ou fitness de 0,9981 (escore máximo obtido com o conjunto total de casos de teste).

Para o total de 2000 Casos de Teste existentes no benchmark, fixou-se também o tamanho do subconjunto em 20, isto é, o objetivo foi selecionar 20 Casos de Teste dentre estes, que sejam eficazes em matar os melhores mutantes. Esse tamanho de subconjunto foi utilizado para todos os algoritmos comparados e consiste no tamanho mínimo necessário ao benchmark *cal*.

Vale observar que os experimentos consistem na variação da quantidade de migração dos indivíduos de uma ilha para outra.

4.3. Discussão dos Resultados

A ideia da experimentação consiste em comparar o AG-MI com o AG tradicional (sem ilhas) e com o algoritmo aleatório em termos de quantidade de avaliações. Todos os resultados foram obtidos por meio de 30 execuções, sendo cada uma de 5 minutos, em média.

Porém, inicialmente foi experimentada a técnica com o parâmetro *época* variando em 25, 50 e 100 (Tabela 1) e extraindo-se resultados por apenas 5 execuções com cada valor, visando descobrir qual o melhor para a seleção de casos de teste, uma vez que este parâmetro funciona como gerações próprias de cada ilha. Com base na elevação do escore máximo e na quantidade de avaliações reduzidas, observou-se que atribuir 50 à quantidade de épocas mostrou-se como a melhor opção.

Época 25					
Benchmark	**MAX**	**MED**	**DESVP**	**QTDAVAL**	**TEXEC**
cal	0,9873773	0,9662615	0,0022556	13020	33,4973
Época 50					
Benchmark	**MAX**	**MED**	**DESVP**	**QTDAVAL**	**TEXEC**
cal	0,9878448	0,9827801	0,0000385	11307	33,866
Época 100					
Benchmark	**MAX**	**MED**	**DESVP**	**QTDAVAL**	**TEXEC**
cal	0,9403927	0,9302634	0,0002802	16501	42,687

Tabela 1: Resultados dos experimentos variando a quantidade de épocas.

Foram realizados mais 3 novos experimentos, com o objetivo de encontrar um escore maior com um tempo de execução menor, variando somente o parâmetro de *migração* de indivíduos de uma ilha à outra, entre 1, 5 e 10. A Tabela 2 apresenta os resultados dos experimentos com relação à variação da migração de indivíduos entre as

diferentes ilhas, foi observado que quanto maior o número de indivíduos migrantes, maior é a média de escore entre as execuções.

Migração 1					
Benchmark	**MAX**	**MED**	**DESVP**	**QTDAVAL**	**TEXEC**
cal	0,9962134	0,9767854	0.0001254	118110	300
Migração 5					
Benchmark	**MAX**	**MED**	**DESVP**	**QTDAVAL**	**TEXEC**
cal	0,9974287	0,9960418	0,0001317	119926	300
Migração 10					
Benchmark	**MAX**	**MED**	**DESVP**	**QTDAVAL**	**TEXEC**
cal	0,9974287	0,9966106	0,0000667	114290	300

Tabela 2: Resultados dos experimentos variando a migração.

Legendas da Tabela 1: a) MAX: escore máximo alcançado; b) MED: média dos escores de mutação; c) DESVP: desvio padrão; d) QTDAVAL: quantidade de avaliações; e) TEXEC: tempo de execução em segundos.

4.3.1. Comparação das diferentes abordagens

A Tabela 3 apresenta um comparativo entre as diferentes abordagens. Nota-se que todas ultrapassaram o tempo máximo de execução de 5 min, o que não proporcionou a chegada ao escore máximo fixado (0,9981).

Foi comparado a melhor execução do AG-MI com os parâmetros fixos: a) migração: 10; b) época: 50; c) quantidade de ilhas: 2 e; d) tamanho da população: 50 (fixado em [tamanho da população]/[quantidade de ilhas]).

Algoritmo	**Escore Máximo**	**Média**	**Desvio Padrão**	**Quantidade de Avaliações**	**Tempo de Execução**
AL	0,9752221	0,9701262	0,0001525	53023	300
AG	0,9974287	0,9948652	0,0035179	137136	300
AG-MI	0,9974287	0,9966106	0,0000667	114290	300

Tabela 3: Comparação entre as abordagens utilizando o Benchmark cal.

O Gráfico 1 apresenta a evolução do escore que cada algoritmo demonstrou em suas execuções, bem como a média obtida entre os escores inicial e final. O AG e o AG-MI empataram no quesito *escore máximo*. No entanto, a *média* do AG-MI foi maior que as outras abordagens demonstrando que, no geral, o escore chegou a um valor maior em cada execução.

