ANAIS PROCEEDINGS

CBSOFT * 2015

BRAZILIAN CONFERENCE ON SOFTWARE: THEORY AND PRACTICE

BELO HORIZONTE

WESB 2015

VI WORKSHOP DE ENGENHARIA DE SOFTWARE BASEADA EM BUSCA

CBSOFT.ORG

Sponsors:

ThoughtWorks · AvenueCode · Google Brasil · take.net · RARO LABS · Avanti · IBM

Promotion:

SBC Sociedade Brasileira de Computação

Organizing Institutions:

UFMG · PUC Minas · CEFET-MG

CNPq Conselho Nacional de Desenvolvimento Científico e Tecnológico · CAPES · FAPEMIG

WESB 2015

VI WORKSHOP DE ENGENHARIA DE SOFTWARE BASEADA EM BUSCA
September 23rd, 2015
Belo Horizonte – MG, Brazil

VOLUME 01
ISSN: 2178-6097

ANAIS | *PROCEEDINGS*

COORDENADORES DO COMITÊ DE PROGRAMA DO WESB 2015 | *PROGRAM COMMITTEE CHAIR OF WESB 2015*
Maria Cláudia Figueiredo Pereira Emer (UTFPR-Curitiba)
Thelma Elita Colanzi (UEM)

COORDENADORES GERAIS DO CBSOFT 2015 |*CBSOFT 2015 GENERAL CHAIRS*
Eduardo Figueiredo (UFMG)
Fernando Quintão (UFMG)
Kecia Ferreira (CEFET-MG)
Maria Augusta Nelson (PUC-MG)

REALIZAÇÃO | *ORGANIZATION*
Universidade Federal de Minas Gerais (UFMG)
Pontíficia Universidade Católica de Minas Gerais (PUC-MG)
Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG)

PROMOÇÃO |*PROMOTION*
Sociedade Brasileira de Computação | Brazilian Computing Society

APOIO | *SPONSORS*
CAPES, CNPq, FAPEMIG, Google, RaroLabs, Take.net,
ThoughtWorks, AvenueCode, AvantiNegócios e Tecnologia.

APRESENTAÇÃO

Sejam todos bem-vindos ao Workshop de Engenharia de Software Baseada em Busca – WESB 2015. O WESB tem contribuído para o crescimento da área no cenário nacional, tornando-se um fórum de grande importância para discussão e divulgação de pesquisas em temas relacionados à área no Brasil. Nas últimas edições realizadas observou-se que o WESB é fundamental para a formação de uma comunidade de pesquisa em Engenharia de Software Baseada em Busca. Ele tem permitido que pesquisadores, ora trabalhando isoladamente, se conheçam, possam se unir, discutir interesses em comum e propor/realizar projetos de pesquisa. Sendo assim, nesta sexta edição o WESB pretende fortalecer a comunidade de pesquisa recém-criada e contribuir para o crescimento da área no Brasil.

No contexto do WESB, técnicas de busca englobam tanto técnicas tradicionais, como força bruta ou *branch-and-bou*nd, quanto meta-heurísticas, como algoritmos genéticos e outros algoritmos bio-inspirados. 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.

Os trabalhos submetidos para esta sexta edição foram cuidadosamenterevisados por três avaliadores do comitê de programa, que contou compesquisadores de diferentes regiões do país, e com a colaboração de alguns revisoresexternos. Estes anais contêm os trabalhos selecionados dentre as submissõesrecebidas. Ao todo, sete trabalhos completos foram selecionados e serão apresentadosnas três sessões técnicas que integram o evento. Os principais temas abordadosincluem: teste de software, requisitos e linha de produto de software. Além das sessões técnicas, aprogramação também inclui uma palestra convidada e discussões sobre SBSE.

A realização do evento não seria possível sem a colaboração de diversas pessoas. Portanto, gostaríamos de agradecer a todos que auxiliaram para a realização do evento, em especial, aos membros do comitê de programa. Também, gostaríamos de parabenizar a todos os autores dos trabalhos selecionados e de agradecer aos autores de todas as submissões realizadas. Certamente que sem a colaboração de todos não seria possível realizar o evento. Agradecemos também aos organizadores do CBSoft 2015 pela oportunidade, apoio e infraestruturadisponibilizada.

Desejamos a todos um excelente evento e esperamos que o WESB 2015 possa contribuir para estimular e consolidar a Engenharia de Software Baseada em Busca no Brasil.

Belo Horizonte, setembro de 2015.

Maria Cláudia Figueiredo Pereira Emer (UTFPR-Curitiba)

Thelma Elita Colanzi (UEM)

Coordenadoras do Gerais do WESB 2015

COMITÊ DE ORGANIZAÇÃO | *ORGANIZING COMMITTEE*

CBSOFT 2015 GENERAL CHAIRS

Eduardo Figueiredo (UFMG)
Fernando Quintão(UFMG)
Kecia Ferreira (CEFET-MG)
Maria Augusta Nelson (PUC-MG)

CBSOFT 2015 LOCAL COMMITTEE

Carlos Alberto Pietrobon (PUC-MG)
Glívia Angélica Rodrigues Barbosa (CEFET-MG)
Marcelo Werneck Barbosa (PUC-MG)
Humberto Torres Marques Neto (PUC-MG)
Juliana Amaral Baroni de Carvalho (PUC-MG)

WEBSITE AND SUPPORT

Diego Lima (RaroLabs)
Paulo Meirelles (FGA-UnB/CCSL-USP)
Gustavo do Vale (UFMG)
Johnatan Oliveira (UFMG)

COMITÊ TÉCNICO | *TECHNICAL COMMITTEE*

COORDENADORES DO COMITÊ DE PROGRAMA DO WESB 2015 | *2015 | PROGRAM COMMITTEE CHAIR OF WESB 2015*

Maria Cláudia Figueiredo Pereira Emer (UTFPR-Curitiba)
Thelma Elita Colanzi (UEM)

COMITÊ DIRETIVO | *STEERING COMMITTEE*

Auri Marcelo Rizzo Vincenzi (UFG)
Celso Gonçalves Camilo Junior (UFG)
Maria Cláudia Figueiredo Pereira Emer (UTFPR-Curitiba)
Thelma Elita Colanzi (UEM)
Silvia Regina Vergílio (UFPR)

COMITÊ DE PROGRAMA | *PROGRAM COMMITTEE*

Adriana C. F. Alvim (UNIRIO)
Arilo Claudio Dias Neto (UFAM)
Auri Marcelo Rizzo Vincenzi (INF/UFG)
Cássio Leonardo Rodrigues (INF/UFG)
Celso G. Camilo-Junior (INF/UFG)
Eliane Martins (IC/Unicamp)
Geraldo Robson Mateus (UFMG)
Gledson Elias (UFPB)
Jerffeson Teixeira de Souza (UECE)
Leila Silva (UFS)
Márcio Eduardo Delamaro (ICMC/USP)
Márcio de Oliveira Barros (UNIRIO)
Maria Cláudia Figueiredo Pereira Emer (UTFPR)
Mel Ó Cinnéide (University College Dublin, IE (Irlanda))
Pedro de Alcântara dos Santos Neto (UFPI)
Plínio de Sá Leitão Júnior (INF/UFG)
Silvia Regina Vergilio (UFPR)
Thelma Elita Colanzi (UEM)

REVISORES EXTERNOS | *EXTERNAL REVIEWERS*

Eduardo Freitas, IFG
Kenyo Faria, IFG
Renata Rego, UFAM

PALESTRAS CONVIDADAS | *INVITED TALKS*

Estudos Experimentais com Algoritmos de Otimização em Engenharia de Software

Márcio de Oliveira Barros(UNIRIO)

Resumo: Nesta palestra apresentaremos a área de Engenharia de Software baseada em Buscas como um excelente domínio para a realização de estudos experimentais relacionados com problemas da Engenharia de Software. A realização de estudos experimentais dentro da Engenharia de Software é um assunto relativamente recente. Embora os primeiros estudos já tenham mais de três décadas, o uso deste recurso de pesquisa para identificação de evidências se intensificou nos últimos 10 anos. Existem diversas razões para esta lenta adoção dos estudos experimentais na Engenharia de Software, mas muitas destas razões deixam de existir quando tratamos da Engenharia de Software baseada em Buscas. Sendo assim abordaremos as razões pelas quais estudos com algoritmos heurísticos tendem a ser mais simples do que em outras áreas da Engenharia de Software e veremos as técnicas mais comumente utilizadas nestes estudos.

Márcio de Oliveira Barros possui graduação em Informática pelo IM/UFRJ (1992), Mestrado em Engenharia de Sistemas e Computação pela COPPE/UFRJ (1995) e doutorado em Engenharia de Sistemas e Computação pela COPPE/UFRJ (2001). Atualmente é Professor Associado da UNIRIO. Atua como revisor de diversos periódicos da área de Engenharia de Software, como o Journal of Systems and Software, JSERD, JSEP, ESE, IJSEKE, TOSEM e JUCS. Atua também como revisor de projetos de fomento do CNPq, onde possui uma bolsa de Produtividade em Pesquisa N2. Tem experiência na área de Ciência da Computação, com ênfase em Engenharia de Software e atua em pesquisa nos seguintes temas: otimização heurística na Engenharia de Software (SBSE), software design, gerência de projetos, modelagem e simulação de projetos de software.

ARTIGOS TÉCNICOS | *TECHNICAL RESEARCH PAPERS*

SESSÃO 1: Hiper-Heurísticas

SESSÃO 2: Requisitos

SESSÃO 3: Linha de Produto de Software

Uma hiper-heurística de seleção de meta-heurísticas para estabelecer sequências de módulos para o teste de software

Vinicius Renan de Carvalho[1], **Silvia Regina Vergilio**[1], **Aurora Pozo**[1]

[1]DInf-UFPR, Centro Politécnico, Jardim das Américas, CP 19081,
CEP 19031-970, Curitiba - PR. (UFPR)

Resumo. *Meta-heurísticas, tais como algoritmos evolutivos multi-objetivos, são utilizadas para estabelecer sequências de módulos para o teste de integração, a fim de minimizar o custo relacionado à construção de stubs. Contudo, resultados da literatura mostram que nenhum algoritmo pode ser considerado o melhor para qualquer sistema e contexto. Muitas vezes é necessário configurar diferentes algoritmos e realizar experimentos para determinar qual é o melhor. Para evitar esta tarefa e reduzir o esforço do testador, este trabalho propõe uma hiper-heurística, chamada MOCAITO-HH, que utiliza o método de seleção Choice Function para selecionar, em tempo de execução, o melhor algoritmo com o melhor desempenho em um dado momento e sistema, considerando diferentes indicadores da área de otimização. O uso da hiper-heurística obteve resultados equivalentes estatisticamente ao melhor algoritmo, sem que seja necessária a escolha da meta-heurística por parte do testador.*

Abstract. *Meta-heuristics, such as multi-objective and evolutionary algorithms, have been successfully used to establish sequence of modules for the integration testing with minimal stubbing cost. However, results from the literature show that no algorithm has been obtained the best performance for any system and context. Many times it is necessary to configure different algorithms and to conduct experiments to determine the best one. To avoid this and reducing tester's efforts, this work introduces a hyper-heuristic, named MOCAITO-HH, which uses the method Choice Function to select, in execution time, the algorithm with the best performance in a given moment and system, according to common quality indicators of the optimization field. The use of the hyper-heuristic obtained results that are statistically equivalent to the best algorithm, without requiring the tester's choice of a meta-heuristic.*

1. Introdução

O problema de determinar uma sequência de módulos para o teste de integração visa a determinar uma ordem em que os módulos de um sistema devem ser integrados e testados, que possua o menor custo possível, relacionado à construção de *stubs*. Isto pode ser feito decidindo qual *stub* deve ser implementado primeiro. Esta tarefa não é uma tarefa trivial, pois o custo é influenciado por diversos fatores.

Diferentes abordagens são propostas na literatura para resolver este problema. Dentre estas, destacam-se abordagens baseadas em busca [Wang et al. 2011], tais como a abordagem MOCAITO [Assunção et al. 2014] que trata este problema como um problema de muitos objetivos, e por esta razão, a ser otimizado com algoritmos multi-objetivos. A abordagem inclui passos para a escolha dos objetivos e do algoritmo de

otimização (meta-heurística), entretanto não oferece nenhum suporte para auxiliar o testador nesta tarefa. Nos experimentos conduzidos com algoritmos evolutivos multi-objetivos (MOEAs), e relatados em [Assunção et al. 2014], observou-se que nenhum algoritmo se mostrou o melhor para todos os sistemas. A escolha pode depender de características e interdependências existentes nos módulos dos sistemas e esta escolha pode implicar na condução de experimentos para a determinar o melhor algoritmo. Isto é uma tarefa difícil para testadores que não têm conhecimento na área de Engenharia de Software Baseada em Busca (SBSE) e que consome bastante tempo e esforço.

Em tais situações, o uso de hiper-heurísticas tem despertado interesse da comunidade de SBSE mais recentemente [Harman et al. 2012]. Ainda são poucos os trabalhos que tratam deste tópico. Segundo Burke et al. (2013), hiper-heurística pode ser definida como: (i) metodologias de seleção de heurísticas: heurísticas para escolher heurísticas, e (ii) metodologias de geração de heurísticas: heurísticas para gerar heurísticas. Ainda são poucos os trabalhos que aplicam hiper-heurísticas em SBSE [Kaelbling et al. 1996, Basgalupp et al. 2013, Guizzo et al. 2015a, Jia et al. 2015]. No contexto do problema de integração e teste de módulos, destaca-se a hiper-heurística HITO [Guizzo et al. 2015a] com o objetivo de auxiliar o testador na escolha de operadores, tais como o de cruzamento e mutação, que melhor resolvem o problema. HITO foi implementada com o algoritmo NSGA-II e apresentou melhores resultados quando comparada ao algoritmo tradicional. Entretanto, HITO não auxilia na escolha do algoritmo multi-objetivo mais adequado ao sistema sendo testado.

Com o objetivo de facilitar esta escolha e evitar esforço comparando diferentes algoritmos, este trabalho propõe uma hiper-heurística, chamada MOCAITO-HH, que trabalha com a abordagem MOCAITO para resolver o problema de determinação de sequências de teste, e que utiliza a função de seleção CF (*Choice Function* [Cowling et al. 2001]), que permite escolher durante a otimização o melhor algoritmo em um dado momento, dentre os algoritmos evolutivos: NSGA-II, SPEA-2 e IBEA. MOCAITO-HH utiliza um ranqueamento considerando o desempenho dos algoritmos de acordo com os indicadores *Hypervolume*, *Spread*, *Algorithm Effort* (AE), *Ratio of Non-dominated Individuals* (RNI), e um mecanismo para troca de população que permite a alternância de execução entre os algoritmos. Resultados obtidos pela hiper-heurística mostram equivalência estatística com a execução de um único (o melhor) algoritmo em cada caso, mas com a vantagem de não ser necessário realizar experimentos para saber qual é o melhor.

O trabalho está organizado como segue. Na Seção 2 a abordagem MOCAITO é brevemente descrita. Após isso, é fornecida uma introdução à área de hiper-heurísticas e a função de seleção CF (Seção 3). A hiper-heurística é proposta na Seção 4 e resultados de sua avaliação são apresentados na Seção 5. A Seção 6 contém trabalhos relacionados e a Seção 7 apresenta as considerações finais.

2. Abordagem MOCAITO

A atividade de teste é geralmente realizada em fases. O teste de unidade, por exemplo, visa a testar cada módulo de software separadamente. Depois disto, estes módulos são integrados e testados. Na maioria das vezes os módulos não são integrados de uma vez. Essa integração acontece em partes e, por isso a ordem de integração de módulos pode

depender da disponibilidade prévia de um outro módulo, do qual eles dependem. Desta forma, faz-se necessária a determinação da ordem com que os elementos devem ser construídos e testados, e caso necessário a construção de um *stub*. Criar um *stub* tem um custo, este custo deve ser minimizado por meio de alguma estratégia que decida qual *stub* deve ser implementado primeiro, esta tarefa não é uma tarefa trivial devido a conflitos de restrições, fatores que impactam a criação de um *stub* e possíveis restrições contratuais.

Dentre as diversas abordagens existentes para o problema [Wang et al. 2011], destaca-se a abordagem MOCAITO (*Multi-objective Optimization and Coupling-based Approach for the Integration and Test Order problem*), de Assunção et al. (2014). Esta abordagem foi escolhida devido ao fato de trabalhar com algoritmos multi-objetivos. A abordagem MOCAITO permite o uso de diferentes métricas e meta-heurísticas, fazendo desta uma abordagem genérica e abrangente. Duas entradas devem ser fornecidas. A primeira é a informação sobre os relacionamentos de dependência entre os módulos. No contexto de software orientado a objetos (OO) os módulos são classes, e a dependência entre as classes é dada por um diagrama chamado ORD (*Object Relation Diagram*). No contexto de software Orientado a Aspectos, este diagrama é estendido para conter também as dependências entre aspectos e entre classes e aspectos. A segunda entrada é a informação sobre o custo envolvido na construção de um *stub*, seus valores são utilizados pela função de aptidão do algoritmo (objetivos) [Assunção et al. 2014]. Estas entradas são utilizadas em conjunto pelo algoritmo de otimização multi-objetivo pois a informação sobre os relacionamentos é necessária para que a minimização do custo envolvido na criação não viole as restrições do problema a ser trabalhado. Ao final o testador deve escolher a melhor solução de acordo com os recursos do teste.

Em [Assunção et al. 2014] a abordagem MOCAITO foi avaliada em experimentos com diferentes sistemas e com os algoritmos NSGA-II, SPEA2 e PAES, e os operadores *Two Points Crossover* e *Swap Mutation* como operadores de cruzamento e mutação, respectivamente. Dois modelos de custo foram utilizados: um com dois objetivos, composto do número de atributos (A) e do número de operações (O) que precisam se emulados caso o *stub* seja construído; e um com quatro objetivos, que utiliza além das medidas A e O, o número de tipos distintos de retorno (R) e número de tipos distintos de parâmetro (P) envolvidos nas operações. Os resultados obtidos foram promissores, mas nenhum algoritmo provou ser o melhor para todos os sistemas. Para que não seja necessária a escolha de tal algoritmo este artigo propõe uma abordagem baseada em hiper-heurística.

3. Hiper-heurísticas

Meta-heurísticas têm sido amplamente utilizadas na busca de soluções de diversos problemas, contudo conforme vão surgindo aplicações de meta-heurísticas em novos problemas, tem-se a dificuldade de determinar qual combinação de meta-heurísticas e operadores seriam os mais efetivos para um dado problema. Neste contexto, o conceito de hiper-heurística surge como uma possível solução para que um processo de busca seja guiado de forma que seja aplicada automaticamente a melhor heurística de baixo nível (*Low Level Heuristic* - LLH) [Sabar et al. 2015].

Segundo Cowling et al. (2001) hiper-heurísticas são heurísticas que escolhem heurísticas. Posteriormente devido à expansão da área, uma nova definição foi proposta por Burke et al.(2013), hiper-heurísticas portanto são: (i) metodologias de seleção de

heurísticas; e/ou (ii) metodologias de geração de heurísticas. Assim a escolha de uma LLH é realizada comparando os resultados obtidos com resultados anteriores, baseando-se em algum indicador de qualidade como *Hypervolume*, etc. As LLH podem ser operadores (como por exemplo operadores de mutação e cruzamento), ou meta-heurísticas (como NSGA-II, SPEA2, etc.).

Neste trabalho, as LLH são meta-heurísticas, algoritmos de otimização multi-objetivos a serem escolhidos automaticamente. É utilizado um método de seleção de heurísticas conhecido como *Choice Function* (CF) [Cowling et al. 2001] por sua simplicidade de implementação e melhora geralmente proporcionada por sua aplicação.

3.1. Choice Function

O método de *Choice Function* (CF) foi inicialmente proposto por Cowling et al. (2001) e tem como objetivo escolher uma LLH de acordo com um ranqueamento. Para isto, este método adaptativamente classifica cada LLH com relação a uma pontuação [Burke et al. 2013]. Esta pontuação leva em consideração quão bom foram os resultados da execução da LLH, a melhora obtida de sua execução em relação à executada previamente, e o tempo de espera desde sua última utilização. Maashi et al. (2014) propuseram uma simplificação para o método CF, implementando um sistema que utiliza dois ranqueamentos baseados em indicadores (Figura 1). O primeiro ranqueamento é criado a partir dos valores obtidos, para cada heurística h_i, dos indicadores de *Hypervolume* (H), *Algorithm Effort* (AE), *Ratio of Non-dominated Individuals* (RNI) e *Spread* (S), ordenados do melhor para o pior, e posteriormente classificados de acordo com a quantidade de vezes que um dado algoritmo obteve o melhor resultado frente aos outros, ou seja, obteve a primeira colocação em algum indicador. A partir do primeiro ranqueamento é criado o segundo, que tem como função ordenar os elementos do melhor para o pior atribuindo-lhes uma colocação $Freq_{rank}$.

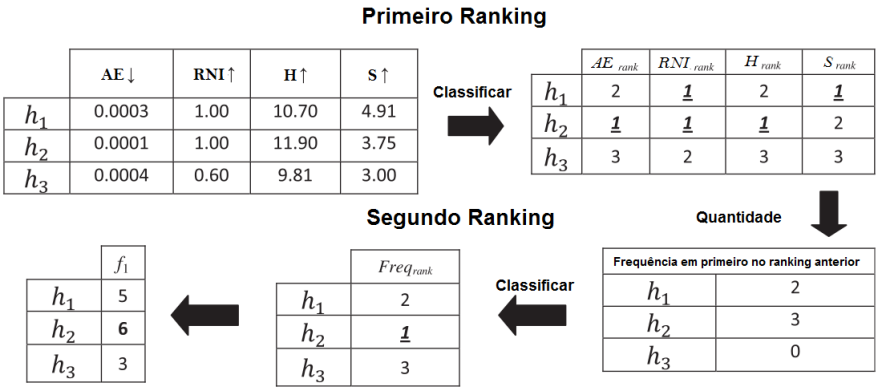


Figura 1. Ranqueamento em dois níveis, adaptada de [Maashi et al. 2014]

A função de seleção é então dada pela Equação 1:

$$F(h_i) = \alpha f_1(h_i) + \beta f_2(h_i) \tag{1}$$

onde α e β são parâmetros que regulam o impacto de cada item da equação.

O valor de f_1 é dado pela Equação 2 trabalhando-se diretamente com as variáveis $Freq_{rank}$ (obtida através do segundo ranqueamento (Figura 1)), RNI_{rank} (obtido através

do ranqueamento do maior RNI para o menor) e *N* (quantidade de LLHs). A adição do valor de RNI na Equação 2 permite que o algoritmo não olhe apenas para os melhores resultados, mas também busque um maior número de soluções não dominadas.

$$f_1(h_i) = 2 * (N + 1) - (Freq_{rank}(h_i) + RNI_{rank}(h_i)) \quad (2)$$

O elemento $f_2(h_i)$ é o tempo (em segundos) decorrido desde a última execução da heurística h_i. A heurística que maximiza CF é escolhida.

No trabalho de Maashi et al. (2014) as meta-heurísticas NSGA-II, SPEA2 e MOGA foram usadas como LLH para solucionar o problema *Walking Fish Group (WFG) benchmark* [Huband et al. 2006]. Devido aos bons resultados obtidos, esta abordagem foi empregada neste trabalho como descrito na próxima seção.