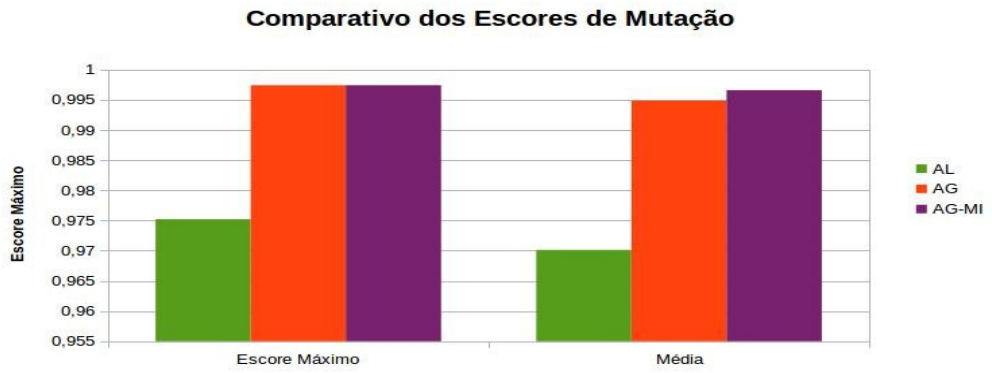


Gráfico 1 – Comparação dos escores máximos e média de cada algoritmo

Outro fator a ser destacado é a quantidade de avaliações que cada algoritmo faz para elevar o escore, para isso o Gráfico 2 apresenta a quantidade de avaliações de cada um, ilustrando este fator.

A técnica aleatória faz um número bem reduzido de cálculos em suas execuções, porém possui um escore de mutação inferior às outras abordagens. O AG-MI faz menos avaliações em relação ao AG canônico (tradicional), isso porque realiza o processo de elevação do escore mais rapidamente por causa do paralelismo e visto que chegou ao mesmo escore em mesmo tempo. Esse resultado evidencia que o AG-MI obteve o melhor resultado com menor esforço quando comparado ao AG tradicional. Acredita-se que tal melhoria foi obtida pela diversidade que o modelo em ilhas proporciona.

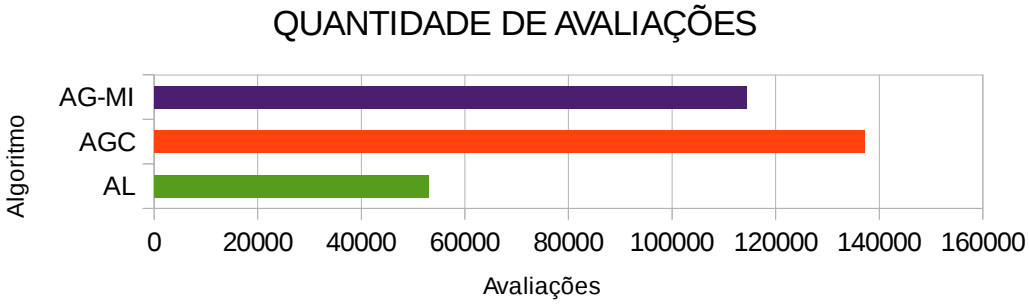


Gráfico 2 – Comparação da quantidade de avaliações dos algoritmos

Visto que existem configurações específicas do AG-MI, como número de épocas e migração, a generalização dos resultados pode se tornar uma ameaça à validade dos mesmos. Ao se contabilizar os números extraídos dos experimentos sem levar em conta as particularidades do AG-MI, de certa forma o algoritmo proposto pode não ser uma estratégia viável quando comparado ao AG canônico.

5. Conclusões e Trabalhos Futuros

Neste trabalho foi abordado um Algoritmo Genético Paralelo, utilizando-se o Modelo de Ilhas para selecionar Casos de Teste no contexto da Análise de Mutantes.

O AG-MI foi avaliado sob a perspectiva de melhoria na seleção de subconjuntos de

casos de teste. Inicialmente foram apresentadas algumas abordagens de seleção de subconjuntos, dentre elas o AG (canônico) e o AG Aleatório, os quais foram escolhidas para comparação com o AG-MI.

Utilizou-se o *benchmark* cal como forma de verificação da agilidade do AG-MI na obtenção dos subconjuntos de tamanhos reduzidos. Diante disso, acredita-se que o AG-MI constitui-se numa abordagem promissora para minimização de subconjuntos de casos de teste como contribução à redução de custos do Teste de Mutação.

Como trabalho futuro, almeja-se realizar experimentos mais robustos com vistas a explorar a técnica do Modelo em Ilhas para se obter maiores resultados acerca das melhorias proporcionadas pela estratégia.