4. MOCAITO-HH

A MOCAITO-HH (*MOCAITO using Hyper Heuristics*), combina o método *Choice Function* proposto em [Maashi et al. 2014] com a abordagem MOCAITO [Assunção et al. 2014]. Para isto, aplica a CF por classificação usando as métricas *Hypervolume*, AE, RNI e *Spread*. A MOCAITO-HH utiliza o algoritmos como sendo as heurísticas de baixo nível (LLH), sendo estas NSGA-II, SPEA2 e IBEA, todos implementados no jMetal[1].

A Figura 2 mostra como se dá a transferência das populações entre os algoritmos. No caso de uma transição do algoritmo NSGA-II para SPEA2/IBEA, a população é atribuída normalmente e o arquivo recebe o subconjunto de soluções não dominadas. No caso da transição de SPEA2/IBEA para NSGA-II, uma união é realizada entre a população e o arquivo provenientes de SPEA2/IBEA, e um subconjunto é selecionado com ajuda do mecanismo *Crowding Distance* do NSGA-II, posteriormente, este subconjunto é atribuído como população no NSGA-II.

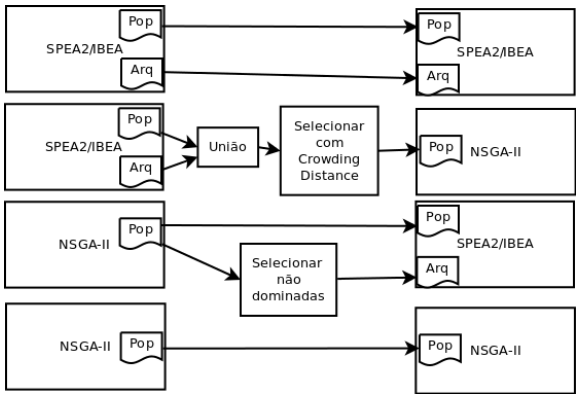


Figura 2. Tipos de passagem de população

Os cálculos dos indicadores *Hypervolume* e *Spread* são feitos como base uma dada população e um ponto de referência, sendo este ponto a pior solução possível para o problema a ser otimizado. O RNI representa o percentual de soluções não dominadas

[1]Neste trabalho, diferentemente do trabalho de Assunção et al. (2014) , não foi utilizado o algoritmo PAES, pelo fato de este não ser um algoritmo populacional. Em seu lugar foi escolhido então o IBEA.

da população em questão. O AE representa o tempo computacional necessário para a LLH executar. Depois de efetuado o cálculo dos indicadores, os ranqueamentos são realizados, com base na Equação 2, e obtido o $F(h_i)$ para cada LLH levando também em consideração o $f_2(h_i)$, que é o tempo em segundos que uma dada LLH ficou sem atividade (representado pelo vetor $EstimatedTimeWaiting$) . Por fim, o maior valor de $F(h_i)$ é selecionado e a LLH em questão é aplicada. Para isto recebe a população LLH vigente.

Algoritmo 1: Pseudocódigo da hiper-heurística MOCAITO-HH na escolha de algoritmos

```
1  Entrada: A - Problema a ser trabalhado; W - tamanho da janela
2  Saída: O problema trabalhado A.
3  início
4      Algs ← Inicializa algoritmos;
5      enquanto Todos algoritmos não forem executados faça
6          Alg ←Algs;
7          Alg.atribuirProblema(A);
8          enquanto O tamanho da janela W não for atingido faça
9              Alg.execute();
10         fim
11     fim
12     EstimatedTimeWaiting ← Inicia vetor com 0 para cada LLH;
13     Hypervolumes ← Calcula Hypervolume para cada algoritmo;
14     RNIs ← Calcula RNI para cada algoritmo;
15     Spreads ← Calcula Spread para cada algoritmo;
16     AEs ← Calcula AE para cada algoritmo;
17     Ranking ← CalculaRanking(Algs, Hypervolumes, RNIs, Spreads, AEs);
18     Alg ← ChoiceFunction(Ranking, EstimatedTimeWaiting, RNIs);
19     enquanto A quantidade máxima de avaliações não for realizada faça
20         enquanto O tamanho da janela W não for atingido faça
21             Alg.execute();
22         fim
23         EstimatedTimeWaiting ← Incrementa EstimatedTimeWaiting de outros algoritmos;
24         Hypervolumes ← Calcula Hypervolume para cada algoritmo;
25         RNIs ← Calcula RNI para cada algoritmo;
26         Spreads ← Calcula Spread para cada algoritmo;
27         AEs ← Calcula AE para o algoritmo Alg;
28         Ranking ← CalculaRanking(Algs, Hypervolumes, RNIs, Spreads, AEs);
29         proxAlg ← ChoiceFunction(Ranking, EstimatedTimeWaiting, RNIs);
30         proxAlg.setPopulation(Alg.getPopulation);
31         Alg ←proxAlg;
32     fim
33     A ← Alg.obterResultado();
34     retorna A;
35 fim
```

O Algoritmo 1 mostra o comportamento da MOCAITO-HH na escolha de algoritmos. Primeiramente os algoritmos NSGA-II, IBEA e SPEA2 são inicializados e recebem uma mesma população inicial aleatória. Em seguida, na linha 8 cada um dos algoritmos é executado por *W* avaliações para que posteriormente os vetores de *Hypervolume*, *Spread*, AE e RNI sejam preenchidos de forma que cada posição destes vetores representem um dos algoritmos. A abordagem de dois ranqueamentos [Maashi et al. 2014] é utilizada, e o vetor *Ranking* é gerado (linha 17), que juntamente com o vetor *RNIs*, são utilizados pela CF para que o algoritmo *Alg* seja escolhido.

Na linha 19 do pseudocódigo inicia-se uma repetição que controla a quantidade de avaliações que devem ser executadas. Na linha 20 o mesmo processo da linha 8 é realizado, contudo este processo é executado apenas para o algoritmo *Alg*. Na linha 23 o incremento dos itens do vetor *EstimatedTimeWaiting* é realizado para todos os algoritmos diferentes de *Alg* e, posteriormente, os vetores de *Hypervolume*, *Spread*, AE, RNI e

Ranking são recalculados. A variável *proxAlg* recebe o algoritmo escolhido pela CF na linha 29, para que posteriormente, na linha 30, seja realizada a passagem de população do algoritmo *Alg* (população corrente) para o algoritmo *proxAlg*.

5. Avaliação da Hiper-heurística

Para avaliar a MOCAITO-HH foram consideradas duas instâncias do problema: uma com 2 objetivos (A e O) e outra com 4 (A,O,R,P) (ver Seção 2). Foram utilizados os sistemas mais complexos descritos em [Assunção et al. 2014] e na Tabela 1, ou seja, para os quais houve diferença no comportamento dos algoritmos.

Foram utilizados os parâmetros da Tabela 2, conforme usado no trabalho de Assunção et al.; α, β e W foram obtidos empiricamente, testando-se diferentes valores. Os algoritmos e a hiper-heurística foram executados 30 vezes. Para cada execução foram obtidas a quantidade de soluções não dominadas e realizado o cálculo do *Hypervolume* para a população final obtida. Após a execução, foram calculadas a médias das 30 execuções para a métrica *Hypervolume* e para a quantidade de soluções não dominadas. Em seguida o teste estatístico Kruskal-Wallis [Derrac et al. 2011] foi aplicado com a finalidade de verificar equivalências estatísticas (significância de 95%).

5.1. Resultados

As Tabelas 3 e 4 mostram os resultados para cada uma das meta-heurísticas individualmente comparados ao resultado do MOCAITO-HH. A coluna Id se refere à identificação do sistema usado, Hyp a média do *Hypervolume* com seu desvio padrão em parenteses, e a média da quantidade de soluções não dominadas e seu desvio padrão. Valores em negrito representam igualdade estatística segundo o teste Kruskal-Wallis.

Tabela 1. Sistemas Utilizados no Experimento

Id	Sistema	Linguagem	Versão	LOC	Classes	Aspectos	Dependências
1	AJHotDraw	AspectJ	0.4	18586	290	31	1592
2	AJHSQLDB	AspectJ	18	68550	276	25	1338
3	BCEL	Java	5.0	2999	45	-	289
4	MyBatis	Java	3.0.2.2	23535	331	-	1271

Tabela 2. Parâmetros do experimento

	Aval.	População/Arquivo	Cruzamento	Mutação	α	β	W
MOCAITO	60000	300	95%	2%	-	-	-
MOCAITO-HH	60000	300	95%	2%	1	0.1	600

Os resultados mostram que a MOCAITO-HH obteve, em todos os casos, resultados de Hyp equivalentes aos resultados do melhor algoritmo, independentemente do número de objetivos utilizados. Com relação ao tempo de execução os resultados mostraram que o NSGA-II obteve o menor tempo de execução, seguido pelo IBEA, MOCAITO-HH e SPEA2. O fator custo deverá ser objeto de estudos futuros, considerando o uso da MOCAITO-HH em diferentes problemas.

As Figuras 3 e 4 mostram, respectivamente, o percentual de escolha de cada meta-heurística pelo MOCAITO-HH em cada um dos problemas testados. Percebe-se que o

Tabela 3. Resultados para 2 objetivos para MOCAITO-IBEA, MOCAITO-NSGA-II, MOCAITO-SPEA2 e MOCAITO-HH

ID		IBEA	NSGA-II	SPEA2	MOCAITO-HH
1	Hyp	9.63E-1 (4.14E-3)	**9.67E-1 (5.00E-3)**	**9.65E-1 (3.80E-3)**	**9.67E-1 (5.05E-3)**
	Qtd	3.87 (1.50)	**4.40 (1.81)**	**4.83 (2.19)**	**5.10 (2.25)**
2	Hyp	8.59E-1 (8.19E-3)	**8.71E-1 (9.24E-3)**	**8.65E-1 (9.98E-3)**	**8.64E-1 (1.07E-2)**
	Qtd	17.93 (7.31)	**32.20 (7.83)**	**27.80 (10.97)**	25.40 (8.39)
3	Hyp	**7.78E-1 (3.33E-3)**	**7.79E-1 (3.32E-3)**	**7.77E-1 (5.44E-3)**	**7.77E-1 (3.90E-3)**
	Qtd	**27.46 (1.16)**	**28.87 (0.35)**	**28.77 (0.86)**	**28.60 (0.6215)**
4	Hyp	9.05E-1 (6.95E-3)	**9.10E-1 (6.59E-3)**	**9.11E-1 (7.16E-3)**	**9.10E-1 (6.21E-3)**
	Qtd	47.30 (4.45)	**60.84 (8.43)**	**57.20 (5.86)**	**55.43 (6.59)**

Tabela 4. Resultados para 4 objetivos para MOCAITO-IBEA, MOCAITO-NSGA-II, MOCAITO-SPEA2 e MOCAITO-HH

ID		IBEA	NSGA-II	SPEA2	MOCAITO-HH
1	Hyp	9.34E-1 (4.56E-3)	**9.48E-1 (4.47E-3)**	**9.47E-1 (5.03E-3)**	**9.45E-1 (5.33E-3)**
	Qtd	7.63 (2.95)	**65.26 (11.35)**	**74.56 (24.28)**	**57.80 (19.50)**
2	Hyp	7.17E-1 (1.91E-2)	**7.35E-1 (1.72E-2)**	**7.33E-1 (1.67E-2)**	**7.31E-1 (1.81E-2)**
	Qtd	33.40 (18.53)	**164.73 (37.95)**	122.73 (33.13)	96.66 (32.83)
3	Hyp	4.92E-1 (5.79E-3)	**5.01E-1 (1.64E-3)**	**5.01E-1 (3.01E-4)**	**5.01E-1 (5.87E-4)**
	Qtd	19.60 (2.14)	**36.20 (1.44)**	**36.33 (1.27)**	**36.33 (1.88)**
4	Hyp	7.82E-1 (1.25E-2)	8.28E-1 (1.34E-2)	**8.25E-1 (1.10E-2)**	**8.22E-1 (1.01E-2)**
	Qtd	38.47 (13.64)	**276.74 (7.15)**	248.73 (5.50)	195.90 (53.05)

NSGA-II foi a meta-heurística mais escolhida pela MOCAITO-HH, e a menos utilizada foi a IBEA, o algoritmo que apresentou os piores resultados. Isto mostra a capacidade da hiper-heurística em escolher a meta-heurística mais adequada.

6. Trabalhos Relacionados

O uso de hiper-heurísticas desperta interesse da comunidade de SBSE [Harman et al. 2012], mas ainda são poucos os trabalhos que tratam deste tópico.

Jia et al. (2015) introduziram uma hiper-heurística para aprender e aplicar estratégias de teste combinatorial. O objetivo é obter uma algoritmo genérico para aplicar este tipo de teste. Basgalupp et al. (2013) propuseram uma hiper-heurística para a geração de algoritmos que criam árvores de decisão, onde estas árvores de decisão fo-

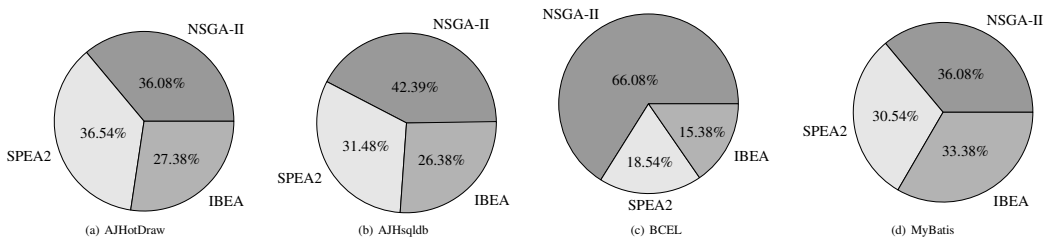


Figura 3. Percentual de escolha para 2 objetivos.

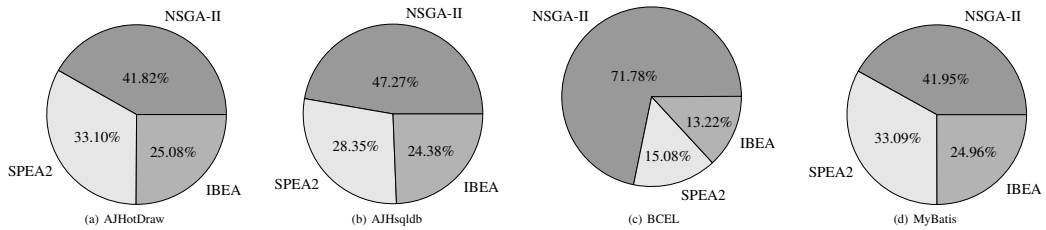


Figura 4. Percentual de escolha para 4 objetivos.

ram utilizadas na predição de esforço software. Kumari e Srinivas (2013) propuseram uma hiper-heurística para trabalhar com o problema de agrupamento de módulos, a fim de selecionar LLHs tais como operadores de mutação e crossover. Com este mesmo objetivo, o trabalho de Guizzo et al. (2015a) introduziu uma hiper-heurística, chamada HITO, que seleciona LLHs a serem usadas com o problema de teste e integração de classes e aspectos. Este trabalho aplica hiper-heurísticas para o mesmo problema abordado neste trabalho, entretanto HITO seleciona dentre um conjunto de 9 operadores de busca os mais adequados ao problema. Estes operadores são as LLHs. Já o trabalho de Guizzo et al. (2015b) aplica a abordagem HITO utilizando NSGA-II no problema de ordens de teste na presença de restrições de modularização, este trabalho seleciona dentre um conjunto de 20 operadores de busca os mais adequados ao problema.

No presente trabalho, as LLHs não são os operadores de busca e sim as meta-heurísticas. Neste sentido as abordagens tem diferentes objetivos e podem ser usadas de maneira complementar. Isto deverá ser investigado em trabalhos futuros.

7. Conclusão

Este trabalho introduziu a hiper-heurística MOCAITO-HH que tem como objetivo selecionar em tempo de execução, durante o processo de otimização, um algoritmo de otimização multi-objetivo para determinar a sequência de módulos para o teste de integração associada ao menor custo para construção de *stubs*.

MOCAITO-HH utiliza a função de seleção CF, baseada em ranqueamento de acordo com o desempenho dos algoritmos considerando os indicadores *Hypervolume*, *Spread*, AE e RNI. Além disso, é considerado o tempo decorrido desde a última vez que um dado algoritmo foi executado. Os resultados da avaliação mostram que a hiper-heurística é capaz de obter resultados equivalentes estatisticamente ao melhor algoritmo, sem que seja necessário realizar experimentos para determinar qual o mais adequado em cada sistema. Isso reduz o esforço do testador que não precisa optar por um algoritmo.

Como trabalhos futuros pretende-se avaliar o uso de MOCAITO-HH para outros sistemas e realizar um estudo experimental da abordagem em empresas de software. Além disso, poderão ser implementadas e avaliadas outras funções de seleção tais como o MAB (Multi-Armed Bandit [Fialho et al. 2010]). MOCAITO-HH também poderá ser utilizada para resolver outros problemas da Engenharia de Software.

Referências

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

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

Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., and Qu, R. (2013). Hyper-heuristics. *J. of the Operational Research Society*, 64(12):1695–1724.

Cowling, P. I., Kendall, G., and Soubeiga, E. (2001). A hyperheuristic approach to scheduling a sales summit. In *Selected Papers from the Third International Conference on Practice and Theory of Automated Timetabling*, pages 176–190.

Derrac, J., Garcia, S., Molina, D., and Herrera, F. (2011). A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1:3–18.

Fialho, A., Da Costa, L., Schoenauer, M., and Sebag, M. (2010). Analyzing bandit-based adaptive operator selection mechanisms. *Annals of Mathematics and Artificial Intelligence*, 60(1-2):25–64.

Guizzo, G., Fritsche, G. M., Vergilio, S. R., and Pozo, A. T. R. (2015a). A hyper-heuristic for the multi-objective integration and test order problem. In *Proceedings of the 24th Genetic and Evolutionary Computation Conference (GECCO'15)*. ACM.

Guizzo, G., Vergilio, S. R., and Pozo, A. T. R. (2015b). Uma solução baseada em hiper-heurística para determinar ordens de teste na presença de restrições de modularização. In *Workshop de Engenharia de Software Baseada em Busca*.

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

Huband, S., Hingston, P., Barone, L., and While, L. (2006). A review of multiobjective test problems and a scalable test problem toolkit. *IEEE Transactions on Evolutionary Computation*, 10(5):477–506.

Jia, Y., Cohen, M., Harman, M., and Petke, J. (2015). Learning combinatorial interaction test generation strategies using hyperheuristic search. In *Proceedings of the 37th International Conference on Software Engineering (ICSE'15)*.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, 4:237–285.

Kumari, A. C., Srinivas, K., and Gupta, M. P. (2013). Software module clustering using a hyper-heuristic based multi-objective genetic algorithm. In *3rd International Advance Computing Conference (IACC'13)*, pages 813–818.

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

Sabar, N., Ayob, M., Kendall, G., and Qu, R. (2015). A dynamic multiarmed bandit-gene expression programming hyper-heuristic for combinatorial optimization problems. *IEEE Transactions on Cybernetics*, 45(2):217–228.

Wang, Z., Li, B., Wang, L., and Li, Q. (2011). A brief survey on automatic integration test order generation. In *Proceedings of the 23rd Software Engineering and Knowledge Engineering*, pages 254–257.

Uma Solução Baseada em Hiper-Heurística para Determinar Ordens de Teste na Presença de Restrições de Modularização

Giovani Guizzo, Silvia Regina Vergilio, Aurora Trinidad Ramirez Pozo*

[1]DInf - Universidade Federal do Paraná (UFPR) – Curitiba, PR – Brasil
CP:19081, CEP: 19031-970

{gguizzo,silvia,aurora}@inf.ufpr.br

Abstract. *Hyper-heuristics are techniques to select or generate low-level heuristics, generally applied to make the use of search algorithms more generic and to improve their results. Despite their advantages, hyper-heuristics have been underexplored in the Search Based Software Engineering (SBSE) field. An initiative for this purpose that was successfully applied to established unit orders for the integration test, is the Hyper-Heuristic for the Integration and Test Order Problem (HITO). HITO helps in the selection of genetic operators during the algorithm execution. This work presents results from the application of HITO in a more complex version of the same problem, in which unit clusters are considered. This version encompasses modularization restrictions and a greater number of possible operators. Evaluation results obtained with HITO are better or equivalent to that, obtained with the conventional algorithm for all tested systems.*

Resumo. *Hiper-heurísticas são técnicas para a seleção ou geração de heurísticas de baixo nível, geralmente utilizadas para tornar a utilização dos algoritmos de busca mais genérica e melhorar os seus resultados. Apesar das suas vantagens, hiper-heurísticas têm sido pouco exploradas no contexto de Engenharia de Software Baseada em Busca (SBSE). Uma iniciativa com este propósito que obteve bons resultados no estabelecimento de ordens de unidades para realizar o teste de integração, é a Hyper-Heuristic for the Integration and Test Order Problem (HITO). HITO visa à seleção de operadores genéticos em tempo de execução dos algoritmos. Neste trabalho são apresentados resultados da avaliação de HITO em uma versão mais complexa do mesmo problema, na qual o agrupamento de unidades é considerado, o que envolve restrições de modularização e um conjunto maior de possíveis operadores. Os resultados obtidos com HITO são superiores ou equivalentes aos obtidos com o algoritmo genético convencional em todos os sistemas testados.*

1. Introdução

Algoritmos Evolutivos Multiobjetivos (*Multi-Objective Evolutionary Algorithms* – MOEAs) têm sido aplicados em diferentes problemas da engenharia de software [Harman et al. 2012]. Entretanto, tais algoritmos demandam um certo esforço e experiência do engenheiro de software para configurá-los e ajustar seus parâmetros, como

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

por exemplo determinar quais são os melhores operadores de busca (cruzamento, mutação e seleção) e suas probabilidades de aplicação. Além disso, as soluções são, em geral, dependentes do problema e os algoritmos precisam ser ajustados constantemente. O uso de hiper-heurísticas pode ajudar a superar esses desafios [Harman et al. 2012]. Uma hiper-heurística pode ser definida como uma "heurística para selecionar ou gerar heurísticas [de baixo nível]" [Burke et al. 2010]. A principal característica de hiper-heurísticas é que elas agem sobre o espaço de heurísticas para selecionar ou gerar heurísticas que melhor resolvam o problema, ao invés de agir sobre o problema diretamente, o que geralmente acarreta em bons resultados.

Mesmo com as vantagens que podem ser obtidas, o uso de hiper-heurísticas na área de Engenharia de Software Baseada em Busca ainda é um tema pouco explorado. Dentre eles destacam-se: o trabalho de Basgalupp et al. (2013) que apresenta uma hiper-heurística para evoluir algoritmos de geração de árvores de decisão para a predição de esforço. Kumari et al. (2013) apresentam uma hiper-heurística para a solução do problema de agrupamento de módulos. Jia et al. (2015) propõem uma hiper-heurística para aprender e aplicar estratégias de teste combinatorial. Em um trabalho anterior, Guizzo et al. (2015) propuseram uma hiper-heurística para resolver o problema de determinar ordens de unidades (classes ou aspectos) para realizar o teste de integração. Este é um problema difícil da área de teste de software, que consiste em encontrar uma sequência para se testar unidades de software de modo a diminuir o custo para a criação de *stubs* [Wang et al. 2011]. A hiper-heurística chamada HITO (*Hyper-heuristic for the Integration and Test Order Problem*) foi proposta para selecionar operadores de cruzamento e mutação (heurísticas de baixo nível) enquanto MOEAs são executados. Para este fim, HITO utiliza um método de seleção baseado em escore chamado *Choice Function* (CF) [Maashi et al. 2014] e utiliza o MOEA *Non-dominated Sorting Genetic Algorithm-II* (NSGA-II) [Deb et al. 2002]. Nos experimentos realizados, HITO obteve melhores resultados que os obtidos pelo NSGA-II [Guizzo et al. 2015].