Referências

A. Oliveira (2013) “Uma Abordagem Coevolucionária para seleção de Casos de Teste e Mutantes no Contexto do Teste de Mutação”. Dissertação, 155 f. Goiânia, UFG, Inf.

A. Oliveira, C. Camilo-Junior e A. Vincenzi (2013) “Um Algoritmo Genético Coevolucionário com Classificação Genética Controlada aplicado ao Teste de Mutação”. IV WESB, Brasília. CBSoft: Teoria e Prática.

A. Silva (2005) “Implementação de um Algoritmo Genético utilizando o Modelo de Ilhas”. Dissertação, 73 f. - Rio de Janeiro, COPPE/UFRJ.

A. Vincenzi (1998) “Subsídios para o Estabelecimento de Estratégias de Teste Baseadas na Técnica de Mutação”. PhD thesis, Universidade de São Paulo.

J. Louzada, C. G. Camilo-Junior, A. M. R. Vincenzi, C. L. Rodrigues (2012), "An elitist evolutionary algorithm for automatically generating test data", Evolutionary Computation (CEC), IEEE Congress on , vol., no., pp.1,8.

M. Delamaro, J. Maldonado, M. Jino (2007) “Introdução ao Teste de Software”, Ed. Elsevier - Rio de Janeiro.

M. Delamaro (1996) “Proteum - A Tool for the Assessment of Test Adequacy for C ”. In: Proceedings of the Conference on Performability in Computing Systems (PCS), p. 79-95.

M. Papadakis, N. Malevris, and M. Kallia (2010) “Towards automating the Generation of Mutation Tests”. Workshop on Automation of Software Test (AST '10) p. 111-118.

R. DeMillo, R. Lipton, F. Sayward, Hints on Test Data Selection: Help for the Practicing Programmer. Computer, 11(4):34–41, Apr. 1978.

R. Costa (2012) em “Topologias Dinâmicas para Modelo em Ilhas usando Evolução Diferencial”. Dissertação, 113 f. - UFMG, Escola de Engenharia.

Y. Jia, and M. Harman (2011) “An analysis and survey of the development of mutation testing”. In: IEEE Transactions on Software Engineering, p. 649-678.

W. E. Wong (1993). “On mutation and data flow”. Technical report, Purdue University.

Uso de Algoritmo Genético Distribuído na Seleção de Casos de Teste para o Teste de Mutação

Acabias Marques Luiz[1], André Assis Lôbo de Oliveira[1], Celso Gonçalves Camilo-Junior[1], Cássio Leonardo Rodrigues[1], Auri Marcelo Rizzo Vincenzi[1]

[1]Instituto de Informática – Universidade Federal de Goiás (UFG)
Caixa Postal 131 – 74.001-970 – Goiânia – GO – Brasil

{acabiasml,andre.assis.lobo}@gmail.com, {celso,cassio,auri}@inf.ufg.br

Abstract. *This paper presents the use of parallel processing and distributed in Mutation Analysis. Based on hight computation cost related to Mutation Analysis, this paper presents optimization technique to reduce its computation cost without lost effectiveness. The optimization process relay on search for a good set of test cases for a group of mutants. The results found indicate that approach using ECJ framework is able to obtain gains in runtime process.*

Resumo. *Este artigo apresenta o uso de processamento paralelo e distribuído na Análise de Mutantes. Com base no alto custo computacional referente à Análise de Mutantes, este trabalho apresenta uma técnica de otimização que reduz esse custo sem perda de eficácia. Otimiza-se o tempo da busca por um bom conjunto de casos de teste para um grupo de mutantes. Apresenta-se os resultados de experimentos, indicando a abordagem utilizada com o uso framework ECJ, que revelam ganha no tempo de execução.*

1. Introdução

Teste de Software é uma área da Engenharia de Software [IEEE Computer Society 2014] de grande importância para o desenvolvimento e manutenção de software com qualidade. Ela se faz necessária, pois por meio dessa etapa identifica-se se todo um programa, ou parte dele, atende a especificação e cumpre com seu objetivo, revelando possíveis erros ou falta de recursos. Além disso, o Teste de Software tem grande destaque dentre as atividades de validação e verificação. A importância de tais atividades pode ser vista no investimento para o seu emprego, podendo consumir mais da metade do orçamento do desenvolvimento do sistema [Sommerville 2007].