Os bons resultados obtidos com HITO motivaram estudos futuros tais como o descrito em [de Carvalho et al. 2015] e o presente trabalho aqui descrito, que tem como objetivo apresentar resultados da aplicação de HITO em uma outra instância do problema, que é o estabelecimento de ordens de teste na presença de restrições de modularização. Neste problema o objetivo também é determinar uma sequência de unidades a fim de minimizar o custo de construção de *stubs*, entretanto considerando uma característica muito importante do software que é a modularidade. Isso implica na existência de restrições que determinam que unidades relacionadas geralmente sejam desenvolvidas em conjunto e devam ser testadas como um agrupamento (*cluster*). Dessa maneira, o problema torna-se mais complexo e outros operadores genéticos que lidam com restrições foram propostos [Assunção et al. 2013], o que aumenta ainda mais a dificuldade de configuração adequada para o testador na hora de aplicar a solução baseada em busca.

Portanto, a hipótese deste trabalho é que a hiper-heurística HITO é capaz de obter melhores resultados que o MOEA, no caso o NSGA-II, em um problema mais restritivo e mais difícil de ser resolvido. Para responder a esta hipótese são apresentados resultados com os mesmos sistemas e metodologia descritos no trabalho que oferece o tratamento convencional para o problema [Assunção et al. 2013].

Este trabalho é organizado da seguinte forma: A Seção 2 contém brevemente

conceitos de hiper-heurísticas. A Seção 3 descreve a hiper-heurística HITO. A avaliação experimental é apresentada na Seção 4, juntamente com as discussões dos resultados. Por fim, a Seção 5 conclui este trabalho e apresenta trabalhos futuros.

2. Hiper-Heurísticas

Uma hiper-heurística pode ser definida como um conjunto de abordagens para selecionar ou gerar heurísticas de baixo nível, com aprendizado durante ou antes da otimização [Burke et al. 2010]. Hiper-heurísticas com aprendizado antes da otimização são chamadas *off-line*, as quais utilizam instâncias de *benchmarks* para serem treinadas *a priori*. Aprendizado durante a otimização é chamado *on-line*, o qual se dá enquanto o problema está sendo resolvido, sem a necessidade de um treinamento prévio [Burke et al. 2010]. Outra possibilidade é não utilizar aprendizado, como por exemplo com uma técnica aleatória ou gulosa. Heurísticas de baixo nível podem ser operadores de busca ou até mesmo meta-heurísticas, sendo classificadas de natureza de perturbação (muda uma solução existente) ou criação (sintetiza uma solução gradualmente).

Hiper-heurísticas de seleção geralmente possuem uma barreira de domínio que esconde detalhes do problema. Assim, a barreira previne que a hiper-heurística se acople com o problema e a mantém genérica. Hiper-heurísticas de seleção geralmente possuem dois componentes principais [Burke et al. 2010]: i) método de seleção – faz a seleção, com ou sem aprendizado; e ii) método de aceitação – decide aceitar ou não uma solução gerada. Neste artigo é utilizado o método de seleção Choice Function (CF) [Maashi et al. 2014]. A Equação 1 apresenta o cálculo de escore de uma heurística de baixo nível utilizando o CF original [Cowling et al. 2001].

$$f(h_i) = \alpha f_1(h_i) + \beta f_2(h_j, h_i) + \delta f_3(h_i) \tag{1}$$

onde h_i é a heurística de baixo nível sendo avaliada; f_1 é a última performance de h_i; f_2 é a última performance de h_i ao ser executada logo após h_j; f_3 é a quantidade de segundos de CPU que se passaram desde a última vez que h_i foi executada; e α, β e δ são os valores de peso para f_1, f_2 e f_3. Uma característica peculiar do CF é que este método faz o balanceamento entre a exploração e a intensificação (*exploration vs exploitation – EvE dilemma*). Enquanto f_1 e f_2 são utilizadas para medir a performance de uma heurística de baixo nível (intensificação), f_3 é utilizada como fator de tempo (exploração). Caso a exploração seja preferível, então o peso δ deve ser incrementado, ou α e β devem ser incrementados caso a intensificação deva ser focada. Isso se dá pois é interessante continuar aplicando heurísticas de baixo nível com boa performance, mas também é interessante aplicar heurísticas de baixo nível que já estão há algum tempo sem serem aplicadas para aumentar a diversidade das soluções.

O método CF utilizado neste trabalho é a versão simplificada apresentada em [Maashi et al. 2014]. Essa adaptação contém apenas as funções f_1 e f_3 do método original. A Equação 2 apresenta esse método.

$$CF(h) = \alpha f_1(h) + \beta f_3(h) \tag{2}$$

Os bons resultados obtidos por esta função em um trabalho anterior [Guizzo et al. 2015] e na literatura [Burke et al. 2010, Maashi et al. 2014] motivaram a sua utilização neste trabalho. Outras vantagens são a sua implementação fácil e a sua compatibilidade com o contexto em que HITO é aplicada (a cada cruzamento).

3. HITO

Hyper-heuristic for the Integration and Test Order Problem (HITO) foi proposta em um trabalho anterior [Guizzo et al. 2015] para a seleção *on-line* de operadores genéticos e para o problema de determinar uma ordem de teste de integração de unidades com mínimo custo de construção de *stubs*, usando o algoritmo NSGA-II.

O problema de ordem de teste de integração de unidades, resumidamente, é um problema de teste de software que tenta diminuir o custo da criação de *stubs* encontrando uma sequência para que as unidades do sistema sejam integradas e testadas [Wang et al. 2011]. Um *stub* é uma simulação de uma unidade requerida e possui um custo para a sua criação. Uma versão mais complexa deste problema insere o agrupamento de unidades, de modo que as unidades de um determinado grupo sejam integradas e testadas juntas [Assunção et al. 2013]. Isso acarreta em outras restrições e em um incremento na dificuldade do problema, uma vez que quando uma solução possui suas restrições violadas, a mesma é corrigida, acaba tendo seu *fitness* deteriorado e o processo de busca se torna menos eficiente. Portanto, é preferível aplicar operadores que consigam otimizar as soluções sem violar suas restrições. Além disso, este problema se torna mais difícil quando múltiplos objetivos são considerados. Em sistemas Orientados a Objetos (OO) uma unidade é uma classe, enquanto que em sistemas Orientado a Aspectos (OA) uma unidade pode ser também um aspecto. Neste trabalho são utilizados sistemas OO e OA, codificados em um problema de permutação e representados por grafos *Object Relation Diagram* (ORD). Além disso, os seguintes objetivos compõem as funções de *fitness* utilizadas para avaliar o custo dos *stubs*: i) A – quantidade de atributos que deverão ser emulados por *stubs*; e ii) M – quantidade de métodos que deverão ser emulados por *stubs*.

Este problema é propício de ser resolvido por hiper-heurísticas como HITO devido às suas características principais: i) é impactado por diferentes fatores e resolvido adequadamente por MOEAs [Assunção et al. 2013]; ii) é difícil de ser resolvido, principalmente considerando o agrupamento de unidades o qual é acompanhado de restrições de modularidade; iii) possui diversos operadores que podem ser adaptativamente selecionados; e iv) o problema é encontrado em diversos contextos, como por exemplo OO e OA [Assunção et al. 2013].

HITO inclui um conjunto de passos apresentados na Figura 1. O primeiro passo é a inicialização de HITO, no qual o usuário seleciona a instância do problema, as funções de *fitness*, o MOEA a ser executado, as heurísticas de baixo nível que sejam compatíveis com a representação e o método de seleção, bem como os parâmetros. No segundo passo são estabelecidas as heurísticas de baixo nível. Neste trabalho cada heurística de baixo nível é composta por um operador de cruzamento e um operador de mutação, ou só um operador de cruzamento, formados por todas as combinações possíveis de quatro operadores de cruzamento e quatro operadores de mutação: i) Inter e Intra Cluster Two Points Crossover, e Inter e Intra Cluster Uniform Crossover; ii) Inter e Intra Cluster Swap Mutation, e Inter e Intra Cluster Simple Insertion Mutation. De acordo com Assunção et al. (2013), a diferença entre os operadores Inter e Intra Cluster é que os operadores Intra Cluster fazem a permutação diretamente nas unidades dentro de um grupo, enquanto que os operadores Inter Cluster fazem a permutação de grupos inteiros. Dados estes operadores, foram formadas 20 heurísticas de baixo nível. A vantagem aqui é que o engenheiro de software não vai precisar selecionar qual destes operadores deverá ser utilizado, sendo

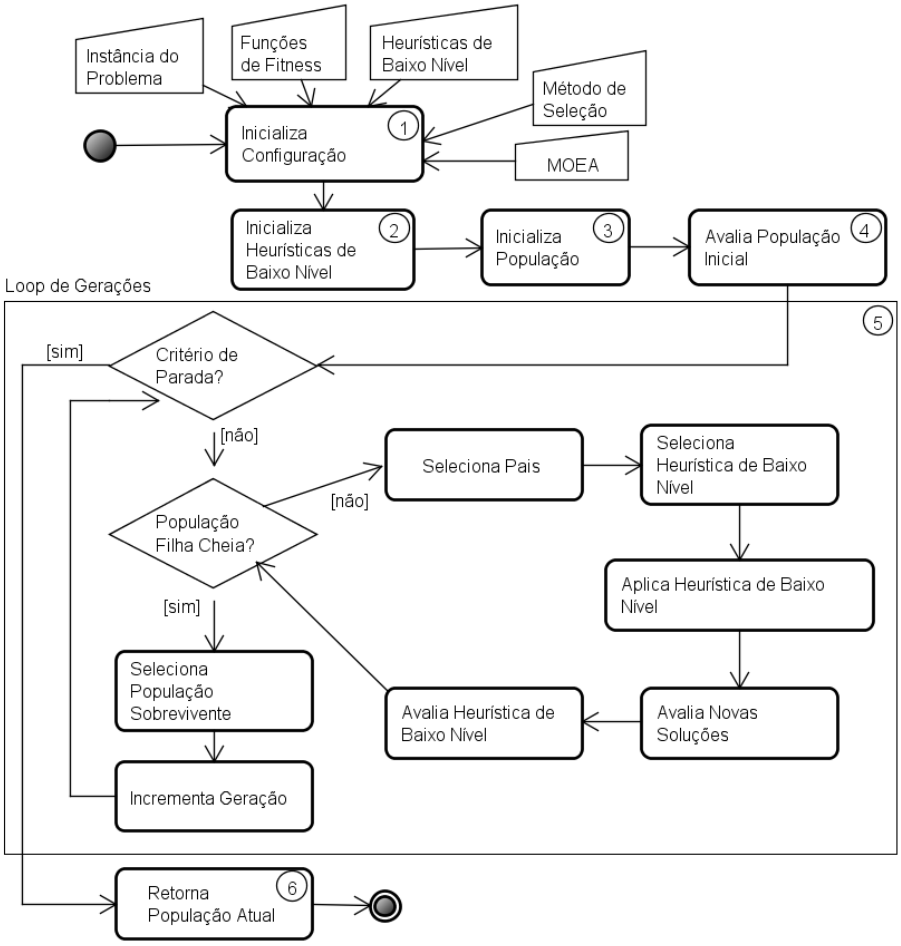


Figura 1. Fluxo de HITO

delegada esta decisão à hiper-heurística. Como mencionado anteriormente, essa seleção é feita por meio do método CF [Maashi et al. 2014].

Cada heurística de baixo nível possui um valor de performance instantânea calculado por f_1 no intervalo $[0, 1]$ e um valor de tempo calculado por f_3 no intervalo $[0..+\infty]$, que definem respectivamente a última performance da heurística de baixo nível e quantos cruzamentos passaram desde a sua última execução. Quanto maior o valor de f_1, melhor a heurística de baixo nível se saiu. No segundo passo de HITO, ambos valores são inicializados com 0 e posteriormente cada heurística é aplicada uma vez para atribuir uma performance inicial. Esses valores são atualizados toda vez que uma heurística de baixo nível é aplicada. Basicamente, a cada atualização, o valor f_3 da heurística aplicada é zerado e o valor f_3 das demais é incrementado em 1. Já para f_1, a sua atualização é feita utilizando apenas os pais e os filhos envolvidos no cruzamento e na mutação, de acordo com a seguinte equação:

$$f_1 = \frac{1}{|P| \cdot |C|} \cdot \sum_{c \in C} \sum_{p \in P} \begin{cases} 1 & \text{if } c \prec p \\ 0 & \text{if } p \prec c \\ 0.5 & \text{caso contrário} \end{cases} \tag{3}$$

onde P são os pais envolvidos no cruzamento; C são os filhos gerados; p é um pai; e c é

um filho. O resultado é um somatório normalizado entre $[0, 1]$ onde 0 significa que ambos os filhos são dominados ($p \prec c$) pelos pais (resultado ruim) e 1 significa que ambos os filhos dominam os pais (resultado bom). Portanto, quanto maior o resultado da equação, melhor a performance da heurística e mais propícia esta é para ser aplicada.

No terceiro passo a população é inicializada, e avaliada no quarto passo. A partir daí o laço de repetição de gerações é iniciado até que o critério de parada seja atingido. A cada geração a população filha é preenchida com novas soluções. Para cada cruzamento, os pais são selecionados pelo algoritmo, a heurística de baixo nível com o maior valor de CF é selecionada e aplicada nos pais, as soluções filhas são avaliadas e a atualização de performance de f_1 é feita no próximo passo, que também atualiza f_3. Após isso, a população sobrevivente é selecionada pelo MOEA e a geração é incrementada. Quando o critério de parada é atingido, o algoritmo finaliza e retorna a população atual.

Hiper-heurísticas que abordam problemas multiobjetivos devem se preocupar em como avaliar a performance de heurísticas de baixo nível, uma vez que soluções podem ser não dominadas. Em [Maashi et al. 2014] os autores utilizam um mecanismo de performance baseado em indicadores de qualidade que avaliam uma população como um todo a cada geração. Outros trabalhos como [Kumari et al. 2013, Kateb et al. 2014] também avaliam a performance a cada geração. Em [Li et al. 2014] os autores avaliam a performance de uma heurística de baixo nível de acordo com os subproblemas de um MOEA baseado em Decomposição (MOEA/D). Entretanto, até onde foi possível analisar, na literatura apenas HITO utiliza uma função de performance que se baseia inteiramente no conceito de dominância de Pareto utilizando apenas os pais e filhos de um cruzamento, e seleciona as heurísticas de baixo nível a cada cruzamento.

Neste artigo apenas o MOEA NSGA-II é utilizado, mas HITO pode trabalhar com outros algoritmos, que geralmente utilizam operadores genéticos para gerar soluções, e dada a grande quantidade de operadores existentes, a seleção destes pode ser delegada à hiper-heurísticas tais como HITO.

4. Avaliação Experimental

Na avaliação experimental apresentada em [Guizzo et al. 2015], HITO superou o algoritmo NSGA-II [Deb et al. 2002] em todos os sistemas considerando o problema de ordem de teste de integração. O presente trabalho tem como objetivo avaliar se HITO consegue obter os mesmos bons resultados diante de um problema mais complexo e restritivo, enquanto seleciona operadores genéticos diferentes. De modo a avaliar isto, foram utilizados os mesmos sistemas e agrupamentos de unidades tal como os descritos no trabalho que propõe a solução convencional para o problema utilizando MOEAs [Assunção et al. 2013]. Os resultados de HITO e do algoritmo NSGA-II são comparados utilizando o indicador de qualidade hypervolume [Zitzler and Thiele 1999] e o teste estatístico Kruskal-Wallis com 95% de significância [Derrac et al. 2011]. Este indicador é um dos mais utilizados na literatura e possui compatibilidade com o operador de comparação "$\triangleleft$" (*better* – determina quando uma fronteira de Pareto é melhor que outra [Zitzler and Thiele 1999]).

Os sistemas utilizados são apresentados na Tabela 1. Esses sistemas são OO ou OA e variam em quantidade de unidades, quantidade de dependências entre unidades, linhas de código (LOC) e número de *clusters*.

Tabela 1. Sistemas utilizados no estudo

Nome	Contexto	Unidades	Dependências	LOC	*Clusters*
MyBatis	OO	331	1271	23535	24
AJHSQLDB	OA	301	1338	68550	15
AJHotDraw	OA	321	1592	18586	12
BCEL	OO	45	289	2999	3
JHotDraw	OO	197	809	20273	13
HealthWatcher	OA	117	399	5479	7
JBoss	OO	150	367	8434	8

Para este trabalho foi adotada a mesma representação, restrições e funções de *fitness* adotadas em [Assunção et al. 2013]. HITO foi implementado usando o framework jMetal [Durillo and Nebro 2011]. Tanto o NSGA-II quanto HITO recebem como entrada o ORD dos sistemas e operam sobre uma representação de permutação, onde o cromossomo é uma lista de inteiros e cada valor (gene) é o número identificador de uma unidade. As unidades devem ser integradas e testadas na ordem em que aparecem nesta lista. Um *stub* é necessário quando uma unidade aparece antes de uma unidade requerida na lista e as funções de *fitness A* e *M* (número de atributos e métodos) calculam os custos associados. Além disso, informações sobre os grupos de unidades são disponibilizadas e utilizadas para manter a integridade das soluções durante o cruzamento e a mutação.

Cada algoritmo (HITO e NSGA-II) foi executado 30 vezes para cada sistema. Por motivos de comparação, o algoritmo NSGA-II e seus parâmetros foram mantidos como apresentados em [Assunção et al. 2013]. Já os parâmetros de HITO foram mantidos como no artigo [Guizzo et al. 2015], no qual foram obtidos após um ajuste empírico. Os parâmetros são: tamanho da população: 300; máximo de gerações (critério de parada): 200; CF α (intensificação): 1; e CF β (exploração): 0.00005.

Em adição, os valores de probabilidade de cruzamento de 95% e mutação de 2% foram mantidos para o NSGA-II, conforme [Assunção et al. 2013]. Já para HITO, probabilidades de cruzamento e mutação não se aplicam, pois a hiper-heurística sempre aplica a heurística de baixo nível selecionada. Assim, caso a mutação deva ser aplicada menos vezes, então HITO identificará essa necessidade e aplicará mais as heurísticas de baixo nível que possuem apenas o operador de cruzamento. Essa é uma das principais vantagens de se utilizar hiper-heurísticas: o usuário não precisa escolher operadores e muito menos configurar suas probabilidades.

Como a fronteira de Pareto real (PF_{true}) para esses sistemas não é conhecida, ela foi formada pelas soluções não dominadas encontradas por ambos os algoritmos. Além disso, cada algoritmo obteve uma fronteira de Pareto conhecida (PF_{known}) formada pelas soluções não dominadas encontradas em suas 30 execuções. Essas fronteiras também foram avaliadas utilizando o hypervolume.

4.1. Análise dos Resultados

A Tabela 2 apresenta os valores de hypervolume de cada PF_{known} encontrados pelos algoritmos. Valores destacados em negrito são os melhores resultados.

Como visto na tabela, HITO foi capaz de encontrar as melhores fronteiras para

Tabela 2. Hypervolume of the PF_{known} fronts for 2 objectives

Sistema	NSGA-II	HITO
MyBatis	3,09E-1	**5,00E-1**
AJHSQLDB	5,27E-2	**7,16E-1**
AJHotDraw	1,42E-2	**5,73E-1**
BCEL	0,00E0	**8,17E-2**
JHotDraw	1,24E-1	**2,78E-1**
HealthWatcher	**2,50E-1**	**2,50E-1**
JBoss	**0,00E0**	**0,00E0**

todos os sistemas, empatando em apenas dois sistemas. Entretanto, esses dados mostram apenas a qualidade das fronteiras de Pareto final encontradas após as 30 execuções. Para uma avaliação mais precisa, a Tabela 3 apresenta a média dos 30 valores de hypervolume encontrados pelos algoritmos, juntamente com os valores de desvio padrão em parênteses. Valores destacados em negrito são os melhores valores ou os que possuem igualdade estatística ao melhor valor de acordo com o teste de Kruskal-Wallis (95% de significância).

Tabela 3. Hypervolume average found for 2 objectives

Sistema	NSGA-II	HITO
MyBatis	2,77E-1 (6,91E-2)	**3,63E-1 (9,13E-2)**
AJHSQLDB	7,74E-2 (8,91E-2)	**2,11E-1 (1,18E-1)**
AJHotDraw	8,08E-2 (5,79E-2)	**3,48E-1 (2,14E-1)**
BCEL	2,84E-2 (1,62E-2)	**6,85E-2 (6,69E-2)**
JHotDraw	**1,11E-1 (9,11E-2)**	**1,26E-1 (1,01E-1)**
HealthWatcher	1,67E-1 (8,80E-3)	**2,03E-1 (6,91E-2)**
JBoss	**4,67E-2 (1,70E-1)**	**3,22E-2 (1,72E-1)**

Em resumo, HITO obteve as melhores médias de hypervolume, ou valores estatisticamente equivalentes ao do NSGA-II para todos os sistemas. Esses resultados são compatíveis com o trabalho anterior [Guizzo et al. 2015]. Além de não precisar escolher os operadores e configurá-los, o usuário pode também obter os melhores resultados utilizando hiper-heurísticas, também na presença de restrições de modularização.

De modo a entender melhor o funcionamento da hiper-heurística, foram avaliadas também a quantidade de vezes que cada heurística de baixo nível foi aplicada. Por motivos de espaço, esses dados não foram apresentados em uma tabela. Assim como ocorreu em [Guizzo et al. 2015], dentre as 20 heurísticas de baixo nível, as mais aplicadas foram sempre as que não possuem mutação, pois apresentaram uma performance melhor. Isso se dá pelo fato de que a mutação pode violar alguma restrição e assim deteriorar as soluções. Com isso, HITO decidiu aplicar menos vezes heurísticas com este operador. Dentre os operadores de cruzamento, para todos os sistemas exceto BCEL, a heurística de baixo nível que possui apenas o Inter Cluster Two Points Crossover foi aplicada mais vezes que as demais, e a que possui apenas o Intra Cluster Two Points Crossover foi aplicada mais vezes que as heurísticas de baixo nível que possuem o Uniform Crossover. Já para o sistema BCEL, o Inter Cluster Uniform Crossover foi melhor. De fato, os operadores

de cruzamento Inter Cluster (permutam grupos de unidades como sendo um gene) foram mais eficientes. Já para a mutação, em todos os sistemas, as heurísticas de baixo nível que possuem os operadores Intra Cluster (permutam unidades dentro de um grupo) foram melhores que as heurísticas de baixo nível que possuem o mesmo cruzamento e seus respectivos operadores de mutação Inter Cluster. Apesar disso, a utilização de ambos os tipos de operadores (Inter e Intra Cluster) são essenciais para a obtenção de bons resultados.