As principais técnicas para o teste de software são de ordem funcional (caixa preta), estrutural (caixa branca) e baseada em defeitos [Pressman 2011]. Entre as várias técnicas e critérios do Teste de Software, este trabalho aborda o Análise de Mutantes (ou Teste de Mutação), um critério de teste conhecido por sua grande eficácia em detectar defeitos [Wong et al. 1995]. Tal critério foi proposto em DeMillo et al. 1978 e objetiva verificar se um determinado conjunto de casos de teste é adequado para realizar o teste de determinado programa. Todavia, a Análise de Mutantes é um critério que apresenta dois grandes problemas de aplicabilidade [Jia and Harman 2011]: i) a geração de uma grande quantidade de mutantes que agrega grande custo para sua execução e; ii) o problema dos mutantes equivalentes que exige esforço humano para sua verificação.

A Search-Based Software Testing (SBST) é uma abordagem que se utiliza de meta-heurísticas para otimizar problemas de Teste de Software que podem ser modelados matematicamente. Para isso, utiliza-se de meta-heurísticas [McMinn 2011], um vez que tais problemas apresentam grande complexidade de busca, não sendo resolvido por abordagens determinísticas.

A proposta consiste em unir o potencial do paralelismo e de um ambiente distribuído com a capacidade de busca de um Algoritmo Genético na resolução do problema de alto custo computacional da Análise de Mutantes. Com a experimentação foi possível perceber que, dado a abordagem utilizada, o ganho em tempo de execução é significativo.

O restante desse artigo é organizado como segue. Na Seção 2 é apresentada a fundamentação teórica, priorizando a Análise de Mutantes e trabalhos relacionados. A Seção 3 apresenta as abordagem usada na experimentação. Em seguida os resultados obtidos são discutidos na Seção 4. Por fim, as conclusões relacionadas com o estudos são apresentadas na Seção 5.

2. Fundamentação Teórica

Nesta seção tem-se a apresentação dos conceitos de teste de mutação, teste de software baseado em busca e algumas características dos algoritmos genéticos distribuídos.

2.1. Análise de Mutantes

A Análise de Mutantes (MA), nome dado ao Teste de Mutação (TM), teve um dos primeiros trabalhos publicados por DeMillo et al. 1978. Consiste em um critério de teste baseado em defeitos fundamentado em dois pilares: i) a hipótese do programador competente; e ii) efeito de acoplamento.

A hipótese do programador competente assume que programadores experientes escrevem programas muito próximos de estarem corretos. Já o efeito de acoplamento assume que defeitos complexos estão relacionados à defeitos simples. Assim, a descoberta de um defeito simples pode levar a detecção de defeitos complexos.

A ideia é simular possíveis defeitos que ocorrerem em um programa. A partir de um programa original P, um conjunto P' de programas modificados (mutantes) por operadores de mutação inserem pequenos desvios sintáticos no programa (source code ou byte code). O objetivo consiste em avaliar o quanto um conjunto de teste T é adequado [DeMillo 1989]. O caso de teste executa sobre o programa original e o mutante, caso as saídas sejam diferentes o mutante é dito estar *morto*.

Um problema enfrentado no TM é a geração de uma grande quantidade de programas mutantes. Na prática, isso implica em um alto custo computacional para sua realização. Outro problema, consiste na geração de mutantes equivalentes. A verificação da equivalência entre dois programas exige esforço humano por ser um problema computacionalmente indecidível [Jia and Harman 2011].

A métrica que define a adequabilidade de um conjunto de teste é chamada de *escore de mutação* sendo um número real que varia entre 0.0 e 1.0, calculado conforme a Equação (1). Assim, quanto maior o *escore de mutação* de um caso de teste, maior é a sua capacidade de demonstrar que o programa em teste não contém o defeito representado pelo mutante, ou seja, de matar programas mutantes.

$$ms(P,T) = \frac{DM(P,T)}{M(P) - EM(P)} \quad (1)$$

Onde:

- $ms(P,T)$: escore de mutação do conjunto de testes T;
- P: programa original;
- T: total de casos de teste;
- $DM(P,T)$: total de programas mutantes mortos por T;
- $M(P)$: conjunto de mutantes do programa P;
- $EM(P)$: total de mutantes equivalentes.

Apesar da grande eficácia da Análise de Mutantes para revelação de defeitos, ele possui problemas de aplicabilidade, não sendo muito usado fora da área acadêmica. Dentre as dificuldades estão as questões de equivalência entre mutantes, a geração de mutantes e o custo computacional para a execução desses mutantes.

Diante disso, a proposta do presente trabalho consiste em propor uma solução distribuída que torne a execução dos mutantes gerados mais eficiente em termos de tempo de processamento, mas também mais eficaz com uso de um algoritmo genético.

2.2. Teste de Software Baseado em Busca e Trabalhos Correlatos

O teste de software baseado em busca (em inglês, *Search-Based Software Testing*, SBST) pode ser definido como o uso de meta-heurísticas que otimizem as atividades de teste [Lima et al. 2012]. Este trabalho utiliza-se do algoritmo genético para a seleção de um bom conjunto de casos de teste que mate mais mutantes.