5. Considerações Finais

No artigo em que HITO foi proposta [Guizzo et al. 2015], essa hiper-heurística foi avaliada e obteve os melhores resultados quando comparada com o NSGA-II em todos os sistemas testados. Neste trabalho, HITO foi avaliada para estabelecer ordens de integração e teste de unidades considerando restrições de modularidade, uma versão mais difícil e restritiva. Devido à necessidade de se testar as unidades em conjunto, um número maior de operadores de mutação e cruzamento devem ser aplicados.

Os resultados mostraram que HITO é capaz de superar ou se igualar ao NSGA-II nos 7 sistemas testados, tanto na qualidade das fronteiras de Pareto finais quanto na média de hypervolume em 30 execuções. Além dessa análise, o comportamento de HITO foi estudado considerando a quantidade de vezes que cada uma das 20 heurísticas de baixo nível foi aplicada. A avaliação mostrou que heurísticas de baixo nível que possuem apenas operadores de cruzamento obtém melhores performances. Esses resultados positivos demonstram ainda mais as vantagens de se utilizar hiper-heurísticas como HITO: obtenção de bons resultados e a diminuição do esforço na escolha de quais operadores devem ser utilizados, já que HITO faz isso dinamicamente.

Trabalhos futuros devem explorar a utilização de outros MOEAs em outros problemas do teste de software, tais como a seleção e priorização de casos de teste. Outras heurísticas de baixo nível também podem ser utilizadas para analisar o comportamento de HITO. Pretende-se também utilizar abordagens para o ajuste dinâmico de parâmetros durante a execução da hiper-heurística. Por fim, trabalhos futuros devem explorar outras áreas da engenharia de software que podem se beneficiar do uso de hiper-heurísticas.

Referências

[Assunção et al. 2013] Assunção, W. K. G., Colanzi, T., Vergilio, S., and Pozo, A. (2013). Determining integration and test orders in the presence of modularization restrictions. In *Proceedings of the 27th Brazilian Symposium on Software Engineering (SBES'13)*, pages 31–40.

[Basgalupp et al. 2013] Basgalupp, M. P., Barros, R. C., da Silva, T. S., and Carvalho, A. C. P. L. F. (2013). Software Effort Prediction: A Hyper-heuristic Decision-tree Based Approach. In *Proceedings of the 28th SAC*, pages 1109–1116. ACM.

[Burke et al. 2010] Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., and Woodward, J. R. (2010). A classification of hyper-heuristic approaches. In Gendreau, M. and Potvin, J.-Y., editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 449–468. Springer US.

[Cowling et al. 2001] Cowling, P., Kendall, G., and Soubeiga, E. (2001). A hyperheuristic approach to scheduling a sales summit. In Burke, E. and Erben, W., editors, *Pro-*

ceedings of the 3rd Practice and Theory of Automated Timetabling, volume 2079 of *Lecture Notes in Computer Science*, pages 176–190. Springer Berlin Heidelberg.

[de Carvalho et al. 2015] de Carvalho, V. R., Vergilio, S. R., and Pozo, A. T. R. (2015). Uma hiper-heurística de seleção de meta-heurísticas para estabelecer sequências de módulos para o teste de software. In *VI Workshop de Engenharia de Software Baseada em Busca (WESB'15)*. Submetido.

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

[Derrac et al. 2011] Derrac, J., García, S., Molina, D., and Herrera, F. (2011). A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1):3 – 18.

[Durillo and Nebro 2011] Durillo, J. J. and Nebro, A. J. (2011). jMetal: A Java framework for multi-objective optimization. *Advances in Engineering Software*, 42:760–771.

[Guizzo et al. 2015] Guizzo, G., Fritsche, G. M., Vergilio, S. R., and Pozo, A. T. R. (2015). A hyper-heuristic for the multi-objective integration and test order problem. In *Proceedings of the 24th Genetic and Evolutionary Computation Conference (GECCO'15)*.

[Harman et al. 2012] Harman, M., Burke, E., Clarke, J., and Yao, X. (2012). Dynamic adaptive search based software engineering. In *Proceedings of the 6th ESEM*.

[Jia et al. 2015] Jia, Y., Cohen, M., Harman, M., and Petke, J. (2015). Learning combinatorial interaction test generation strategies using hyperheuristic search. In *Proceedings of the 37th International Conference on Software Engineering (ICSE'15)*.

[Kateb et al. 2014] Kateb, D. E., Fouquet, F., Bourcier, J., and le Traon, Y. (2014). Optimizing multi-objective evolutionary algorithms to enable quality-aware software provisioning. In *Proceedings of the 14th International Conference on Quality Software (QSIC'14)*, pages 85–94.

[Kumari et al. 2013] Kumari, A. C., Srinivas, K., and Gupta, M. P. (2013). Software module clustering using a hyper-heuristic based multi-objective genetic algorithm. In *Proceedings of the 3rd IACC*, pages 813–818.

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

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

[Wang et al. 2011] Wang, Z., Li, B., Wang, L., and Li, Q. (2011). A brief survey on automatic integration test order generation. In *Proceedings of the 23rd SEKE*, pages 254–257.

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

Seleção de Elementos de Elicitação de Requisitos utilizando Algoritmos Evolutivos Multiobjetivo

Renata M. Rêgo, Arilo C. Dias-Neto, Rosiane de Freitas

Instituto de Computação (IComp) – Universidade Federal do Amazonas (UFAM)
Av. General Rodrigo Octávio, 6.200, Campus Universitário Senador Arthur Virgílio Filho – Setor Norte - Manaus - CEP 69.077-000 – Manaus – AM – Brasil

{renata.rego, arilo, rosiane}@icomp.ufam.edu.br

Abstract. *Requirements elicitation is a fundamental activity in a software project, because it aims to identify the software requirements to be developed. This activity usually depends on three elements: the technique used to identify/ collect the requirements, the analyst allocated to accomplish this task and the available suppliers to provide customer needs. These elements should be selected properly during the elicitation process, once they have a direct impact on the software project's success or failure. The issues to be considered during the selection of three elements are their suitability and cost in a software project. This paper presents an approach to support the requirement element selection using SBSE strategies. An empirical study was performed using data from a real software project comparing the results provided by a multi-objective evolutionary algorithm (NSGA-II) and a Random algorithm. The results suggest the feasibility of applying evolutionary algorithms, because it provided the best compromise solutions to this problem.*

Resumo. *A elicitação de requisitos é uma atividade fundamental em um projeto de software, pois ela visa a identificação dos requisitos do software a serem desenvolvidos. Esta atividade geralmente depende de três elementos: a técnica usada para identificar/coletar os requisitos, o analista alocado para realizar essa tarefa e os clientes disponíveis para fornecer suas necessidades. Esses elementos devem ser selecionados de maneira adequada durante o processo de elicitação, uma vez que eles têm impacto direto sobre o sucesso ou fracasso do projeto. Durante a seleção dos elementos são consideradas a adequação e custo destes em relação ao projeto de software. Este artigo apresenta uma abordagem de apoio à seleção de elementos de requisitos utilizando estratégias de SBSE. Foi realizado um estudo empírico utilizando dados de um projeto real comparando os resultados providos por um algoritmo evolutivo multiobjetivo (NSGA-II) e um algoritmo Randômico. Os resultados sugerem a viabilidade de aplicar algoritmos evolutivos, pois o mesmo forneceu as melhores soluções de compromisso para este problema.*

Palavras-chave: *Elicitação de Requisitos, SBSE, Metaheurísticas.*

1. Introdução

A Engenharia de Requisitos é uma fase comum em processos de desenvolvimento de software, podendo ser dividida, de acordo com e [Pressman 2014], em algumas tarefas: Concepção, Elicitação, Negociação, Especificação e Validação. Esta pesquisa concentra-se na tarefa de Elicitação de Requisitos (ER), que consiste em descobrir as necessidades específicas do cliente, extrair as informações sobre o sistema a ser desenvolvido e definir suas funcionalidades.

A ER é realizada de forma recorrente durante o projeto de software e existem vários elementos envolvidos na sua realização, tais como: analistas responsáveis pela coleta de requisitos; os clientes que têm as informações sobre os requisitos; e as técnicas que serão utilizadas para apoiar a elicitação. Cada elemento possui características próprias que podem influenciar a ER [Carrizo *et al.* 2014].

Com base neste cenário, este trabalho descreve o problema de selecionar estes três elementos presentes na ER (técnicas, analistas e clientes) considerando as peculiaridades de cada elemento e as características do projeto de software. Acredita-se que a seleção adequada de tais elementos pode contribuir para a qualidade da informação coletada para um projeto de software e, consequentemente, a qualidade do produto final.

O Problema de Seleção de Elementos de Elicitação de Requisitos (em inglês, *Selection of Requirements Elicitation Elements* – SREE) pode ser caracterizado como um cenário no qual o gerente de um projeto de software possui: (1) um conjunto de técnicas de ER; (2) um conjunto de clientes do software a ser desenvolvido; e (3) um conjunto de analistas disponíveis para realizar a ER. Esse gerente precisa encontrar uma combinação entre técnicas, clientes e analistas que possibilite a coleta do maior número de requisitos (maximizar) sem que isso exceda (ou minimizar) o custo destinado a ER.

Considerando o número crescente de técnicas de ER publicadas na literatura técnica, o conjunto de analistas e clientes disponíveis em um projeto e as restrições envolvidas no contexto do projeto, a seleção de tais elementos em um projeto de software torna-se uma tarefa complexa, pois o conjunto de solução torna-se muito grande.

Nesse contexto, diversos problemas de Engenharia de Software (ES) possuem natureza complexa e que não podem ser tratados por técnicas convencionais, pois envolvem a seleção de uma solução dentro de um grande conjunto [Freitas *et al.,* 2009]. Tais problemas requerem soluções automatizadas que tratem seus diversos aspectos de forma eficiente [Harman *et al.,* 2010]. Algoritmos com estratégias avançadas de busca, especialmente metaheurísticas, podem ser utilizados para resolver esses problemas. A aplicação de metaheurísticas em tais problemas se intensificou nos últimos anos, dando origem a uma nova e promissora área, conhecida como Engenharia de Software Baseada em Busca (do inglês, *Search-Based Software Engineering* – SBSE). Segundo Colanzi *et al.*, (2013), SBSE é o campo de pesquisa e prática que aplica técnicas baseadas em busca para resolver diferentes problemas de otimização em ES. Para Harman *et al.* (2010), as abordagens de SBSE oferecem um conjunto de soluções adaptativas para situações em que o problema envolve múltiplos objetivos concorrentes e conflitantes.

Neste trabalho, o problema SREE foi modelado matematicamente para que tornasse possível a utilização de estratégias de SBSE na sua solução. Assim, foi realizado um estudo empírico utilizando o algoritmo evolutivo multiobjetivo NSGA-II e um algoritmo randômico, no qual foi identificada a viabilidade de utilizar estratégias de busca para solucionar esse problema.

Este artigo está organizado da seguinte forma: a Seção 2 aborda os trabalhos relacionados a este. A Seção 3 apresenta o Problema SREE e a modelagem matemática para o mesmo. A Seção 4 mostra a análise empírica realizada. Por fim, na seção 5 são discutidas considerações finais, incluindo as ameaças à validade e trabalhos futuros.

2. Trabalhos Relacionados

Os trabalhos correlatos identificados na literatura técnica abordam a seleção de elementos da fase de ER (em geral, técnicas) e outros que tratam a seleção de tecnologias em geral, utilizando estratégias de SBSE, cujas contribuições são úteis para essa pesquisa, conforme apresentado na Tabela 1.

Tabela 1. Trabalhos Relacionados e Proposta de Pesquisa.

Trabalho	Contribuição	Estratégia Utilizada	Solução
Kausar, *et. al.* (2010)	Seleção de técnica de elicitação de requisitos	Função matemática	Ranking
Tiwari *et al.* (2012)	Seleção de técnica de elicitação de requisitos	Mapeamento	Lista de Técnicas
Carrizo *et al.* (2014)	Seleção de técnica de elicitação de requisitos considerando perfil de analistas e clientes	Mapeamento	Ranking
Dias-Neto *et al.* (2011)	Seleção de tecnologias de software	Metaheurísticas – SBSE	Gráfico da Frente de Pareto
Grande *et al.* (2014)	Seleção de tecnologias de software	Metaheurísticas – SBSE	Gráfico de radar
Esta Pesquisa	Combinação e Seleção de elementos de elicitação de requisitos	Metaheurísticas – SBSE	Gráfico da Frente de Pareto

Como se pode observar na Tabela 1, os trabalhos de Kausar *et al*. (2010) e Tiwari *et al*. (2012) tratam da seleção de técnicas de ER. Eles utilizam como estratégia de solução função matemática e mapeamento, e as soluções são apresentadas, respectivamente, em forma e *ranking* e lista de técnicas. O trabalho de Carrizo *et al*. (2014) também trata da seleção de técnicas de ER e ainda considera os outros elementos (analistas e clientes) para a seleção. Como limitação, ele não faz a combinação entre esses, mas apenas o mapeamento que torna possível fazer o *ranking* das técnicas. Os trabalhos de Dias-Neto *et al*. (2011) e Grande *et al*. (2014) tratam a seleção de tecnologias de software em geral aplicando estratégias de SBSE (metaheurísticas) e em ambos, as respostas são apresentadas em forma de gráfico, apresentando os itens considerados na seleção.

Esta pesquisa visa recomendar ao engenheiro de software soluções que contenham a combinação dos elementos de elicitação adequados ao projeto utilizando estratégias de SBSE (metaheurísticas) para encontrar o conjunto soluções. Elas serão apresentadas no Gráfico da Frente de Pareto de forma a identificar a adequação e o custo de cada solução ao projeto, facilitando assim a tomada de decisão.

Na próxima seção será apresentado o problema SREE, a formulação matemática e suas funções objetivo.

3. Definição do Problema SREE

O processo de ER precisa ser realizado de forma adequada para evitar que os requisitos passem para a fase de desenvolvimento incompletos ou inconsistentes, pois isso os tornam mais caros. Um mecanismo para apoiar este processo é sistematizar o problema SREE, pois esta seleção muitas vezes é realizada de forma subjetiva, trazendo resultados negativos à tarefa de elicitação e impactam negativamente a qualidade do produto final.

Em [Carrizo *et al*., 2014] e [Tiwari *et al*., 2012], os autores identificaram 15 técnicas de elicitação de requisitos por meio de revisão sistemática da literatura, bem como os elementos de elicitação e os atributos relacionados a eles (Tabela 2).

Tabela 2. Fatores de influência e seus atributos.

Ele.	ID	Atributo	Possível Classificação
Analista	A01	Treinamento em técnicas de elicitação	Alto, Médio, Baixo
	A02	Experiência com elicitação	Alto, Médio, Baixo
	A03	Experiência com técnicas de elicitação	Alto, Médio, Baixo
	A04	Familiaridade com o domínio	Alto, Médio, Baixo
	A05	Habilidade na condução de conversas	Alto, Médio, Baixo
	A06	Habilidades na resolução de conflitos	Alto, Médio, Baixo
	A07	Habilidades com liderança e motivação	Alto, Médio, Baixo
	A08	Habilidades com trabalho em equipe	Alto, Médio, Baixo
	A09	Habilidade em registrar eventos	Alto, Médio, Baixo
	A10	Habilidade com captura de ideias	Alto, Médio, Baixo
	A11	Habilidade analítica	Alto, Médio, Baixo
Cliente (informante)	I01	Pessoas por sessão	Individual, Grupo (2-5), Muitos (+5)
	I02	Consenso entre os clientes	Alto, Baixo
	I02	Interesse do cliente	Alto, Médio, Baixo
	I04	Nível de Experiência	Especialista, Conhecedor, Iniciante
	I05	Capacidade de Articulação	Alto, Médio, Baixo
	I06	Disponibilidade de tempo	Alto, Baixo
	I07	Localização/ Acessibilidade	Longe, Perto
	I08	Habilidade de comunicação	Alto, Médio, Baixo
	I09	Habilidade de trabalho em equipe	Alto, Médio, Baixo
	I10	Habilidade em expressar ideias	Alto, Médio, Baixo
Projeto	P01	Tipo de informação a ser coletada	Estratégica, Tática, Básica
	P02	Disponibilidade da informação	Muito, Pouco, Zero
	P03	Definição do problema	Alto, Baixo
	P04	Restrição de Tempo do Projeto	Alto, Baixo, Zero
	P05	Tempo de processo	Início, Meio, Fim

Assim, cada técnica pode ser caracterizada de acordo com os níveis, que representam a adequação da técnica em um contexto específico e relacionada com os outros elementos para que seja possível fazer as combinações entre os elementos.

3.1. Caracterização das Técnicas e Adequação dos elementos

Para modelar a solução proposta, é necessária a realização da adequação das técnicas com relação aos outros elementos. Os possíveis níveis de adequação foram definidos por Carrizo *et al*., (2014), e os valores associados a eles seguiram a regra de penalidade de soluções onde:

- **Recomendado (valor 10):** A técnica é recomendada quando o seu uso no contexto analisado proporciona bons resultados.
- **Indiferente (valor 0):** A técnica é indiferente quando não há garantia que seu uso fornece bons (ou maus) resultados no contexto em que está sendo analisada.
- **Inadequada (valor -5):** A técnica é considerada inadequada quando o seu uso no contexto analisado pode fornecer resultados negativos.

Como exemplo, ao caracterizarmos a técnica de **Entrevista** para o atributo **Pessoas por sessão** do elemento **Cliente**, ela obteve valor 10 para a classificação **Individual** (ou seja, é recomendada neste contexto), 0 para a classificação **Grupo** (ou seja, é indiferente neste contexto) e -5 para a classificação **Muitos** (ou seja, é inadequada neste contexto). Esta caracterização foi realizada para todas as 15 técnicas de ER considerando todos os 16 atributos apresentados na Tabela 2 para todas as classificações que compõem os atributos.

3.2. Definições dos Objetivos do Problema SREE

Dois objetivos são considerados no problema SREE: encontrar um subconjunto de técnicas, analistas e clientes que maximizam o nível de adequação em relação às características do projeto e que ao mesmo tempo minimizem o esforço necessário para que os analistas e clientes utilizem o subconjunto de técnicas escolhidas.

Como exemplo, considerando 15 técnicas, 5 clientes e 5 analistas disponíveis para um projeto, podendo selecionar até 3 técnicas, 3 clientes e 3 analistas, o número de possíveis soluções seria (15 x 14 x 13) x (5 x 4 x 3) x (5 x 4 x 3) = 9.828.000 soluções diferentes. Por isso, técnicas de otimização podem ser usadas para encontrar soluções ótimas ou aproximadas em uma quantidade razoável de tempo, mas é necessário que o problema esteja formulado adequadamente. A seguir, serão apresentados os termos que representam os conjuntos a serem considerados neste problema.

- Seja $T = \{t_1, t_2, \dots, t_n\}$, o conjunto de técnicas de ER candidatas à seleção. Cada técnica $t_i \in T$ tem um custo associado ct_i, que representa o esforço e o tempo necessários para que esta seja aplicada a um projeto de software. Então, seja $CostTec = \{ct_1, ct_2, \dots, ct_n\}$ o conjunto dos custos das técnicas.
- Seja $A = \{an_1, an_2, \dots, an_q\}$, o conjunto de analistas disponíveis para a seleção. Assim, cada analista j terá um nível de adequação adA_{ij}, que mede a sua adequação para aplicar a técnica i. Além disso, cada analista $an_j \in A$ tem um custo associado ca_j, que representa o custo para alocar o analista ao projeto de acordo com o sua habilidade. Então, seja $CostAn = \{ca_1, ca_2, \dots, ca_q\}$ o conjunto dos custos de analistas.
- Seja $I = \{in_1, in_2, \dots, in_r\}$ o conjunto de clientes disponíveis para a seleção. Então, cada clientes k terá um nível de adequação adI_{ik}, que mede a sua adequação em relação ao aplicar a técnica i. Além disso, cada cliente $in_k \in I$ tem um custo associado ci_k, que representa o custo para alocar o cliente no projeto de acordo com sua posição na empresa. Então, seja $CostIn = \{ci_1, ci_2, \dots, ci_r\}$ o conjunto de custo informantes.

Várias técnicas podem ser adequadas a um projeto de software. Assim, a adequação que a técnica t_i tem para o projeto de software é dada pela soma dos seus atributos. Quanto maior for este valor, maior é a adequação da técnica para o projeto em questão. As adequações dos analistas e clientes são calculadas da mesma forma.

3.3 Modelagem Multiobjetivo do Problema SREE

De modo a definir as funções objetivo para o problema SREE, foi definida primeiramente a estrutura dos cromossomos que representam uma solução para este problema. O cromossomo é composto por três partes, representando as técnicas, analistas e clientes. Os valores dos genes são binários, definidos como segue:

$$gene\begin{cases}1, \text{se o elemento está presente na solução}\\0, \text{outro caso}\end{cases}$$

No exemplo apresentado na Figura 1, a solução é formada pelas técnicas #1 e #3, os analistas #1 e #2, e os clientes #2 e #4, pois eles têm o valor 1 em seus genes.

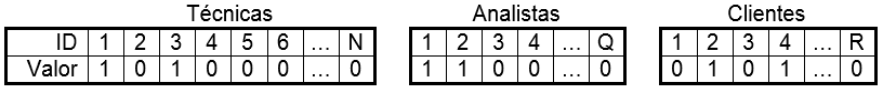


Figura 1: Problema da Seleção de Elementos de Elicitação de Requisitos.

Depois de definir estrutura do cromossomo, foram formalizadas as duas funções objetivo, que estão descritas nas próximas subseções.

3.3.1 Função Objetivo para Adequação

A função objetivo para adequação utiliza os seguintes parâmetros:

- adT_t: nível de adequação da técnica t ao projeto.
- $adA_{t,a}$: nível de adequação do analista a para aplicar a técnica t.
- $adC_{t,c}$: nível de adequação do cliente c para aplicar técnica t.
- N: número de técnicas de elicitação de requisitos a serem avaliadas.
- A: número de analistas a serem avaliadas.
- C: número cliente a serem avaliados.

Função Objetivo

$$maximize\ f1 = \sum_{t=1}^{N} adT_t\, s_t * \sum_{t=1}^{N}\sum_{a=1}^{A} adA_{t,a}\, anT_{t,a} * \sum_{t=1}^{N}\sum_{p=1}^{C} adC_{t,c}\, clT_{t,c}$$

Variáveis de Decisão

$$s_t = \begin{cases} 1, & \text{se a técnica } t \text{ faz parte da solução} \\ 0, & \text{caso contrário} \end{cases}$$

$$anT_{t,a} = \begin{cases} 1, & \text{se a técnica } t \text{ e o analista } a \text{ fazem parte da solução} \\ 0, & \text{caso contrário} \end{cases}$$

$$clT_{t,c} = \begin{cases} 1, & \text{se a técnica } t \text{ e o cliente } c \text{ fazem parte da solução} \\ 0, & \text{caso contrário} \end{cases}$$

Para maximizar adequação dos elementos para um projeto de software são utilizadas a adequação da técnica (adT_t) presente na solução, ou seja, $t_t = 1$ somada com a adequação dos outros elementos: analistas ($adA_{t,a}$) e clientes ($adC_{t,c}$) também presentes, por exemplo, quando o analista $anT_{t,a} = 1$ e o cliente $clT_{t,c} = 1$, mas estes a adequação é calculada em relação a técnica selecionada.