Assim, num primeiro momento podemos gerar indivíduos formados por números inteiros que representem os casos de teste. Todos os indivíduos juntos formam a população e passam pela função objetivo, como a dita na Equação (1), para determinar seus valores de escore. A partir desse momento tem-se o Algoritmo Genético mutando os valores, calculando escores e selecionando o indivíduo mais apto, ou seja, o que melhor atende a função objetivo.

Os algoritmos genéticos tradicionais são demorados, assim a paralelização e distribuição, por dividirem a população em pequenas subpopulações que atuam em processadores distintos, tendem a trazer bons resultados em um melhor tempo.

Em Huang 2007, um algoritmo genético distribuído é usado com o objetivo de localizar uma quantidade ótima e boas localizações para turbinas eólicas em um parque eólico. A metodologia abrange o uso de um algoritmo genético comum, no sentido conceitual, em todas as suas etapas, porém dividindo a população em subconjunto, com a tramitação ocasional de um indivíduo de um subconjunto ao outro.

Já em Krauser et al. 1991 se levanta a questão da busca de uma boa estratégia para máquinas SIMD na realização teste de software. São apresentados resultados de simulações de cinco programas de distintas áreas, com a esperança de que, ao se utilizar da unificação de mutantes, se consiga uma aceleração significativa sobre um processador escalar.

Krishnan et al. 2008 faz uso de paralelismo e distribuição de algoritmos genéticos no desenvolvimento baseado em um *framework* de microcontroladores, implementando o

algoritmo genético em chips. Sua intenção é que problemas de recursos limitados ajudem a satisfazer as necessidades de projetos complexos, assim o foco está na ampliação do papel dos microcontroladores PIC.

Não foram localizadas referências sobre a utilização de paralelismo e distribuição, levando em consideração a seleção de casos de teste para Análise de Mutantes. Desta forma, este trabalho quer demonstrar o ganho em tempo de execução que se tem ao se utilizar paralelismo e distribuição.

3. Abordagem Proposta

Apesar de encontrar boas soluções, como dito anteriormente é grande o custo computacional do processo de seleção de um bom grupo de casos de teste via algoritmos evolucionários, como o genético. Para reduzir o esforço empreendido, durante os anos surgiram algumas boas técnicas de redução para a Análise de Mutantes, que podem ser classificadas em dois tipos: redução de mutantes (menos trabalho) e redução de tempo de execução (trabalhar mais rápido) [Jia and Harman 2011].

Sabidamente, processos computacionais podem ser agilizados considerando otimizar os recursos físicos disponíveis, levando à redução do tempo de execução das tarefas. Essa otimização tanto pode ser local, em um computador com múltiplos processadores (paralelização), quanto em rede, quando há duas ou mais máquinas que podem ser orquestradas para o mesmo objetivo (distribuição).

De forma geracional por um algoritmo genético, a população pode ser repartida entre processadores distintos para se ocuparem da evolução e mutação de subconjuntos de indivíduos independentes, sem em nada atrapalhar a busca pelo melhor indivíduo da geração.

O *framework* ECJ[1], escrito em Java, além de facilitar o desenvolvimento de algoritmos evolucionários também fornece meios de paralelização e distribuição desses programas. Os modelos que podem ser implementados são três: *threads*, evolução distribuída e modelo de ilhas [Luke 2013].

Com o recurso *threads* é possível ocupar os processadores da máquina local em etapas de reprodução e mutação, enquanto que, na evolução distribuída, o ECJ envia indivíduos para serem evoluídos em dispositivos remotos. Nele, o computador responsável por orquestrar os trabalhos é chamado *master* e cada nó ao qual envia uma tarefa é conhecido por *slave*. Durante ou antes de iniciar a execução do algoritmo, sem qualquer tipo de risco de perca total do processo, é possível um *master* se conectar a N *slaves*. Demonstrar o acréscimo em performance do modelo de evolução distribuída combinado ao uso de *threads* é o objetivo dos experimentos.

A Figura 1 exemplifica a estrutura utilizada pelo ECJ quando executa um algoritmo evolucionário distribuído. No *master*, o *framework* avalia o problema que recebe, identificando o tipo e seus parâmetros, em seguida inicializa sua interface padrão de problemas evolucionários. A partir deste ponto, a classe que contém realmente o problema proposto é carregada, enquanto o monitoramento de *slave* também acontece.