3.3.2 Função Objetivo para Custo

Todos os elementos presentes no problema SREE tem um custo associado à sua aplicação em um projeto de software. Estes custos são descritos abaixo:

Técnica: o custo de uma técnica de elicitação está relacionado ao tempo requerido para coletar os requisitos e o esforço dispendido para aplicá-la. Foi definida uma escala ordinal de 1 a 5 para representar estes valores, onde o 1 é o mínimo e 5 é o máximo tanto para tempo quanto para o esforço.

Analista: o custo de um analista no problema SREE está relacionado à sua posição/especialização/salário em uma organização de software. Neste trabalho, este item foi normalizado em uma escala ordinal de 1 a 5, onde 1 é o menor custo e 5, o mais elevado.

Cliente: o custo de um cliente no problema SREE também está relacionado à sua posição/especialização/salário em uma organização de software. Por exemplo, o custo para alocar um diretor da empresa seria superior ao de alocar um membro do pessoal técnico. A escala utilizada é a mesma do analista.

A função objetivo para adequação utiliza os seguintes parâmetros:

- cT_t: custo de aplicar a técnica t no projeto.
- cA_a: custo de alocar o analista a ao projeto.
- cC_c: custo de alocar o cliente c ao projeto.
- N: número de técnicas de elicitação de requisitos a serem avaliadas.
- A: número de analistas a serem avaliadas.
- C: número clientes a serem avaliados.

Função Objetivo

$$minimize\ f2 = \sum_{t=1}^{N} cT_t\, s_t + \sum_{a=1}^{A} cA_a\, an_a + \sum_{p=1}^{C} cC_c\, cl_c$$

Variáveis de Decisão

$$s_t = \begin{cases} 1, & \text{se a técnica } t \text{ faz parte da solução} \\ 0, & \text{caso contrário} \end{cases}$$

$$an_a = \begin{cases} 1, & \text{se o analista } a \text{ faz parte da solução} \\ 0, & \text{caso contrário} \end{cases}$$

$$cl_c = \begin{cases} 1, & \text{se o cliente } c \text{ faz parte da solução} \\ 0, & \text{caso contrário} \end{cases}$$

Para minimizar o custo de elementos para um projeto de software foram somados o custo para aplicar as técnicas selecionadas para um projeto de software (cT_t), o custo de analistas (cA_a) e clientes (cC_c), também presentes na solução.

Na próxima seção, serão apresentados os resultados de um experimento que avaliou o problema SREE instanciado utilizando um algoritmo evolutivo NSGAII. O algoritmo foi modelado utilizando a formulação do problema descrito nesta seção.

4. Análise Empírica com o Problema SREE

4.1. Dados e Algoritmos Utilizados no Estudo

Neste estudo foram utilizadas 15 técnicas de elicitação de requisitos com base no trabalho de Carrizo *et al.*, (2014): 1. Entrevista Aberta, 2. Entrevista Estruturada, 3. Observação de Tarefa, 4. *Card Sorting*, 5. Questionários, 6. Análise de Protocolo, 7. *Repertory Grid*, 8. *Brainstoming*, 9. Grupo nominal, 10. Técnica *Delphi*, 11. Observação de participantes, 12. Prototipagem, 13. Grupo Focal, 14. Oficina JAD e 15. Cenários/Casos de uso.

Assim, a fim de avaliar o modelo proposto foi utilizado um projeto real desenvolvido por uma organização de software localizada em Manaus/Amazonas. O projeto consiste em um sistema web para apoiar o controle de tarefas administrativas, operacionais e financeiras de uma empresa de negócios.

Os dados relacionados ao projeto (adT), analistas ($adA_1, adA_2\ e\ adA_3$) e clientes (adC_1, adC_2, adC_3) são apresentados na Tabela 3 onde é apresentada a adequação destes em relação às técnicas de elicitação ($t_1, \ldots, t_{15}$). As linhas da tabela representam a adequação do elemento à técnica de acordo com a situação contextual e os custos de cada uma das técnicas e de cada um dos elementos, por exemplo, na célula $adT\ x\ t_1$ = 125, significa que a adequação da técnica 1 (Entrevista aberta) ao projeto é

de 125, na célula $adT\ x\ t_7 = 25$, significa que a adequação da técnica 7 (*Repertory Grid*) ao projeto é de 25, na célula $adA_2\ x\ t_6 = 10$, significa que a adequação do analista 2 à técnica 6 (Análise de Protocolo) ao projeto é de 10, e assim sucessivamente.

Tabela 3. Dados utilizados no estudo.

Adequação das Técnicas ao Projeto															
ID	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}
adT	125	100	125	50	100	90	25	100	100	100	125	100	125	90	100
Adequação dos Analistas às Técnicas															
ID	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}
adA_1	140	140	110	110	140	140	140	140	110	110	140	110	140	140	110
adA_2	60	60	70	15	100	10	15	60	110	55	100	30	30	-25	30
adA_3	140	140	110	110	140	100	110	100	110	110	140	110	110	100	110
Adequação dos clientes às Técnicas															
ID	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}
adC_1	115	115	90	115	115	95	115	75	100	75	90	115	90	75	115
adC_2	115	115	70	115	115	95	115	75	100	75	90	115	90	75	115
adC_3	75	75	90	115	115	65	115	45	70	55	70	115	60	45	95
Custo das Técnicas															
ID	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}
cT	4	6	6	8	12	12	15	5	4	15	6	15	8	10	15

Custo dos Analistas			
ID	an_1	an_2	an_3
cA	5	3	4

Custo dos Cliente			
ID	cl_1	cl_2	cl_3
cC	4	5	3

Neste estudo foi utilizado o algoritmo evolutivo multiobjetivo NSGA-II (*Non-dominated Sorting Genetic Algorithm II*). Ele é um algoritmo que implementa o conceito de dominância e elitismo, sendo realizada em duas etapas, utilizando mecanismos de: *Non-dominated Sorting* – buscando soluções próximas à Frente de Pareto ótima – e *Crowding Distance Sorting* - a realização da busca de soluções distribuídas no espaço [Deb et. al. 2000]. Todos os indivíduos são classificados em uma frente de Pareto-ótima, e a elite da primeira população é utilizada como ponto de partida para a próxima geração.

De acordo com Harman e Jones (2011), qualquer metaheurística deve ser capaz de superar um algoritmo de busca puramente randômica em melhores soluções. Então, foi utilizado um algoritmo Randômico para comparação com os resultados do NSGA-II. Na busca randômica, a solução é gerada aleatoriamente, esta pode encontrar boas soluções, mas não se tem garantia, pois não existe processo de melhoria na solução.

4.2. Avaliação e Análise dos Resultados

Os experimentos foram executados em um computador com a seguinte configuração: Hardware – Processador Intel Core i5, 1.80 GHz e memória RAM de 8GB; e Software – Sistema Operacional Windows 8 (64 bits). Os parâmetros utilizados no algoritmo NSGA-II foram: (a) População Inicial de 50, 100, 150 e 200; (b) Número Máximo de Avaliações de 25000; (c) Taxa de Cruzamento de 0,9; (d) Tipo de Cruzamento SBX Crossover; (e) Taxa de Mutação 1/# variáveis de decisão; (f) Tipo de Mutação *Polynomial Mutation*; (g) Tipo de Seleção *Binary Tournament* e para o algoritmo Randômico foram são utilizadas a população inicial de 50, 100, 150 e 200.

Neste trabalho, foi calculada a média e o desvio padrão de cinquenta execuções de cada algoritmo em cada caso para definir o comportamento do tempo de execução (em milissegundos). Os resultados obtidos em cada algoritmo são apresentados na Tabela 4. Com relação ao tempo de execução, o algoritmo randômico apresentou melhor desempenho (menor média e menor desvio-padrão) em todas as instâncias, justificável pelo fato de não utilizar nenhum critério custoso na geração das soluções.

Tabela 4. Tempo de execução em milissegundos e Desvio Padrão.

Instância	NSGA-II	Randômico
50	37132 ± 1310.407	197 ± 18.288
100	67129 ± 3553.506	268 ± 37.653
150	87023 ± 5846.762	406 ± 74.117
200	130256 ± 4449.420	587 ± 116.242

A eficácia de ambos os algoritmos foi analisada pelas soluções geradas em função dos dois objetivos: custo x adequação. Os resultados podem ser observados na Figura 2, onde os valores foram gerados por ambos os algoritmos para todos os casos (50, 100, 150 e 200 indivíduos como população inicial). Em todos os casos, a qualidade das soluções obtidas pelo algoritmo randômico foi superada pelo NSGA-II, pois as soluções do algoritmo randômico encontram-se dispersas e longe da frente de Pareto ótima.

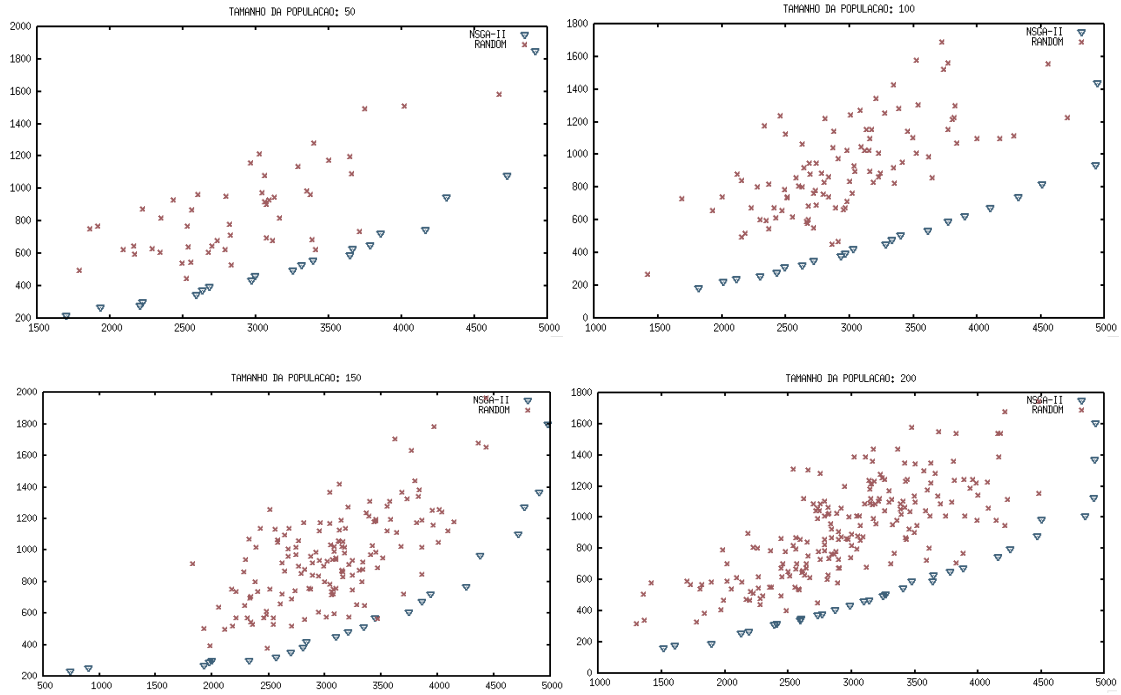


Figura 2: Solução do problema SREE com os Algoritmos NSGA-II e Randômico.

5. Conclusões

Este artigo discutiu o problema SREE, que visa, simultaneamente, maximizar a adequação dos elementos de levantamento (analista, cliente e técnica) e minimizar o custo para aplicação das técnicas selecionadas para ER de software. Um estudo empírico comparando um algoritmo evolutivo multiobjetivo (NSGA-II) com um algoritmo randômico foi executado e os resultados sugerem a aplicação de algoritmos evolutivos no problema SREE, pois o NSGA-II gerou as melhores soluções em todas as instâncias.

Algumas ameaças à validade deste experimento podem ser citadas, como por exemplo: a utilização de escalas intervalares nos valores das técnicas no que diz respeito aos valores atribuídos aos atributos com relação as técnicas – recomendado, indiferente, inadequado. Esses valores estão sendo analisados e calibrados durante os experimentos afim de aproximar ao máximo do mundo real. Outra ameaça seriam os parâmetros utilizados no algoritmo NSGA-II, pois foram utilizados os valores que estavam configurados na ferramenta JMetal. Neste caso, podem ser testados mais combinações de taxas de mutação e cruzamento, bem como outros métodos de cruzamento e mutação. Por fim, como o cromossomo é dividido em 3 partes, as operações de cruzamento e mutação ocorreram apenas na parte relacionada a técnicas e podem, ainda, ser ampliado para as outras duas partes, o que poderá, inclusive, melhorar a qualidade das soluções geradas.

Como trabalhos futuros estão previstos a ampliação das operações de cruzamento e mutação nas soluções geradas e a realização de novos experimentos com outros parâmetros no NSGA-II e ainda com outras metaheurísticas (por exemplo: SPEA2 e MOCell) a fim de comparar os resultados e identificar a metaheurística mais adequada para o problema SREE. Além disso, pretende-se desenvolver uma interface para apoiar os gerentes de projeto de software para executar experimentos com o objetivo de encontrar boas soluções (técnicas, analistas e clientes mais adequados de acordo com as características do projeto de software) para projetos de software reais.

Referências

Carrizo, D.; Dieste, O.; Juristo, N. (2014) “Systematizing requirements elicitation technique selection”, In: Information and Software Technology. Vol, 56, pp. 644-669.

Deb, K., Pratap, A., Agarwal, S. e Meyarivan, T. (2000), “A fast and elitist multiobjective genetic algorithm: NSGA-II”, In: IEEE Transactions on Evolutionary Computation, pp. 182–197.

Dias-Neto, A. C., Rodrigues, R. F., Travassos, G. H. (2011) “Porantim-opt: optimizing the combined selection of model-based testing techniques”, In: 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops, pp. 174-183.

Freitas, F. G. de; Maia, C. L. B.; Coutinho, D. P. G.; Campos, A. L. de, e Souza, J. T. de. (2009) “Aplicação de Metaheurísticas em Problemas da Engenharia de Software: Revisão de literatura”, II Congresso Tecnológico Infobrasil.

Grande, A. S.; Rodrigues, R. F.; Dias-Neto, A.C. (2014) “A Framework to Support the Selection of Software Technologies by Search-Based Strategy”, In: IEEE 26th International Conference on Tools with Artificial Intelligence (ICTAI), pp. 979-983.

Harman, M. e Jones, B. F. (2001) “Search-Based Software Engineering”, In: Information & Software Technology, 43(14), pp. 833-839.

Harman, M. e Mansouri, A. (2010) "Search Based Software Engineering”, In: IEEE Transactions on Software Engineering, vol.36, no.6, pp.737, 741

Kausar, S.; Tariq, S.; Riaz, S.; Khaum, A. (2010) “Guidelines for the selection of elicitation techniques”, In: 6th International Conference on Emerging Technologies (ICET), pp. 265-269.

Pressman, R. (2014). Software Engineering: A Practitioner's Approach (8th ed.). McGraw-Hill Higher Education.

Uma abordagem utilizando NSGA-II In Vitro para resolução do Problema do Próximo Release Multiobjetivo

Átila Freitas[1], Allysson Allex Araújo[1], Matheus Paixão[2], Altino Dantas[1], Celso Camilo-Júnior[3], Jerffeson Souza[1]

[1]Universidade Estadual do Ceará (UECE)
Avenida Dr. Silas Munguba, 1700 – Fortaleza, Ceará. Brasil
Grupo de Otimização em Engenharia de Software da UECE
http://goes.uece.br

[2]University College London (UCL)
Malet Place, WC1E 6B – London. United Kingdom
CREST Centre

[3]Universidade Federal de Goiás (UFG)
Alameda Palmeiras, s/n – Goiânia, Goiás. Brasil
Instituto de Informática

atila.freitas@aluno.uece.br, matheus.paixao.14@ucl.ac.uk, celso@inf.ufg.br

{allysson.araujo, altino.dantas, jerffeson.souza}@uece.br

Abstract. *The selection of requirements to be implemented in the next release is considered a complex activity, characterized by the presence of several conflicting aspects that have some influence in the decision making. This problem is usually called the Next Release Problem. Several multiobjective evolutionary algorithms were proposed to solve it, especially the NSGA-II. However, this algorithm presents shortcomings regarding convergence and search space exploration. Thus, this paper proposes an approach to mitigate this flaw through the incorporation of an auxiliar algorithm, known as In Vitro, to the NSGA-II. Empirical studies are able to statistically demonstrate that the proposed algorithm presents better solutions when compared to the standard versions of both NSGA-II and MoCell.*

Resumo. *A seleção de requisitos para serem implementados no próximo release é considerada uma atividade complexa, caracterizada pela presença de diversos aspectos conflitantes que podem influenciar na tomada de decisão. Neste contexto, surge o Problema do Próximo Release, para o qual vários algoritmos evolucionários multiobjetivos foram propostos para sua resolução, com destaque para o NSGA-II. Todavia, verifica-se uma recorrente dificuldade com relação a capacidade de convergência e exploração do espaço de busca. Desta forma, este artigo propõe mitigar essa dificuldade através da incorporação de um algoritmo auxiliar, conhecido como In Vitro, ao NSGA-II. Experimentos realizados são aptos a demonstrar estatisticamente que o algoritmo proposto apresenta soluções melhores que as versões tradicionais do NSGA-II e MoCell.*

1. Introdução

Um dos modelos de desenvolvimento de software mais empregados atualmente é o modelo iterativo e incremental. Nessa metodologia, o desenvolvimento é realizado em ciclos, nos quais a versão executável disponibilizada ao final de cada ciclo é chamada de *release* [Sommerville 2011]. Apesar das suas vantagens, esse modelo possui algumas dificuldades como a seleção de quais requisitos deverão ser implementados no próximo *release*.

É notória a presença de diversos fatores que podem influenciar na decisão de quais requisitos devem ser incluídos no próximo *release*. Por exemplo, cada requisito pode possuir uma certa importância para o(s) cliente(s), porém em um projeto constituído de muitos clientes é natural que haja divergências com relação aos requisitos. Além disso, ainda existem restrições importantes a serem consideradas, como o custo atribuído à implementação de cada funcionalidade. Dessa forma, surge o Problema do Próximo *Release*, ou *Next Release Problem* (NRP) [Bagnall et al. 2001], o qual consiste em selecionar um subconjunto de requisitos a serem implementados no próximo *release*, considerando tanto a importância dos requisitos, quanto os respectivos custos.

O NRP foi primeiramente modelado como um problema de otimização mono-objetivo [Bagnall et al. 2001], onde a meta é maximizar a importância dos clientes satisfeitos com o *release*. Seguindo os princípios da *Search Based Software Engineering* [Harman et al. 2012], tal abordagem pode ser considerada como uma ferramenta de tomada de decisão, pois a técnica de busca retorna uma única solução, ou seja, sugere apenas um subconjunto de requisitos a serem desenvolvidos.

Motivada pela necessidade de representar melhor a realidade, uma versão multi-objetivo desse problema foi modelada e nomeada *Multi-Objective Next Release Problem* (MONRP) [Zhang et al. 2007]. Devido ao seu caráter multiobjetivo, o problema apresenta um conjunto de soluções igualmente boas, ao invés de apenas uma única solução. Tal conjunto de soluções é chamado de Frente de Pareto e o engenheiro de software tem a responsabilidade de definir qual solução (*release*) será adotada.

Dentre as várias técnicas para resolução do MONRP, pode-se destacar os algoritmos evolucionários multiobjetivos, em especial o NSGA-II [Deb 2014]. Em [Zhang et al. 2007], a modelagem é baseada no conflito entre maximização da importância dos requisitos e a minimização do custo total do *release*. Para avaliar a proposta, foram utilizados o NSGA-II, um AG mono-objetivo e um algoritmo de busca aleatória. Conforme esperado, o NSGA-II encontrou as melhores Frentes de Pareto. Porém, identificou-se que o mesmo não foi capaz de explorar devidamente o espaço de busca, constatado através da dificuldade em buscar soluções encontradas pelo AG.

Em [Durillo et al. 2009] é conduzido um estudo aprofundado sobre o MONRP, comparando os resultados do NSGA-II com outra técnica multiobjetivo denominada MoCell e, visando o teste de sanidade, com um algoritmo de busca aleatória. Em termos de experimentos, verificou-se através das métricas *Hypervolume* e *Spread* definidas no mesmo trabalho, que os resultados entre os dois algoritmos multiobjetivos são similares, mas novamente constatou-se a dificuldade de ambos algoritmos com relação a exploração do espaço de busca. Em [Silva et al. 2011] é desenvolvida uma abordagem híbrida utilizando o NSGA-II e técnicas de otimização exata. A partir do estudo empírico realizado, provou-se que a abordagem atingiu resultados superiores ao NSGA-II tradicional.

Em virtude da recorrente dificuldade em proporcionar aos algoritmos evolucionários uma maior capacidade de convergência e exploração do espaço de busca, considera-se válida e proeminente a proposta discutida em [Camilo-Junior and Yamanaka 2011] na qual é apresentada uma estratégia que contribui diretamente para lidar com tais quesitos. Fundamentada no conceito de fertilização *In Vitro*, a mesma funciona como um algoritmo auxiliar que realiza operações na população em prol de uma manipulação mais refinada do material genético. Devido a sua característica auxiliar, essa abordagem pode ser executada em paralelo ou incorporada a qualquer algoritmo genético.

Portanto, este trabalho propõe incorporar a técnica *In Vitro Fertilization* (IVF) ao algoritmo NSGA-II para resolução do Problema do Próximo *Release* Multiobjetivo. Ao se acoplar o *In Vitro*, espera-se proporcionar ao NSGA-II uma exploração mais rápida e efetiva do espaço de busca, ou seja, encontrar rapidamente boas soluções para o MONRP.

O restante do trabalho é organizado da seguinte forma: na Seção 2 descreve-se a formulação matemática referente ao Problema do Próximo *Release* Multiobjetivo; na Seção 3 elucidam-se os conceitos do algoritmo *In Vitro* e o processo de adaptação ao NSGA-II; na Seção 4 discute-se os resultados atingidos a partir dos experimentos realizados. Finalmente, na Seção 5 apresentam-se as conclusões e trabalhos futuros.

2. Formulação Matemática do Problema

Considere o conjunto $R = \{r_1, r_2, \ldots, r_N\}$ de requisitos a serem selecionados, onde N é o número total de requisitos. Considere também o conjunto dos clientes representado por $C = \{c_1, c_2, \ldots, c_M\}$, onde M é a quantidade total de clientes. Cada cliente indica um certo valor de importância específica para cada requisito, no formato $V_j = \{v_1^j, v_2^j, \ldots, v_N^j\}$, onde $1 \leq j \leq M$ e v_i^j representa a nota atribuída pelo cliente c_j para o requisito r_i. Cada cliente ainda possui um valor que retrata sua importância perante a empresa, representados por $W = \{w_1, w_2, \ldots, w_M\}$, onde w_j indica a importância do cliente c_j. A importância total de um certo requisito r_i é dada pelo valor de $score_i$, o qual é calculado como uma soma dos valores de importância atribuídos por cada cliente, ponderada pela respectiva importância do cliente da seguinte forma:

$$score_i = \sum_{j=1}^{M} v_i^j \times w_j \tag{1}$$

Além da importância, cada requisito r_i possui um certo custo de desenvolvimento, representado por $COST = \{cost_1, cost_2, \ldots, cost_N\}$, onde $cost_i$ representa o custo referente ao requisito r_i. Dessa forma, a formulação do MONRP consiste em:

$$min - \sum_{i=1}^{N} score_i \times x_i \tag{2}$$

$$min \sum_{i=1}^{N} cost_i \times x_i \tag{3}$$

sendo as variáveis de decisão representadas pelo vetor $X = \{x_1, x_2, \ldots, x_N\}$, onde $x_i = 1$ implica que o requisito r_i está incluído, e $x_i = 0$ caso contrário. Visando uma maior

adequação na visualização dos resultados, o objetivo de maximizar o $score_i$ foi convertido para uma minimização, através da multiplicação do seu respectivo valor por -1.