No monitoramento de *slave*, o ECJ realiza as conexões necessárias com o com-

[1]Disponível em: http://cs.gmu.edu/ eclab/projects/ecj/

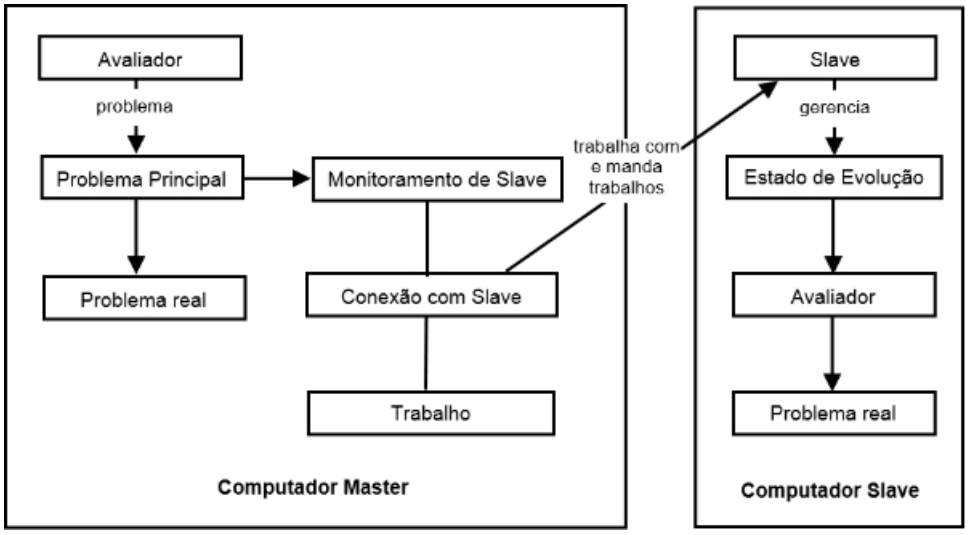


Figura 1. Esquema de distribuição do ECJ [Luke 2013]

putador remoto para troca de informações, e também inicia um trabalho. É chamado de trabalho cada execução de uma configuração sobre um problema real. As configurações do trabalho dizem ao *master* como será dividida a população entre ele e os *slaves*. Além de coordenar a distribuição, o *master* participa do trabalho.

Iniciando, o *slave* se prepara para receber parte da população, interpreta sua tarefa, avalia as configurações do problema e parte para executar o problema real. Ao final de um ciclo geracional, de cada *slave* é retornado ao *master* seu melhor fitness encontrado.

O avaliador identifica se haverá o uso de *threads* e distribuição, ou não, por meio de parâmetros que devem ser configurados pelo usuário do *framework*, na especificação do problema.

Então, por meio de técnicas de SBST, a abordagem vem de encontro ao anseio de unir a eficácia do uso de algoritmo genético na seleção de casos de teste para o teste de mutação, com a eficiência do uso de processamento paralelo e distribuído usando o *framework* ECJ.

4. Estudos Experimentais

Nesta seção apresenta-se os subsídios dos experimentos, os resultados e algumas considerações sobre eles.

4.1. Algoritmos e Benchmarks

A efeito de comparação em eficácia e eficiência, experimentou-se o mesmo problema por três abordagens implementadas:

- Algoritmo Aleatório (AL), onde se escolhe ao acaso casos de teste para formarem um indivíduo até a quantidade de gerações, multiplicada pelo tamanho da população (o mesmo definido para os outros algoritmos), seja satisfeita. Depois, a aptidão dos indivíduos é calculada pela Equação (1) e, por fim, escolhe-se o melhor indivíduo entre todos os submetidos à análise.

- Algoritmo Genético (AG canônico) comum, que seleciona sobre uma população o melhor subconjunto (indivíduo), via alguma métrica pré-definida, até a condição de parada ser satisfeita. O cálculo da aptidão dos indivíduos é dado conforme a Equação (1).
- Algoritmo Genético Paralelo e Distribuído (AGPD) que segue como o AG canônico, acrescentando ao processo métodos de paralelização (threads) e distribuição do trabalho entre dois ou mais computadores, como visto na Seção 3.

O utilitário Cal do UNIX, escritos em linguagem C, foi utilizado para a realização dos experimentos, com seus mutantes gerados por meio da ferramenta Proteum (*PROgram TEsting Using Mutants*) [Delamaro and Maldonado 1996], que fornece o critério de Análise de Mutantes para testes de programas em linguagens de programação procedurais. O Cal é um aplicativo que apresenta um calendário para o ano e/ou mês especificado.

Pode-se observar na Tabela 1 as informações sobre o conjunto de programas mutantes e conjunto de casos de teste considerados na experimentação.

	Cal
Nº de Mutantes	4622
Nº de Equivalentes	344
% de Equivalentes	7,44%
Nº de Casos de Teste	2000
Escore Máximo	0,9981

Tabela 1. Informações sobre os benchmarks utilizados [Oliveira 2013]

4.2. Parâmetros dos Algoritmos

Para os algoritmos genéticos, em todas as instâncias, teve-se como parâmetros genéticos:

- Operador de seleção: Torneio, com 2 competidores;
- Taxa de cruzamento: 95%;
- Taxa de mutação: 5%;
- Elitismo: 1 indivíduo;
- Tamanho da população: 300 indivíduos;
- Condição de parada: 500 gerações.

Para as execuções distribuídas, no *master* definiu-se o máximo de 10 pacotes de indivíduos por *slave*, e tamanho fixo de 100 inteiros para cada pacote.

4.3. Configuração dos Computadores

Os experimentos foram executados em um ambiente uniforme, onde os computadores para *master* e *slaves* contavam com as mesmas configurações:

- Sistema operacional: Ubuntu 14.04 64bits;
- Versão do Java: 1.8.0;
- Versão do ECJ: 21;
- Processador: GenuineIntel 4 CPUs;
- Modelo do processador: Xeon(R) CPU E5504 @ 2,00GHz;
- L2 cache do processador: 4096 KB;
- Memória RAM: 4GB;
- Placa de rede: Broadcom NetXtreme BCM5761 @ 1Gb/s.

4.4. Discussão dos Resultados

Com a experimentação se consegue comparar o AG distribuído com o AG tradicional e o algoritmo aleatório em termos de eficiência e eficácia. Todos os dados foram obtidos via 30 execuções de cada algoritmo, variando-se o tamanho dos indivíduos. As Tabelas 2, 3 e 4 apresentam os resultados.

	MAX	MED	DESVP	TEXEC
Canônico	0,98784	0,98484	0,00163	1744
Aleatório	0,84877	0,83915	0,00556	11
M1S	0,98597	0,98215	0,00191	661
M3S	0,98668	0,98187	0,00243	279

Tabela 2. Resultado do Experimento - Indivíduo de Tamanho 10

	MAX	MED	DESVP	TEXEC
Canônico	0,99532	0,99345	0,00122	3810
Aleatório	0,88425	0,84556	0,00842	16
M1S	0,99369	0,99093	0,00117	741
M3S	0,99252	0,99064	0,00148	416

Tabela 3. Resultado do Experimento - Indivíduo de Tamanho 15

	MAX	MED	DESVP	TEXEC
Canônico	0,99673	0,99547	0,00070	3463
Aleatório	0,88685	0,85443	0,01262	22
M1S	0,99486	0,99328	0,00081	1033
M3S	0,99486	0,99302	0,00104	565

Tabela 4. Resultado do Experimento - Indivíduo de Tamanho 20

Legenda das tabelas:

- M1S: *master* com mais um *slave*;
- M3S: *master* com mais três *slaves*;
- MAX: escore máximo;
- MED: média dos escores;
- DESVP: desvio padrão dos escores;
- TEXEC: média do tempo de execução (em segundos).

Nota-se nas Imagens 2 e 3 que, ao contrário do Algoritmo Aleatório, o uso de Algoritmo Genético na Análise de Mutantes conseguiu bons resultados, e que esses resultados se mantiveram com a paralelização e distribuição entre dois ou mais computadores. Com o acréscimo de recurso computacional, em nada se perde em termos de eficácia, mas se ganha em eficiência.