Considere também que o requisito r_i pode requerer o requisito r_j, gerando uma relação de interdependência. Com isso, se uma solução possui r_i, ela deve conter também r_j para ser considerada uma solução válida no espaço de busca.

3. Proposta de NSGA-II In Vitro

Uma dificuldade recorrente em algoritmos evolucionários é a perda de informações importantes na transição de uma população para outra. Conforme mencionado anteriormente, em [Camilo-Junior and Yamanaka 2011] é proposto um algoritmo auxiliar, conhecido como *In Vitro*, para lidar com esse tipo de situação em um Algoritmo Genético.

O presente artigo propõe a adaptação da técnica *In Vitro* ao NSGA-II. Em relação a este processo, o primeiro passo foi decidir que a cada iteração será escolhido um objetivo a se otimizar de acordo com uma taxa previamente estipulada. Dessa forma, torna-se possível "ordenar" os indivíduos e determinar se uma solução é melhor do que outra.

A partir dessa concepção, torna-se plausível ponderar as preferências referentes a convergência de um determinado objetivo. Por exemplo, caso utilize-se uma taxa de 80% para o $score$, existe 80% de chance do $score$ ser o objetivo a ser otimizado, gerando soluções que terão maiores valores em termos dele. Naturalmente, caso se estabeleça uma taxa de 50%, há uma chance igualitária de escolha entre os dois objetivos.

No Algoritmo 1 apresenta-se o funcionamento do NSGA-II utilizando a técnica *In Vitro* (IVF/NSGAII). Conforme pode-se visualizar, a diferença com relação à versão tradicional do NSGA-II reside em acoplar e adaptar os conceitos de **Coleta**, **Manipulação Genética** e **Transferência** específicos do *In Vitro*, os quais serão discutidos a seguir:

Algoritmo 1: *IVF/NSGAII*

```
   Entrada: pesoObjetivo1, pesoObjetivo2
 1 início
 2     Pop ← popInicial();
 3     Frentes ← nonDominatedSort();
 4     crowdingDistance(Frentes);
 5     enquanto criterioDeParada() faça
 6         Filhos ← geraFilhos(Pop);
 7         objetivo ← escolheObjetivo();
 8         indivSel ← coleta();
 9         pai ← melhor(indivSel, objetivo);
10         superFilhos ← manipulaGenetica(pai, indivSel, objetivo);
11         Filhos ← transfere(superFilhos, Filhos);
12         union ← pai ∪ Filhos;
13         Frentes ← nonDominatedSort(union);
14         Pop ← geraNovaPop(Frentes);
15     fim
16     crowdingDistance(frentes[i]);
17     completaMembrosColetados(frentes[i]);
18     retorna frenteDePareto
19 fim
```

A fase da **Coleta** tem por função selecionar uma parte dos indivíduos que farão

parte do processo do *In Vitro* de acordo com alguma estratégia de seleção previamente definida. Neste caso, N indivíduos são selecionados a partir das primeiras frentes, porém, caso o tamanho da frente supere o valor separado para coleta, utiliza-se o critério de *Crowding Distance* presente em [Deb 2014], para definir quais indivíduos serão coletados. Desses indivíduos coletados, o melhor deles, quanto ao objetivo previamente definido, será o pai e o restante serão as mães. Assim, os filhos gerados deverão ser melhores que o pai escolhido.

Antes de se prosseguir com o fluxo do algoritmo, é necessário definir a estratégia de divisão do material genético a ser empregada, haja vista que a próxima fase será referente ao processo de **Manipulação Genética**. No presente artigo, optou-se por cortar o indivíduo em um ponto aleatório. Diante dessa decisão, ocorre a execução de duas sub-etapas:

1. A *alteração genética* é responsável por modificar ou não características de uma parcela das mães coletadas. Consequentemente, espera-se melhorar a busca global e o resgate de informações que podem ter sido perdidas ao longo do algoritmo. Para este momento existem dois grupos de operadores. O primeiro grupo, composto apenas pelo operador denominado *Assisted Recombination* (AR), consiste simplesmente em não alterar os indivíduos. Já o segundo grupo, chamado de *Exploratory Assisted Recombination*, é constituído por quatro operadores (EAR-P, EAR-T, EAR-N e EAR-PA) que têm a responsabilidade de alterar os genes de algumas mães antes de serem recombinadas. Em síntese, o operador EAR-N gera novas mães através do processo de criação de soluções escolhido. O EAR-T altera todos os genes das soluções selecionadas. O EAR-P e o EAR-PA são responsáveis por alterar uma parte das mães, sendo modificados todos os genes a partir do corte (EAR-P) ou alguns genes definidos aleatoriamente (EAR-PA). Tais operadores são exemplificados na Figura 1.

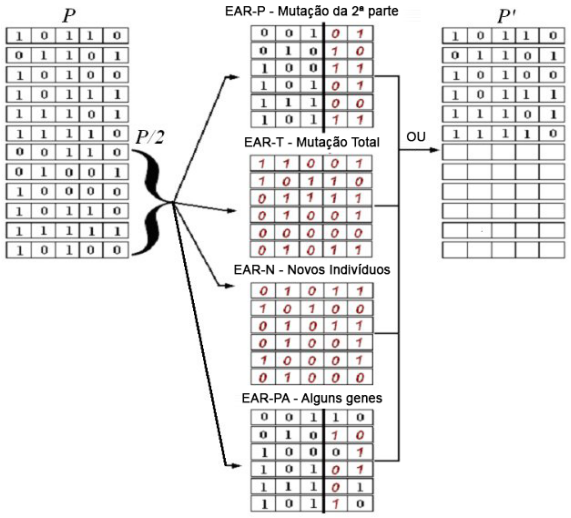


Figura 1. Operadores de Alteração Genética - Fonte: Adaptado de [Camilo-Junior and Yamanaka 2011].

2. Depois de alteradas, as mães são recombinadas ao pai dependendo do ponto de corte do material genético, denominada *recombinação genética*. É relevante men-

cionar que na versão mono-objetivo, apresentada na Figura 2, o pai é substituído pelo melhor filho recursivamente. Já para a presente adaptação multiobjetivo, optou-se por possibilitar que todos os filhos melhores que o pai (de acordo com o objetivo selecionado) sejam capazes de gerar um conjunto de super-filhos.

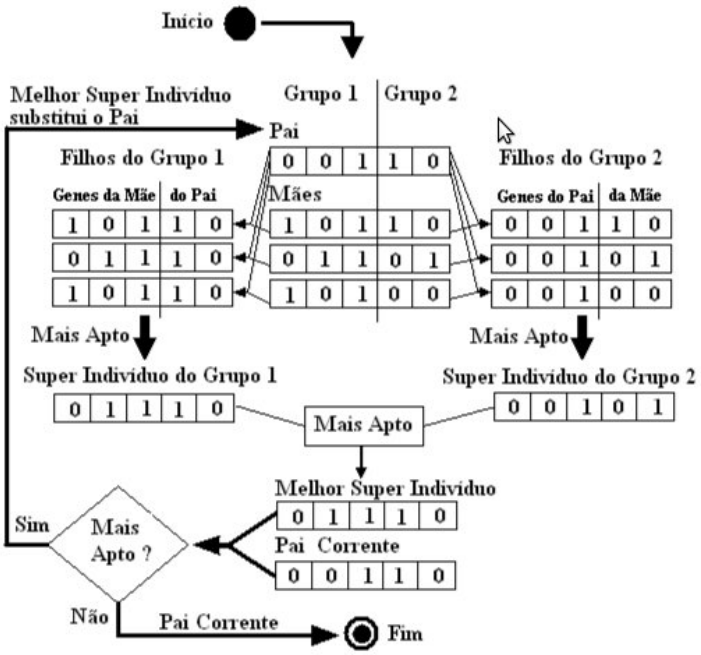


Figura 2. Recombinação Genética - Fonte: [Camilo-Junior and Yamanaka 2011].

Por fim, após o processo de manipulação genética ter sido devidamente realizado e caso tenham sido gerados indivíduos melhor adaptados que o melhor indivíduo da população corrente, ocorre a última fase denominada **Transferência**. Fundamentalmente, esse processo é responsável por introduzir os super-filhos na população. Neste trabalho utilizou-se a introdução direta de todos os indivíduos na população corrente, restando ao processo particular de ordenação do NSGA-II selecionar as melhores soluções.

4. Avaliação Empírica

O conjunto de instâncias utilizado neste trabalho também foi o mesmo empregado em [Silva et al. 2011]. São 15 instâncias geradas aleatoriamente variando de 20 a 500 requisitos e 2 a 20 clientes. Os nomes das instâncias estão no formato I_R_C, onde R representa o número de requisitos e C o número de clientes. Os valores de importância do requisito, custo do requisito e importância do cliente podem variar entre 1 e 5.

A presente proposta foi implementada utilizando o *framework* jMetal [Durillo and Nebro 2011]. Objetivando uma análise mais abrangente, cada instância foi solucionada utilizando um operador distinto de *alteração genética* do *In Vitro*, totalizando cinco variações do IVF/NSGAII: IVF-AR/NSGAII, IVF-EAR_N/NSGAII, IVF-EAR_T/NSGAII, IVF-EAR_P/NSGAII e IVF-EAR_PA/NSGAII. Para fins de comparação, coletou-se os resultados do NSGA-II tradicional, de um outro algoritmo evolucionário multi-objetivo denominado MoCell e do Branch-and-Bound, o qual auxi-

liou na geração das frentes reais. Todos os algoritmos excluem soluções que quebram as interdependências.

Todos os algoritmos (com exceção da técnica exata) foram executados com os seguintes parâmetros: taxa de cruzamento de 90%, taxa de mutação $\frac{1}{N}$, onde N é a quantidade de requisitos da instância. Cada população possui um total de 100 indivíduos. O valor referente ao *feedback* do MoCell é de 5 indivíduos. Como critério de parada estabeleceu-se um número máximo de 100.000 avaliações.

Para lidar com o caráter estocástico e se certificar da coerência dos resultados, para cada instância, cada algoritmo foi executado 100 vezes coletando a média e desvio padrão dos valores de *Hypervolume (HV)* e *Spread*, métricas abordadas em [Durillo et al. 2009], além do tempo de execução em milissegundos. Mensurou-se a diferença estatística utilizando o teste U de Mann-Whitney e o *effect size* através do Vargha-Delaney $\hat{A}_{12}$, conforme é sugerido em [Arcuri and Briand 2014].

4.1. Resultados e Análises

Por restrição de espaço, optou-se por apresentar na Tabela 1 apenas os dados coletados a partir das instâncias que possuem 20 clientes, com quantidade de requisitos variando entre 20, 50, 100, 250 e 500. Todavia, todos resultados utilizando as demais instâncias estão disponíveis on-line[1]. Conforme pode-se observar, ao se empregar algum dos operadores *In Vitro*, o tempo tende a ser maior que a versão tradicional, por exemplo, para a I_20_20, o tempo do IVF-AR/NSGAII é 5,6% maior que do NSGA-II tradicional. Porém, em alguns casos, os tempos do NSGA-II tradicional são superiores às versões IVF-EAR_T/NSGAII, IVF-EAR_P/NSGAII e IVF-EAR_PA/NSGAII, como na instância I_500_20 para a qual o NSGA-II tradicional obtém um tempo 3,8% maior do que o IVF-EAR_N/NSGAII. Isto pode ser explicado pelo fato de que o IVF/NSGAII converge mais rápido que o NSGA-II. Com isso, mesmo tendo mais operações, estas não consomem tanto tempo quanto o processo de ordenação pelo *Crowding Distance* que é chamado de acordo com o número de frentes que precisam ser geradas até gerar a população que irá para próxima geração.

Tabela 1. Valores de *Hypervolume*, *Spread* e Tempo referentes ao MoCell, NSGA-II e variações do IVF/NSGAII. Melhores resultados destacados em negrito.

Instância	Métrica	MoCell	NSGA-II	IVF-AR/NSGAII	IVF-EAR_N/NSGAII	IVF-EAR_T/NSGAII	IVF-EAR_P/NSGAII	IVF-EAR_PA/NSGAII
I_20_20	HV	0.594±0.242	0.594±0.242	0.594±0.242	0.594±0.242	0.580±0.237	0.582±0.238	0.582±0.237
	SPREAD	**0.264 ± 0.108**	0.895±0.365	0.895±0.365	0.895±0.365	1.042±0.427	1.026±0.420	1.025±0.420
	TEMPO	**829.87±339.47**	1424.91±582.51	1504.18±616.42	1448.82±591.69	1443.76±589.90	1460.28±597.12	1463.86±599.74
I_50_20	HV	**0.643±0.262**	0.640±0.261	0.640±0.261	0.640±0.261	0.625±0.255	0.623±0.254	0.625±0.255
	SPREAD	**0.216±0.088**	0.469±0.194	0.474±0.196	0.469±0.194	0.639±0.277	0.652±0.288	0.629±0.273
	TEMPO	**1277.1±524.15**	1727.24±705.71	1798.61±736.12	1741.13±712.08	1707.29±697.20	1726.4±705.62	1731.25±707.43
I_100_20	HV	**0.627±0.256**	0.622±0.254	0.625±0.255	0.625±0.255	0.573±0.235	0.590±0.241	0.592±0.242
	SPREAD	**0.185±0.076**	0.447±0.185	0.370±0.153	0.375±0.156	0.723±0.336	0.629±0.282	0.617±0.273
	TEMPO	**2196.11±897.60**	2536.64±1036.26	2517.12±1029.73	2413.43±986.394	2396.85±978.807	2425.49±990.328	2425.13±990.246
I_250_20	HV	0.595±0.243	0.586±0.239	**0.615±0.251**	0.612±0.249	0.583±0.241	0.593±0.242	0.597±0.244
	SPREAD	**0.407±0.168**	0.610±0.250	0.446±0.185	0.472±0.195	0.672±0.333	0.627±0.287	0.620±0.280
	TEMPO	5741.19±2346.18	5673.62±2317.31	5708.91±2331.91	5436.59±2220.70	**5320.25±2173.22**	5416.84±2213.29	5426.95±2216.80
I_500_20	HV	0.528±0.215	0.525±0.214	**0.567±0.231**	0.557±0.227	0.549±0.224	0.558±0.228	0.558±0.228
	SPREAD	0.599±0.245	0.735±0.300	**0.588±0.241**	0.618±0.253	0.752±0.326	0.679±0.287	0.666±0.279
	TEMPO	15316.52±6258.88	14542.81±5939.76	14851.04±6067.04	14009.91±5722.43	**13757.98±5620.62**	13907.95±5680.31	13979.18±5711.19

Observando as métricas *Hypervolume* e *Spread*, pode-se verificar que para instâncias com menor número de requisitos o comportamento é similar entre os algoritmos NSGA-II tradicional, o IVF-AR/NSGAII e o IVF-EAR_N/NSGAII. Em específico, na instância I_50_20, o MoCell se mostra superior ao IVF-EAR_N/NSGAII em 0.4% em

[1] `http://goes.uece.br/atilafreitas/ivf-monrp`

Hypervolume e 53,9% em *Spread*. Já na instância I_100_20, o MoCell é melhor 0,3% em *Hypervolume* e 50% em *Spread*.

Em contrapartida, conforme o aumento da quantidade de requisitos, o que também acresce mais complexidade, a proposta apresenta resultados melhores em ambas as métricas para todas variações de operadores, ou seja, IVF-AR/NSGAII, IVF-EAR_N/NSGAII, IVF-EAR_P/NSGAII e IVF-EAR_PA/NSGAII. Analisando a instância I_500_20, pode-se constatar que o IVF-AR/NSGAII supera o NSGA-II tradicional em *Hypervolume* e *Spread*, em respectivamente 20% e 8%. O mesmo ainda supera o MoCell em *Hypervolume* e *Spread* em 1.8% e 7.38%.

Na Tabela 2 apresenta-se para cada instância os *p-values* e os valores de *effect size* relativos às comparações entre a configuração do IVF/NSGAII que atingiu melhores resultados, ou seja IVF-AR/NSGAII, com o NSGA-II tradicional e o MoCell. Conforme pode-se constatar, em sua maioria os *p-values* são consideravelmente baixos, comprovando estatisticamente que as amostras podem ser analisadas como significantemente diferentes. Com exceção da I_20_20, que possui valores de *Hypervolume* e *Spread* totalmente iguais, é nítida a diferença de desempenho entre os algoritmos, mesmo que pequenas. Visando uma maior liberdade de análise, apresentam-se todos os *p-values* obtidos para que o nível de significância seja avaliado de acordo com os critérios do leitor.

Tabela 2. Resultados de *p-values* e *effect size* obtidos através da comparação entre as amostras do IVF-AR/NSGAII, NSGA-II tradicional e MoCell.

Instâncias	Métricas	IVF-AR/NSGAII & NSGA-II		IVF-AR/NSGAII & MoCell	
		p-value	effect size	p-value	effect size
I_r20_c20	HV	1	0.5 (negligible)	1	0.5 (negligible)
	Spread	1	0.5 (negligible)	2.2e-16	1 (large)
	Tempo	4.60e-10	0.72 (medium)	2.2e-16	1 (large)
I_50_20	HV	0.03	0.40 (small)	2.2e-16	0 (large)
	Spread	0.29	0.52 (small)	2.2e-16	1 (large)
	Tempo	1.10e-08	0.76 (large)	2.2e-16	0.99 (large)
I_100_20	HV	0	0.91 (large)	2.2e-16	0.04 (large)
	Spread	2.2e-16	0.07 (large)	2.2e-16	1 (large)
	Tempo	2.e-3	0.39 (small)	2.2e-16	0.99 (large)
I_250_20	HV	2.2e-16	1 (large)	2.2e-16	1 (large)
	Spread	2.2e-16	0 (large)	4.11e-11	0.80 (large)
	Tempo	0.03	0.59 (small)	1.79e-15	0.10 (large)
I_500_20	HV	2.2e-16	1 (large)	2.2e-16	1 (large)
	Spread	2.2e-16	0 (large)	2.12e-05	0.34 (small)
	Tempo	4.95e-09	0.74 (large)	2.2e-16	0.03 (large)

Através dos valores de *effect size* obtidos, pode-se ratificar as discussões delineadas a partir dos resultados descritos na Tabela 1. É válido salientar que em termos de *Hypervolume*, valores superiores a 0.5 são interpretados como positivos, pois indicam que na maioria das vezes o IVF-AR/NSGAII atinge valores *maiores* que o algoritmo comparado. Já para *Spread* e tempo, considera-se que quanto mais próximo de 0 melhor, afinal, isto implica que na maioria das vezes o IVF-AR/NSGAII obteve valores *menores*.

Para as instâncias constituídas de até 100 requisitos, o MoCell apresenta vantagem em todas métricas avaliadas, mas com o IVF-AR/NSGAII superando sua versão tradicional em algumas ocasiões como, por exemplo, em *Hypervolume* para a I_100_20 o qual atinge um *effect size* de 0.91. Para I_250_20, o IVF-AR/NSGAII supera sua versão tradicional em termos de *Hypervolume* e *Spread* com valores de *effect size* de 1 e 0, respecti-

vamente. Já quando se compara ao MoCell, a superioridade é apenas em *Hypervolume*, sendo inferior em *Spread* cujo *effect size* tem um valor de 0.8. No entanto, ao se analisar I_500_20, a técnica *In Vitro* se sobressai claramente em ambas as métricas com relação aos os outros dois algoritmos, validando assim, as conclusões previamente apresentadas.

Com relação ao tempo de execução, verificou-se que o desempenho do IVF-AR/NSGAII também está intimamente relacionado à complexidade da instância. Dessa maneira, à medida que se aumenta a quantidade de requisitos, a presente abordagem tende a atingir resultados inferiores que o NSGA-II tradicional, porém melhores que o MoCell.

A Figura 3, apresenta as frentes obtidas pelos algoritmos Branch-and-Bound (considerada a frente Real), o MoCell, o IVF-AR/NSGAII e o NSGA-II tradicional para a I_500_20, na qual o eixo X representa a importância e o eixo Y o custo. Os demais gráficos encontram-se disponíveis na página de suporte deste artigo. Como pode-se verificar, ao se empregar a técnica *In Vitro* obtém-se uma melhor cobertura da frente real do que os outros algoritmos e ainda consumindo menos tempo que o MoCell.

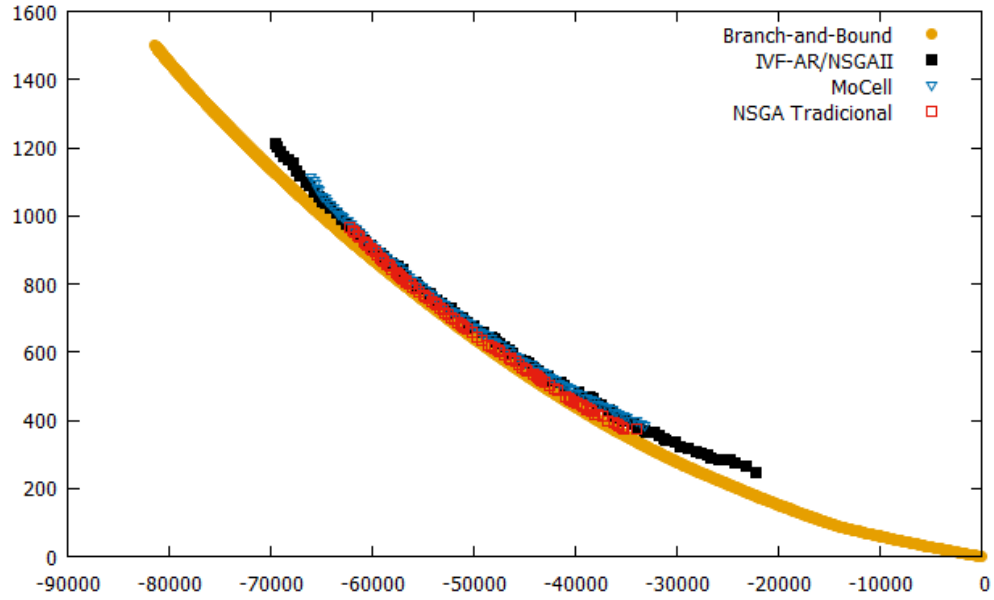


Figura 3. Resultados dos algoritmos IVF-AR/NSGAII, NSGA-II tradicional, MoCell e Branch-and-Bound para I_500_20

Portanto, como constatado nas análises das Tabelas 1 e 2, em específico, o IVF-AR/NSGAII satisfaz às expectativas, sendo relativamente tão bom quanto o NSGA-II tradicional e o MoCell para as instâncias menores e superior para as instâncias maiores. A maior cobertura da frente real, verificada na Figura 3, comprova uma melhor exploração do espaço de busca e geração de uma maior diversidade de soluções.