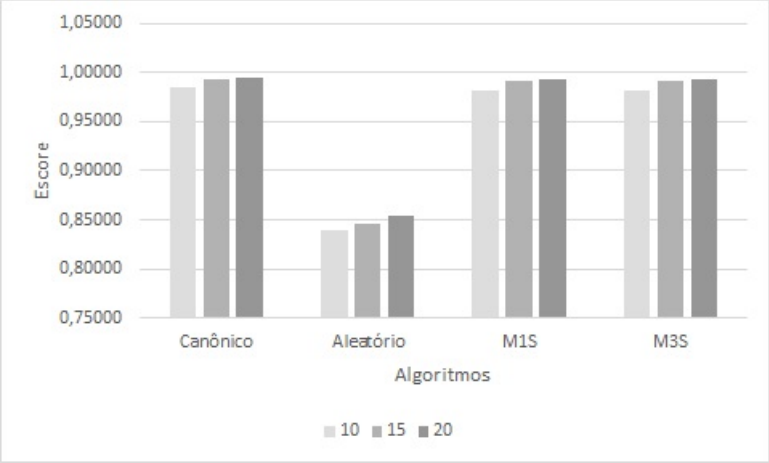


Figura 2. Tamanho do Indivíduo x Escore

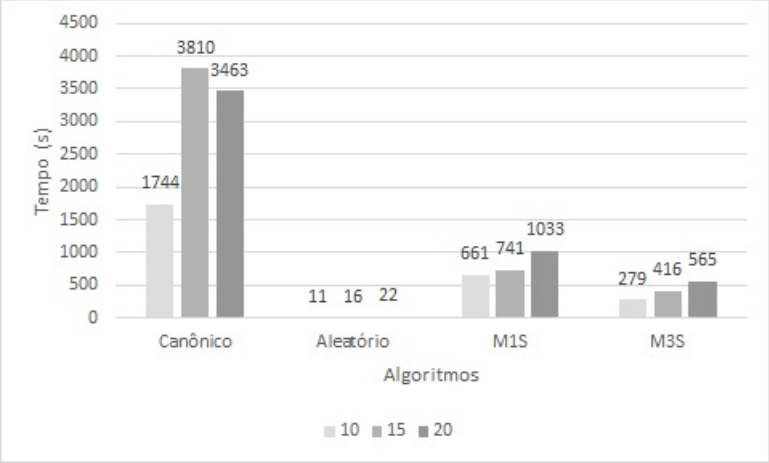


Figura 3. Tamanho do Indivíduo x Tempo

5. Conclusões e Trabalhos Futuros

A partir da implementação e execução dos algoritmos, foi possível verificar que a utilização de paralelismo e distribuição trouxe resultados tão eficazes quanto o AG normal, porém de forma mais eficiente. Assim, valida-se a ideia de se utilizar essa abordagem em problemas de seleção de casos de teste para Análise de Mutantes.

Como trabalhos futuros, pretende-se aprimorar a proposta e, ainda com o uso de distribuição, ganhar em qualidade nas soluções em termos de escore de mutação. Além disso, pretende-se aplicar a abordagem proposta em um número maior de benchmarks que considerem conjuntos de teste com grandes cardinalidades, para o vislumbre dos efeitos desse método em ambientes mais difíceis, se comparada com outras abordagens.

Também pretende-se a utilização de grades computacionais para alcance de melhores resultados. Através de oportunismo, em que seja possível se aproveitar dos recursos computacionais de computadores ociosos da rede, problemas de grande custo poderão ser distribuídos em diferentes grupos base e domínios de forma transparente a quem inicia a tarefa.

Referências

Delamaro, M. E. and Maldonado, J. C. (1996). Proteum-A tool for the assessment of test adequacy for C programs. In *Proceedings of the Conference on Performability in Computing Systems (PCS 96)*, pages 79–95, New Brunswick, NJ.

DeMillo, R. (1989). Test Adequacy And Program Mutation. *11th International Conference on Software Engineering*, pages 355–356.

DeMillo, R., Lipton, R., and Sayward, F. (1978). Hints on test data selection: Help for the practicing programmer. *Computer*, 11(4):34–41.

Huang, H.-S. (2007). Distributed genetic algorithm for optimization of wind farm annual profits. In *Intelligent Systems Applications to Power Systems, 2007. ISAP 2007. International Conference on*, pages 1–6.

IEEE Computer Society, Pierre Bourque, R. E. F. (2014). *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0*. IEEE Computer Society Press, Los Alamitos, CA, USA, 3rd edition.

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

Krauser, E., Mathur, A., and Rego, V. (1991). High performance software testing on simd machines. *Software Engineering, IEEE Transactions on*, 17(5):403–423.

Krishnan, P., Tiong, S. K., and Koh, J. (2008). Parallel distributed genetic algorithm development based on microcontrollers framework. In *Distributed Framework and Applications, 2008. DFmA 2008. First International Conference on*, pages 35–40.

Lima, I. R. S., Yano, T., and Martins, E. (2012). Uso de análise de mutantes para avaliação de uma abordagem de testes baseados em modelos: um estudo exploratório. *Brazilian Workshop on Systematic and Automated Software Testing*.

Luke, S. (2013). The ECJ owner's manual.

McMinn, P. (2011). Search-Based Software Testing: Past, Present and Future. *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*, pages 153–163.

Oliveira, A. A. L. (2013). Uma abordagem coevolucionária para seleção de casos de teste e mutantes no contexto do teste de mutação. Mestrado, Instituto de Informática, UFG.

Pressman, R. (2011). *Engenharia de Software*. McGraw-Hill, São Paulo - SP, 7nd edition.

Sommerville, I. (2007). *Engenharia de Software*. Addison-Wesley, São Paulo - SP, 8nd edition.

Wong, W. E., Mathur, A. P., and Maldonado, J. C. (1995). Mutation versus all-uses: An empirical evaluation of cost, strength and effectiveness. In *Software Quality and Productivity: Theory, Practice and Training*, pages 258–265, London, UK, UK. Chapman & Hall, Ltd.