5. Considerações Finais

Selecionar quais requisitos estarão inclusos no próximo *release* do sistema é uma tarefa complexa que demanda esforço. Logo, a utilização de técnicas de busca multiobjetivos para sua resolução demonstra-se bastante proeminente por proporcionar uma melhor tomada de decisão na presença de critérios conflitantes. Todavia, conforme constatado na

literatura, há uma recorrente dificuldade em tais algoritmos com relação a capacidade de convergência e exploração do espaço de busca.

A técnica *In Vitro* funciona como um algoritmo auxiliar cujo objetivo é proporcionar um maior aproveitamento do material genético dos indivíduos e uma melhor exploração do espaço de busca. Dito isso, este trabalho propôs acoplar esta técnica ao NSGA-II, reconhecidamente um dos algoritmos evolucionários multiobjetivos mais utilizados em *Search Based Software Engineering*. Através da avaliação empírica, demonstrou-se que a abordagem gerou resultados satisfatórios, superando a versão tradicional do NSGA-II e o MoCell. Dessa forma, pode-se concluir que, quanto mais complexa é a instância submetida ao IVF/NSGAII, melhor é a cobertura e exploração do espaço de busca em comparação aos algoritmos comumente usados na literatura.

Em termos de trabalhos futuros pretende-se verificar o comportamento da abordagem ao se empregar instâncias reais, desenvolver uma hiper-heurística que gerencie os operadores do *In Vitro*, aplicar a proposta em outros problemas e elaborar uma comparação com outras abordagens híbridas discutidas na literatura.

Referências

Arcuri, A. and Briand, L. (2014). A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. *Software Testing, Verification and Reliability*, 24(3):219–250.

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

Camilo-Junior, C. G. and Yamanaka, K. (2011). In vitro fertilization genetic algorithm. *Evolutionary Algorithms, Prof. Eisuke Kita (Ed.). InTech*.

Deb, K. (2014). Multi-objective optimization. In *Search methodologies*, pages 403–449. Springer.

Durillo, J. J. and Nebro, A. J. (2011). jmetal: A java framework for multi-objective optimization. *Advances in Engineering Software*, 42(10):760–771.

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

Harman, M., McMinn, P., De Souza, J. T., and Yoo, S. (2012). Search based software engineering: Techniques, taxonomy, tutorial. In *Empirical Software Engineering and Verification*, pages 1–59. Springer.

Silva, T. G. N., Freitas, F., and Souza, J. (2011). Abordagem híbrida para o problema multiobjetivo do próximo release. In *II Workshop em Engenharia de Software Baseada em Busca*.

Sommerville, I. (2011). *Software Engineering*. Addison Wesley.

Zhang, Y., Harman, M., and Mansouri, S. A. (2007). The multi-objective next release problem. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1129–1137. ACM.

Uma Adaptação dos Operadores Genéticos para o Problema do Próximo Release com Interdependência entre Requisitos

Italo Yeltsin, Allysson Allex Araújo, Altino Dantas, Jerffeson Souza

[1]Universidade Estadual do Ceará (UECE)
Avenida Dr. Silas Munguba, 1700. Fortaleza, Ceará. Brasil
Grupo de Otimização em Engenharia de Software da UECE
http://goes.uece.br

italo.medeiros@aluno.uece.br,

{allysson.araujo, altino.dantas, jerffeson.souza}@uece.br

Abstract. *The Next Release Problem consists in selecting a set of requirements to be developed in the next release, aiming to maximize the overall importance value. When considering the requirements' interdependencies, more constraints are added and, consequently, require a properly suiting of search algorithms to reach better results. This paper presents a comparation between a set of adapted operators of the Genetic Algorithm aiming to deal with such characteristics. Empirical study showed this approach outperformed the canonical version in 37% in fitness.*

Resumo. *O Problema do Próximo Release consiste em selecionar um conjunto de requisitos para serem implementados no próximo release, de forma que sua importância seja maximizada. Ao se considerar as interdependências entre requisitos, mais restrições são adicionadas e, consequentemente, demanda-se uma adaptação mais refinada para que os algoritmos de busca atinjam resultados melhores. Este trabalho apresenta uma comparação entre um conjunto de operadores adaptados de um Algoritmo Genético visando lidar melhor com tais características. O estudo empírico realizado demonstrou que a abordagem proposta consegue ser até 37% melhor em fitness que sua versão canônica.*

1. Introdução

O termo *release* é utilizado para designar uma nova versão funcional do software que deverá ser entregue ao cliente. Normalmente, durante o planejamento de um *release*, a equipe de desenvolvimento deve estabelecer o melhor *trade-off* sobre quais requisitos devem ser implementados considerando restrições de projeto como, por exemplo, o orçamento disponível. Nomeado Problema do Próximo *Release*, em inglês *Next Release Problem* (NRP), esse problema é amplamente discutido em *Search Based Software Engineering* [Harman et al. 2012] e foi primeiramente formulado em [Bagnall et al. 2001].

Em termos gerais, o trabalho [Bagnall et al. 2001] consiste em selecionar um subconjunto de clientes cuja satisfação é mensurada através da presença dos requisitos definidos como importantes pelos mesmos, de forma que o custo de implementação do *release* não ultrapasse o orçamento planejado. Naturalmente, surgiram outras modelagens deste problema sob o viés de concepções distintas. Em [Baker et al. 2006], objetiva-se selecionar um subconjunto de requisitos considerados importantes, respeitando o orçamento

definido. Posteriormente, em [Zhang et al. 2007] apresenta-se uma formulação multi-objetiva que visa maximizar a importância e minimizar o custo do *release*.

Considerando o fato de que a engenharia de requisitos é pautada pela presença de características complexas e muitas vezes conflitantes, verifica-se a importância em tornar a modelagem do problema próxima da realidade. Assim, um aspecto importante a ser incorporado na seleção de requisitos é o conceito de interdependências entre requisitos, conforme discutido em [Carlshamre et al. 2001] e [Dahlstedt and Persson 2005].

De acordo com Del Sagrado, Aguila e Orellana (2011), interações entre requisitos, ou seja, requisitos que seguem uma ordem de precedência, requisitos independentes com relação aos outros ou requisitos que devem ser incluídos ao mesmo tempo são situações comuns no desenvolvimento de software. Neste trabalho foi analisado o quão impactante é a presença de interdependências no processo de busca.

Outro trabalho que explora o NRP considerando interdependências é o [Nascimento and Souza 2012], cuja proposta é fundamentada na aplicação de uma adaptação do algoritmo de Otimização por Colônia de Formigas. Os resultados são comparados com outras duas metaheurísticas não adaptadas, que no caso foram a Têmpera Simulada e Algoritmo Genético (AG).

Uma proposta inicial do presente trabalho foi apresentada em [Yeltsin et al. 2013]. Todavia, o presente trabalho estende significativamente o trabalho anterior promovendo: (i) melhorias nos algoritmos de geração de soluções e reparo previamente apresentados; (ii) novas adaptações para os operadores de cruzamento e mutação.

Portanto, este trabalho tem como objetivo principal propor uma adaptação dos operadores genéticos de um AG para o Problema do Próximo *Release*, considerando interdependências entre requisitos. Experimentos realizados demonstram que os operadores propostos possibilitam atingir resultados melhores que sua versão canônica.

O restante deste artigo está organizado da seguinte forma: na Seção 2 são discutidos os conceitos de interdependências entre requisitos; na Seção 3 apresenta-se a formulação matemática do NRP; na Seção 4 detalham-se as adaptações propostas para os operadores genéticos, enquanto na Seção 5 discutem-se os resultados alcançados. Finalmente, na Seção 6 apresentam-se as considerações finais e os trabalhos futuros.

2. Interdependências entre Requisitos

Os requisitos relacionam-se entre si de várias formas diferentes e complexas, sendo esses relacionamentos chamados de interdependências. Em um ambiente de planejamento do próximo *release*, essas interdependências ganham mais importância, pois o fato dos requisitos se relacionarem torna difícil e, eventualmente impossível, selecionar um conjunto de requisitos baseado somente em prioridades do cliente [Carlshamre et al. 2001].

Conforme explanado em [Del Sagrado et al. 2011], as interdependências podem ser distinguidas em dois grupos:

- *Interdependências funcionais*: determinam relações nas quais a implementação de certo requisito depende diretamente da implementação de outro. Tais interdependências adicionam mais restrições ao problema, restringindo o espaço de busca e dificultando sua exploração. Podem ser classificadas como:

  - R_1 **AND** R_2, quando o requisito R_1 depende do requisito R_2 para funcionar e vice-versa. Isto é, ambos devem ser implementados no mesmo *release*.
  - R_1 **REQUIRES** R_2, neste caso o requisito R_1 depende do requisito R_2, mas não vice-versa. O requisito R_1 só pode ser incluído no *release* se R_2 também o for, mas R_2 pode ser selecionado sozinho.

  Nesse contexto, é válido destacar a diferença entre dependências e requisitos dependentes. Dependências de um requisito r_i refere-se ao conjunto de todos requisitos que r_i necessita para ser selecionado para o próximo *release*, incluindo as respectivas dependências das dependências. Já os requisitos dependentes de r_i são todos aqueles que, para serem selecionados, requerem r_i no *release*.

- *Interdependências de valor*: a implementação de um requisito pode influenciar nas características de outro requisito, como aumentar a importância ou diminuir seu custo. Sua influência pode tanto ser positiva quanto negativa. Neste caso, é natural que haja uma maior dificuldade no cálculo da importância e do custo dos requisitos, devido a alta dinamicidade nos valores. Seus tipos são:
  - R_1 **CVALUE** R_2, a implementação de R_1 influencia no valor de importância do requisito R_2.
  - R_1 **ICOST** R_2, a implementação do requisito R_1 influencia no custo de desenvolvimento de R_2.

3. Formulação Matemática do Problema

Considere $R = \{r_1, r_2, \ldots, r_N\}$ o conjunto de requisitos disponíveis para seleção, sendo N o número total destes. Os valores w_i e c_i indicam, respectivamente, a importância e o custo de implementação do requisito r_i. O orçamento determinado para o *release* é dado por b. Dessa forma, a formulação do NRP adotada neste trabalho consiste em:

$$\text{maximizar: } \sum_{i=1}^{N} w_i x_i \tag{1}$$

$$\text{sujeito a: } \sum_{i=1}^{N} c_i x_i \leq b \tag{2}$$

onde, a variável de decisão $X = \{x_1, x_2, \ldots, x_N\}$ indica os requisitos selecionados para o próximo *release*, ou seja, r_i é incluído no *release* quando $x_i = 1$, caso contrário $x_i = 0$.

As interdependências funcionais são representadas a partir de uma única matriz com dimensão $N \times N$, chamada de $functional$. Dessa forma, $functional_{ij} = 1$, tem-se uma interdependência **REQUIRES** entre os requisitos r_i e r_j. Assim, r_i só pode ser incluído no release caso r_j também o seja, mas r_j pode ser selecionado sem a presença de r_i. No caso em que $functional_{ij} = functional_{ji} = 1$, tem-se uma relação **AND** entre os requisitos r_i e r_j, ou seja, os dois requisitos devem ser incluídos juntos no *release*.

As interdependências de valor **CVALUE** e **ICOST** são representadas com matrizes de dimensão $N \times N$, $cvalue$ e $icost$ respectivamente. São compostas por valores contidos no intervalo $[-1, 1]$ que indicam a porcentagem de influência na importância e custo entre os requisitos. Por exemplo, quando $cvalue_{ij} = 0,2$, a inclusão do requisito

r_i no *release* resulta em um aumento da importância do requisito r_j em 20%. Analogamente, $icost_{ij} = -0,4$ indica que, caso o requisito r_i seja selecionado para a *release*, o custo de desenvolvimento de r_j diminui em 40%.

4. Operadores Genéticos Propostos

Conforme exposto em [Del Sagrado et al. 2011], interações entre requisitos podem adicionar novas restrições ao problema, o que implica na necessidade de adaptação das técnicas de busca a serem empregadas. Isto é, quando se pretende resolver o NRP heuristicamente, o fato de considerar tais relações impacta diretamente no espaço de busca. A seguir apresentam-se os operadores adaptados elaborados neste trabalho.

4.1. Gerador de Soluções

Verifica-se no Algoritmo 1 a concepção proposta para geração de soluções considerando interdependências. Primeiramente, define-se aleatoriamente um número desejado de requisitos a serem incluídos na solução de acordo com o intervalo $[0, numRequisitos]$. Enquanto a quantidade de requisitos for menor que quantidade desejada de requisitos, seleciona-se um requisito aleatório e o adiciona em conjunto com suas dependências à solução. A cada nova iteração, os novos requisitos adicionados são contabilizados por meio da variável $qtdRequisitos$. Vale destacar que este processo de geração de soluções não quebra restrições de precedência.

Algoritmo 1: Gerador de Soluções

```
1  início
2      qtdDesejadaRequisitos ← gerarNumAleatório(numRequisitos);
3      qtdRequisitos ← 0;
4      enquanto qtdRequisitos < qtdDesejadaRequisitos faça
5          requisito ← selecionarRequisitoAleatório(solução);
6          adicionar(requisito, solução);
7          qtdRequisitos ← qtdRequisitos + 1;
8          numDependencias ← 0;
9          numDependencias ← adicionarDependencias(requisito, solução);
10         qtdRequisitos ← qtdRequisitos + numDependencias;
11     fim
12     reparar(solução);
13     retorna solução;
14 fim
```

4.2. Cruzamento

Dado um conjunto de requisitos, suas relações de precedência podem ser representadas como um conjunto de grafos acíclicos, orientados e conexos. Nos grafos G_i da Figura 1 cada vértice representa um requisito e cada aresta indica as respectivas dependências. Nessa figura, exemplifica-se uma possível solução, na qual cada vértice valorado indica se o requisito foi incluso (1) ou não (0). Uma solução é válida apenas se todos os requisitos incluídos têm suas precedências também incluídas.

Antes de se discutir o algoritmo de cruzamento proposto, é relevante definir o processo de combinação de grafos. Assim, utilizando como base o grafo G_1 da Figura 1, define-se a combinação de configurações de grafos entre duas soluções de acordo com os seguintes passos:

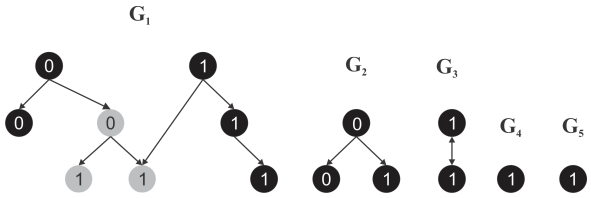


Figura 1. Representação das precedências e de uma solução em forma de grafo

1. Seleciona-se aleatoriamente um vértice presente em G_1 e suas dependências. Nesse caso, os vértices em cinza.
2. Então, define-se dois grafos: G_{11} para representar G_1 com configurações do **pai 1** e G_{21} para o **pai 2**. A combinação entre os grafos G_{11} e G_{21} ocorre por meio da permutação das configurações dos vértices selecionado anteriormente entre as duas soluções, como mostrado na Figura 2.

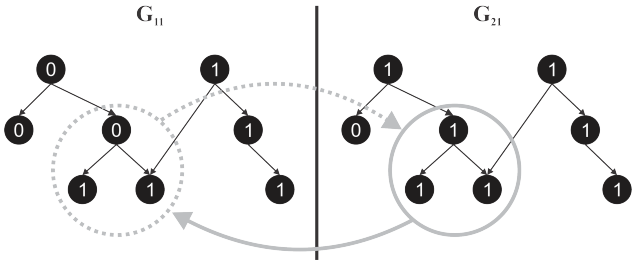


Figura 2. Combinação de subgrafos conexos entre duas soluções a partir do grafo G_1

3. Realizada a permutação definida no passo 2, geram-se os grafos G'_{11} e G'_{21} que correspondem ao **filho 1** e **filho 2**, respectivamente. Após isso, é necessário verificar se os vértices que se ligam ao subgrafo formado pelos vértices selecionados estão com todas suas precedências incluídas. A Figura 3 ilustra que um requisito em G'_{21}, vértice indicado pela seta, não teve todas suas precedências incluídas após a combinação.

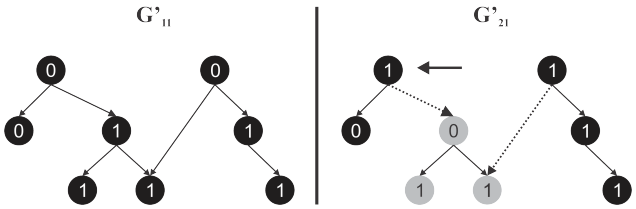


Figura 3. Subgrafos conexos gerados a partir da combinação entre G_{11} e G_{21}

4. Para lidar com a situação decorrente do passo anterior, escolhe-se aleatoriamente entre remover esse requisito e seus dependentes ou adicionar suas precedências. A Figura 4 ilustra a remoção do requisito que quebra a restrição de precedência.

O Algoritmo 2 reflete as etapas acima. Primeiramente, seleciona-se $p\%$ dos grafos (pai 1 e pai 2) em G para serem combinados e, posteriormente, serem incluídos nas

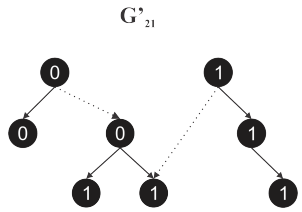


Figura 4. Subgrafo G'_{21} final após remoção de requisito

configurações dos filhos oriundos do cruzamento. No caso dos grafos não selecionados, opta-se por apenas copiar as configurações dos pais para os respectivos filhos.

Algoritmo 2: Operador de Cruzamento

Entrada: G, Pai 1 e Pai 2
Saída: Filho 1 e Filho 2
Dados: C: Conjunto de grafos a serem combinados
C': Conjunto de dados a serem copiados

início
  Inclua aleatoriamente em C, $p\%$ da quantidade de grafos de G;
  Inclua em C', $G - C$;
  para cada $G_i \in C$ **faça**
    Combine G_{1i} e G_{2i} e inclua os resultados em G'_{1i} e G'_{2i};
  fim
  para cada $G_i \in C'$ **faça**
    Copie G_{1i} para G'_{1i};
    Copie G_{2i} para G'_{2i};
  fim
fim

4.3. Mutação

No Algoritmo 3 apresenta-se a estratégia de mutação empregada na presente proposta. A princípio, seleciona-se aleatoriamente um requisito. Se tal requisito está incluso na solução, remove-se recursivamente ele e seus dependentes. Caso contrário, adiciona-se o requisito selecionado e suas respectivas precedências recursivamente.

Algoritmo 3: Operador de Mutação

início
  $requisito \leftarrow$ selecioneRequisitoAleatorio();
  se *estaPresente($requisito$, solução)* **então**
    remover($requisito$);
    removerDependencias($requisito$);
  fim
  senão
    adicionar($requisito$);
    adicionarDependencias($requisito$);
  fim
fim

4.4. Reparo

Objetivando lidar de forma mais adequada com a restrição de orçamento, propõe-se um operador exclusivamente para reparar soluções que não atendem tal quesito. Os vértices cinzas na Figura 5 representam os requisitos sem dependentes, na solução em questão, isto é, aqueles cujos dependentes têm valor 0 ou simplesmente não existem. Requisitos com interdependência do tipo **AND** entre si são tratados como se fossem um único requisito.

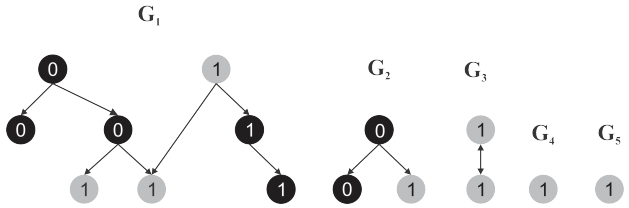


Figura 5. Requisitos sem dependentes inclusos na solução

Dessa forma, o operador de reparo descrito no Algoritmo 4 consiste em, iterativamente, listar os requisitos sem dependentes na solução e retirar aleatoriamente um deles da solução, até que esta satisfaça a restrição de orçamento.

Algoritmo 4: Operador de Reparo

```
   Entrada: Solução a ser reparada
1  início
2      enquanto Custo(solução) > Orçamento faça
3          reqs_sem_dependentes ← getReqsSemDependentes(solução);
4          req_aleatorio ← selecionarRequisitoAleatório(reqs_sem_dependentes);
5          remover(req_aleatorio, solução);
6      fim
7  fim
```

Destaca-se que cada remoção de requisito da solução, linha 5, gera um novo conjunto de requisitos sem dependentes. Na próxima iteração, esse novo conjunto será retornado pela função *getReqsSemDependentes(solução)* (linha 3). Assim, havendo requisitos na solução, haverá também um conjunto de requisitos sem dependentes na solução.

5. Avaliação Empírica

Foram compostos seis algoritmos em prol de uma análise mais detalhada sobre o impacto de cada operador. A Tabela 1 mostra as configurações dos parâmetros e operadores empregados em cada variação do AG, na qual "x" representa a presença do correspondente operador adaptado, enquanto "-" representa a presença do operador canônico. AG_1 (que contém apenas operadores canônicos) foi utilizado como base de comparação para as demais variações. O AG_2 foi avaliado para averiguar a influência da presença do Gerador de Soluções e, analogamente, para o AG_3 e o AG_4 com seus respectivos operadores propostos. A avaliação do AG_5 reflete o impacto da ausência do operador de reparação e o AG_6 representa a influência relativa à presença dos quatro operadores propostos agregados. Todos os algoritmos foram executados com 100, 500 e 1000 gerações.

Neste trabalho foram utilizadas instâncias artificiais e reais. As artificiais variam em número de requisitos e porcentagem de interdependências entre requisitos. Os nomes

Tabela 1. Algoritmos testados e seus respectivos parâmetros

Configurações	AG_1	AG_2	AG_3	AG_4	AG_5	AG_6
Probabilidade de Cruzamento	90%	90%	90%	90%	90%	90%
Probabilidade de Mutação	1%	1%	1%	50%	50%	50%
Tamanho da População	100	100	100	100	100	100
Operador de Gerador de Solucão	-	x	-	x	x	x
Operador de Cruzamento	-	-	x	-	x	x
Operador de Mutação	-	-	-	x	x	x
Operador de Reparo	-	-	-	-	-	x

das instâncias estão no formato I_R_D, onde R é a quantidade de requisitos e D a densidade de interdependências. Por exemplo, uma instância I_25_50 teria 25 requisitos e 50% dos seus requisitos teriam pelo menos uma precedência e no máximo 50% dos requisitos como precedência. As duas instâncias reais utilizadas foram obtidas junto ao trabalho [Karim and Ruhe 2014]. A primeira, nomeada dataset-1, possui 50 requisitos e 68% de interdependências, enquanto a segunda (dataset-2) é composta por 25 requisitos e 52% de interdependências. O orçamento para cada instância foi assumido como sendo uma porcentagem da soma dos custos de todos os requisitos. Para as instâncias artificiais esse orçamento é de 60%, 50%, 40%, 30% e 20% para as instâncias com 25, 50, 100, 250 e 500 requisitos, respectivamente. Já para as instâncias reais, assumiu-se um valor de 60%.

Conforme sugerido em [Arcuri and Briand 2014], tendo em vista o caráter estocástico dos algoritmos, estes foram executados 30 vezes para cada instância, coletando-se a média e o desvio padrão dos resultados das métricas *fitness* e tempo de execução.

5.1. Resultados e Análises

A Tabela 2 mostra as médias de *fitness* e de tempo de execução, bem como seus respectivos desvios padrões de cada algoritmo para cada instância.

Todos os algoritmos que utilizam pelo menos um operador proposto, em média apresentam maiores valores de *fitness* para todas as instâncias. Sendo que para todas as instâncias artificiais os AG_2, AG_3, AG_4, AG_5 e AG_6 apresentam um percentual de melhora em relação ao AG_1 referente à métrica *fitness* de respectivamente de 3.94%, 3.21%, 3.77%, 11.25% e 12.45%, e referente à métrica tempo de 12.91%, -4.88%, 36.69%, 12.06%, -1.77%. Vale destacar que a adição de cada operador acrescenta uma melhora em *fitness* e em tempo, com exceção do operador de cruzamento que acrescenta 4,88% em tempo de execução. Além disso, o AG_6 apresenta melhores valores de *fitness* para todas as instâncias. Para as instâncias maiores, isto é, aquelas com pelo menos 250 requisitos, a melhora em *fitness* é de 18.49%.

Com relação às interdependências, tem-se que para as instâncias com 5% de interdependências entre requisitos a média de melhora em *fitness* do AG_6 foi de 8.72% com um acréscimo de tempo de execução de 0.98%. Para as instâncias maiores a média de melhora é respectivamente de 14.86% com um tempo de execução 4.43% menor. Considerando apenas as instâncias com 25% de interdependências a melhora é de 10.82% com um acréscimo de 3.88% em tempo. Para as instâncias maiores a melhora é de 17.14% com o tempo de execução 13.00% menor. Executando o AG_6 para instâncias com 50% de interdependências, a melhora alcançada foi 15.94% com um acréscimo de tempo de 6.37%,

Tabela 2. Resultados dos algoritmos executados por 100 gerações

Instância	AG_1	AG_2	AG_3	AG_4	AG_5	AG_6	
I_S_25_5	94.81+/-1.8	95.98+/-1.08	96.14+/-1.03	96.53+/-0.66	96.69+/-0.39	96.76+/-0.0	**Fitness**
	80.57+/-66.63	29.3+/-14.27	37.13+/-17.83	25.96+/-14.39	52.29+/-88.99	46.73+/-25.36	**Tempo (ms)**
I_S_25_25	101.95+/-8.22	105.51+/-9.61	105.39+/-9.48	106.15+/-9.64	106.36+/-9.68	106.43+/-9.67	**Fitness**
	53.5+/-54.35	28.38+/-10.16	34.18+/-12.97	24.86+/-10.25	40.28+/-64.08	41.54+/-18.72	**Tempo (ms)**
I_S_25_50	100.96+/-9.62	111.13+/-11.18	103.88+/-9.64	111.74+/-11.16	111.96+/-11.18	112.03+/-11.18	**Fitness**
	43.71+/-46.49	27.71+/-8.36	32.29+/-10.94	24.38+/-8.41	35.98+/-52.7	37.93+/-16.13	**Tempo (ms)**
I_S_50_5	123.51+/-39.95	131.34+/-36.36	126.66+/-40.32	132.19+/-36.72	132.74+/-37.28	132.86+/-37.36	**Fitness**
	49.57+/-41.52	37.75+/-18.86	46.79+/-26.86	33.02+/-16.65	46.86+/-49.41	58.45+/-38.26	**Tempo (ms)**
I_S_50_25	136.72+/-44.45	143.77+/-40.98	140.29+/-45.31	145.17+/-41.88	146.48+/-43.22	146.73+/-43.43	**Fitness**
	54.88+/-38.65	47.65+/-26.12	56.38+/-30.77	41.38+/-22.4	54.14+/-46.53	69.0+/-40.23	**Tempo (ms)**
I_S_50_50	143.47+/-43.39	151.24+/-41.03	146.37+/-43.6	153.38+/-42.44	155.15+/-43.96	155.45+/-44.19	**Fitness**
	55.9+/-35.36	49.98+/-24.45	59.46+/-28.95	43.4+/-20.95	56.63+/-42.85	71.82+/-37.27	**Tempo (ms)**
I_S_100_5	167.76+/-71.81	174.69+/-68.89	171.74+/-74.12	176.86+/-69.66	179.47+/-72.16	179.72+/-72.16	**Fitness**
	80.26+/-68.17	77.47+/-71.1	93.92+/-88.6	66.51+/-59.87	85.4+/-80.88	110.66+/-101.21	**Tempo (ms)**
I_S_100_25	188.66+/-87.02	195.84+/-85.39	194.49+/-91.83	197.11+/-84.47	203.47+/-92.66	203.95+/-93.11	**Fitness**
	98.69+/-80.39	99.37+/-88.34	117.79+/-104.22	83.36+/-71.64	107.09+/-94.95	139.56+/-121.72	**Tempo (ms)**
I_S_100_50	197.33+/-85.66	205.91+/-85.5	203.32+/-90.14	207.24+/-84.71	217.2+/-95.66	218.19+/-96.61	**Fitness**
	110.23+/-82.54	108.74+/-87.47	130.63+/-104.78	91.88+/-71.75	120.92+/-97.71	154.69+/-122.5	**Tempo (ms)**
I_S_250_5	242.16+/-157.17	249.29+/-153.37	250.28+/-164.82	249.11+/-149.18	264.16+/-167.58	265.45+/-168.84	**Fitness**
	263.12+/-465.97	275.94+/-509.42	315.66+/-564.31	183.54+/-284.01	246.37+/-387.82	305.95+/-468.45	**Tempo (ms)**
I_S_250_25	270.47+/-174.58	275.62+/-168.39	280.67+/-184.24	273.67+/-162.15	300.75+/-197.32	304.26+/-202.43	**Fitness**
	392.4+/-604.49	390.0+/-605.7	465.39+/-717.08	265.44+/-375.33	355.73+/-506.56	409.49+/-553.84	**Tempo (ms)**
I_S_250_50	284.62+/-173.72	294.42+/-172.96	296.57+/-184.2	291.18+/-165.87	331.7+/-215.02	336.87+/-221.96	**Fitness**
	466.72+/-629.82	437.76+/-601.34	547.37+/-738.8	307.69+/-385.87	424.19+/-535.57	474.31+/-572.22	**Tempo (ms)**
I_S_500_5	329.71+/-228.62	337.84+/-224.16	344.28+/-242.18	330.0+/-208.6	378.11+/-261.81	386.56+/-274.09	**Fitness**
	1116.17+/-2332.76	1159.46+/-2576.29	1305.78+/-2723.25	542.33+/-896.85	786.24+/-1356.62	835.47+/-1366.69	**Tempo (ms)**
I_S_500_25	352.73+/-235.46	360.15+/-230.56	366.84+/-247.19	351.66+/-215.7	417.48+/-289.52	429.62+/-306.41	**Fitness**
	1587.72+/-2819.04	1471.28+/-2727.55	1841.18+/-3259.47	778.8+/-1217.03	1107.68+/-1747.94	1105.97+/-1638.85	**Tempo (ms)**
I_S_500_50	350.95+/-227.67	372.9+/-227.9	364.5+/-239.13	369.62+/-219.05	445.93+/-299.32	461.38+/-319.03	**Fitness**
	1731.73+/-2777.4	1583.74+/-2668.81	2013.92+/-3215.3	900.37+/-1261.29	1312.56+/-1855.17	1271.26+/-1699.95	**Tempo (ms)**
dataset-1	3868.69+/-170.46	5080.19+/-135.42	3967.0+/-241.09	5175.06+/-103.04	5293.73+/-45.68	5301.33+/-39.68	**Fitness**
	73.66+/-15.73	83.66+/-12.11	90.59+/-16.14	74.63+/-12.76	97.13+/-15.59	100.59+/-21.3	**Tempo (ms)**
dataset-2	4417.01+/-565.26	5029.39+/-128.39	4491.08+/-553.22	5108.3+/-104.52	5172.83+/-126.8	5180.16+/-125.31	**Fitness**
	51.45+/-24.9	56.78+/-28.27	62.63+/-30.29	49.56+/-26.75	64.01+/-34.98	65.96+/-37.84	**Tempo (ms)**

todavia para as instâncias maiores a melhora chega a 24.35% com um decréscimo de 12.48% do tempo de execução. Esses resultados atestam que quanto maior as instâncias e a densidade de interdependências, mais efetiva é a abordagem proposta.

Em relação às instâncias reais, a melhora foi de 37% para o dataset-1 e 17% para o dataset-2. Além disso, os operadores de geração de solução e de mutação, mesmo sozinhos já melhoram de forma considerável o processo de otimização para essas instâncias e as melhorias adicionadas pelo reparador e o cruzamento não são tão significantes.

Destaca-se que para algumas instâncias, os valores obtidos pelo AG_6 com 100 gerações, foram superiores aos do AG_1 com 1000 gerações. Estes e outros resultados, por restrições de espaço, encontram-se disponíveis online no material de suporte da pesquisa[1].

6. Considerações Finais

Diversos aspectos podem influenciar na tomada de decisão sobre quais requisitos devem ser implementados no próximo *release* o que faz desta, uma tarefa complexa. Um dos fatores que aumentam a dificuldade da seleção são as interações entre requisitos, cuja influência também impacta diretamente no espaço de busca. Por isso, para se alcançar bons resultados, em alguns casos requisita-se a implementação de adaptações nos algoritmos de busca tradicionais afim de que estes consigam lidar melhor com todas essas questões.

Dessa forma, objetivou-se nesse trabalho propor e analisar alguns operadores de um AG adaptados para o Problema do Próximo *Release* (NRP) com interdependências. A

[1]http://goes.uece.br/italoyeltsin/operadores-adaptados-nrp

partir da avaliação empírica realizada, constatou-se que dentre as variações de operadores e parâmetros apresentadas, o AG_6 se destacou em termos de *fitness*, sendo tal vantagem mais significante para as instâncias com maiores quantidades de requisitos e de interdependências. Além disso, para algumas instâncias o AG_6 com apenas 100 gerações atingiu resultados (de *fitness*) melhores do que o AG_1 com 1000.

Como trabalhos futuros, pretende-se elaborar novas análises relativas aos parâmetros empregados, avaliar outras configurações de tipos de operadores e utilizar instâncias reais compostas por maior quantidade de requisitos e interdependências.

Referências

Arcuri, A. and Briand, L. (2014). A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. *Software Testing, Verification and Reliability*, 24(3):219–250.

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

Baker, P., Harman, M., Steinhofel, K., and Skaliotis, A. (2006). Search based approaches to component selection and prioritization for the nrp. In *Software Maintenance, 2006. ICSM'06. 22nd IEEE International Conference on*, pages 176–185. IEEE.

Carlshamre, P., Sandahl, K., Lindvall, M., Regnell, B., and Natt och Dag, J. (2001). An industrial survey of requirements interdependencies in software product release planning. In *Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on*, pages 84–91. IEEE.

Dahlstedt, Å. G. and Persson, A. (2005). Requirements interdependencies: state of the art and future challenges. In *Engineering and managing software requirements*, pages 95–116. Springer.

Del Sagrado, J., Aguila, I., and Orellana, F. (2011). Requirements interaction in the next release problem. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, pages 241–242. ACM.

Harman, M., McMinn, P., De Souza, J. T., and Yoo, S. (2012). Search based software engineering: Techniques, taxonomy, tutorial. In *Empirical Software Engineering and Verification*, pages 1–59. Springer.

Karim, M. R. and Ruhe, G. (2014). Bi-objective genetic search for release planning in support of themes. In *Proceedings of the 6th International Symposium on Search-Based Software Engineering*, volume 8636, pages 123–137, Fortaleza, Brazil. Springer.

Nascimento, T. and Souza, J. (2012). Uma abordagem aco para o problema do próximo release com interdependência de requisitos. In *III Workshop em Engenharia do Software Baseada em Busca*.

Yeltsin, I., Paixão, M., and Souza, J. (2013). Uma adaptação de algoritmo genético para o problema do próximo release com interdependências entre requisitos. In *IV Workshop em Engenharia de Software Baseada em Busca*.

Zhang, Y., Harman, M., and Mansouri, S. A. (2007). The multi-objective next release problem. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1129–1137. ACM.

Otimizando o Projeto de Arquitetura de Linha de Produto de Software com Muitos Objetivos: um Estudo Exploratório

Marcelo C. B. Santos, Wainer Moschetta, Thelma E. Colanzi, Edson OliveiraJr

Departamento de Informática – Universidade Estadual de Maringá (UEM)
CEP 87.020–900 – Maringá – PR – Brasil

{uemgmarcelo,wainersm}@gmail.com, {thelma,edson}@din.uem.br

Abstract. *The MOA4PLA approach provides search operators and objective functions to optimize and evaluate Software Product Line Architecture (PLA) design by using multi-objective search algorithms (MOAs). However, some of these functions involve metrics with different ranges related to different architectural properties, which affect the function sensibility. This fact motivated an evaluation model refinement, which resulted in six new objective functions. Given the lack of studies on application of MOAs involving many objectives to optimize PLA design, the main goal of this work is presenting an exploratory study about the application of the MOAs NSGA-II and PAES using six objectives to optimize three PLA designs. A behavior analysis of such MOAs in the referred context provided initial evidence that NSGA-II achieved better results than PAES. This work also presents lessons learned from the study carried out.*

Resumo. *A abordagem MOA4PLA propõe operadores de busca e funções objetivo para otimizar e avaliar o projeto de Arquitetura de Linha de Produto de Software (ALP) utilizando algoritmos de busca multiobjetivos (MOAs). No entanto, algumas dessas funções envolvem métricas de diferentes grandezas e referentes a diferentes propriedades arquiteturais, o que influencia na sensibilidade da função. Isso motivou um refinamento do modelo de avaliação originando seis novas funções objetivo. Dada a inexistência de trabalhos nos quais MOAs envolvendo muitos objetivos tenham sido aplicados para otimizar o projeto de ALP, este trabalho tem como principal objetivo apresentar um estudo exploratório no qual os MOAs NSGA-II e PAES foram aplicados usando seis objetivos para otimizar três projetos de ALP. Uma análise do comportamento desses dois MOAs no referido contexto forneceu evidências iniciais de que o NSGA-II alcançou melhores resultados do que o PAES. Este trabalho apresenta também lições aprendidas com a realização do estudo.*

1. Introdução

O objetivo da abordagem de Linha de Produto de Software (LPS) é desenvolver uma família de produtos de software para um determinado domínio com base na reutilização de artefatos de software [Linden et al. 2007]. A Arquitetura de Linha de Produto de Software (ALP) define um projeto comum para os produtos a serem derivados da LPS, por isso é de extrema importância ter uma ALP modular e bem projetada, visando facilitar a manutenção e a evolução da LPS. Uma ALP inclui as características obrigatórias aos produtos da LPS e as características opcionais ou alternativas. Uma característica *(feature)* é uma capacidade do sistema que é relevante e visível para o usuário final.

Modularizar uma ALP não é uma tarefa fácil, pois existem várias alternativas para definir a estrutura dos módulos, as características do produto podem estar distribuídas em

mais de um módulo, há diversos pontos de variação e variantes [Lopez-Herrejon et al. 2005]. Técnicas da Engenharia de Software Baseada em Busca *(Search Based Software Engineering - SBSE)* têm mostrado bons resultados em descobrir soluções próximas das ideais em problemas difíceis como o projeto de ALP [Aleti et al. 2013]. Nesse contexto, foi proposta a abordagem de otimização multiobjetivo denominada *Multi-Objective Approach for Product-Line Architecture Design (MOA4PLA)* [Colanzi et al. 2014]. A MOA4PLA é uma abordagem sistemática e automatizada que usa algoritmos de busca multiobjetivos (MOEAs) para avaliar e melhorar o projeto de ALP.

A MOA4PLA produz um conjunto com as soluções de melhor *trade-off* entre os vários objetivos otimizados. A abordagem inclui um modelo de avaliação para projeto de ALP composto de funções objetivo, as quais são utilizadas para avaliar cada solução gerada no processo de otimização. O modelo de avaliação da MOA4PLA inclui quatro funções objetivo: CM, FM, Ext e Eleg, as quais utilizam métricas de software para medir: princípios básicos de projeto, modularização de características, extensibilidade e elegância do projeto de ALP, respectivamente.

Um estudo empírico realizado por Colanzi et al. (2014) utilizando as funções CM e FM forneceu indícios de que o MOEA NSGA-II *(Non-dominated Sorting Genetic Algorithm)* [Deb et al. 2002] é capaz de obter melhores resultados que o PAES *(Pareto Archive Evolution Strategy)* [Knowles e Corne 2000] em termos de Hipervolume e Distância Euclidiana à solução ideal, quando da aplicação da MOA4PLA para otimizar projetos de ALPs. No entanto, CM e FM envolvem métricas de diferentes grandezas referentes a diferentes propriedades arquiteturais, o que prejudica a sensibilidade da função objetivo em avaliar o projeto de ALP alcançado no processo de busca já que uma métrica com maior grandeza exerce maior influência no valor final da função [Guizzo et al. 2014]. Isso motivou um estudo cujo principal objetivo foi refinar o modelo originalmente proposto em [Colanzi et al. 2014], no qual seis novas funções, separadas por propriedades arquiteturais, foram propostas a partir das funções CM e FM. As novas funções propostas são acrescentadas ao modelo de avaliação original da MOA4PLA, o qual passa a conter as seis novas funções além de Eleg e Ext.

Considerando as funções objetivo derivadas de CM e FM, um novo estudo se faz necessário para avaliar o comportamento de MOEAs, tais como NSGA-II e PAES, durante a otimização de projetos de ALP na presença de muitos objetivos. Os MOEAs possuem algumas limitações quando aplicados a problemas com muitos objetivos, tais como: uma grande parte da população é constituída de soluções não dominadas, a avaliação da diversidade é computacionalmente cara e o cruzamento pode ser ineficiente [Deb e Jain 2012]. Somente estudos envolvendo dois objetivos durante a otimização de projetos de ALP foram encontrados na literatura [Colanzi e Vergilio 2014a, Colanzi et al. 2014, Colanzi e Vergilio 2014b, Guizzo et al. 2014]. Assim, fica evidente a importância de experimentar a aplicação de MOEAs com muitos objetivos para otimizar projetos de ALPs já que esse problema é naturalmente influenciado por vários fatores.

Dessa forma, o objetivo deste artigo é apresentar um estudo exploratório realizado para avaliar o desempenho dos MOEAs NSGA-II e PAES com seis funções objetivo. Para analisar os resultados alcançados, foram utilizados os indicadores de qualidade: Hipervolume e Distância Euclidiana à solução ideal. A partir desse estudo foi possível aprender algumas lições que permitem guiar novos estudos empíricos. Tais lições constituem uma das principais contribuições deste artigo já que até o momento nenhum outro estudo foi publicado envolvendo seis objetivos para a otimização de projetos de ALPs por meio de MOEAs.

Este artigo está organizado em seções. A Seção 2 descreve a abordagem MOA4PLA. A Seção 3 apresenta as funções objetivo propostas e a Seção 4 aborda a descrição do estudo exploratório. As Seções 5 e 6 apresentam os resultados e as lições aprendidas, respectivamente. E, a Seção 7 conclui o artigo e direciona trabalhos futuros.

2. Abordagem MOA4PLA

A MOA4PLA é uma abordagem sistemática e automatizada que usa MOEAs para avaliar e melhorar o projeto de ALP em termos de princípios básicos de projeto, modularização de características e extensibilidade de LPS [Colanzi et al. 2014]. A abordagem utiliza um metamodelo que representa as ALPs e possui operadores de busca específicos para otimizar projetos de ALP, tais como o operador de mutação e o operador de cruzamento dirigidos a características [Colanzi e Vergilio 2014a; Colanzi e Vergilio 2014b]. Após a aplicação dos operadores de busca, cada solução é avaliada de acordo com o modelo de avaliação (vide Seção 2.1) estabelecido pelo arquiteto.

O objetivo da MOA4PLA é a melhoria do projeto de uma ALP, independentemente do MOEA adotado. Por isso, diferentes MOEAs podem ser aplicados no processo de otimização. A Otimização Multiobjetivo consiste em otimizar simultaneamente vários interesses interdependentes e muitas vezes conflitantes e, por isso, não há uma única solução. Assim, o objetivo é encontrar o conjunto de soluções que melhor representa os diversos interesses definidos nos objetivos. Para isso é utilizado o conceito de dominância de Pareto [Pareto 1927] que permite comparar soluções considerando todos os objetivos do problema. As soluções são chamadas de dominadas e não-dominadas. As não-dominadas são as melhores e formam uma aproximação à fronteira de Pareto *(PF_{approx})*. A união de todos os PF_{approx}, excluindo as soluções dominadas, formam o PF_{known}, que representa as melhores soluções encontradas por um algoritmo para um determinado problema. O PF_{true} representa a fronteira de Pareto ótima e é obtido por meio da união de todos os PF_{knowns} removendo as soluções dominadas e repetidas, desse modo PF_{true} contém as melhores soluções encontradas para o problema.

Colanzi e Vergilio (2014a) realizaram um estudo envolvendo nove ALPs para comparar o desempenho dos MOEAs NSGA-II e PAES na otimização de projeto de ALP. A principal motivação foi avaliar diferentes estratégias de otimização e, por isso, o NSGA-II foi aplicado em duas versões, sendo uma com cruzamento e outra sem cruzamento, além do PAES que naturalmente não utiliza cruzamento. Nesse estudo, foram utilizados dois objetivos envolvendo a otimização das funções: CM e FM. O NSGA-II obteve melhores resultados em termos de Hipervolume (HV) e de Distância Euclidiana à solução ideal (DE), comparado ao PAES. E a versão do NSGA-II com cruzamento obteve soluções ainda melhores que a versão só com mutação. HV [Zitzler et al. 2003] mede a área de cobertura de uma fronteira de Pareto conhecida (PF_{known}) sobre o espaço de busca. DE mede a distância entre uma solução e a solução ideal em um espaço de objetivos, sendo a solução ideal uma solução com os melhores valores possíveis em todos os objetivos do problema [Cochrane e Zeleny 1973].

2.1 Modelo de avaliação da MOA4PLA

A definição do modelo de avaliação abrange a seleção de métricas e a construção de funções objetivo. No caso da MOA4PLA, o modelo de avaliação inclui métricas convencionais para medir princípios básicos de projeto, tais como acoplamento e coesão, além de métricas específicas para LPS, tais como métricas dirigidas a características e de

extensibilidade de ALP [Colanzi et al. 2014]. A Tabela 1 apresenta uma breve descrição das métricas utilizadas no modelo de avaliação da MOA4PLA.

Tabela 1. Métricas utilizadas no Modelo de Avaliação da MOA4PLA

Atributo	Métrica	Definição
Métricas dirigidas a Características [Sant'Anna 2008] (Função FM)		
Difusão de Características	CDAC	Número de componentes que contribuem para a realização de uma dada característica.
	CDAI	Número de interfaces que contribuem para a realização de uma dada característica.
	CDAO	Número de operações que contribuem para a realização de uma dada característica.
Interação de Características	CIBC	Número de características com os quais uma dada característica compartilha ao menos um componente.
	IIBC	Número de características com os quais uma dada característica compartilha ao menos uma interface.
	OOBC	Número de características com os quais uma dada característica compartilha ao menos uma operação.
Coesão	LCC	Número de características com os quais um dado componente está associado.
Métricas Convencionais [Wüst 2014] (Função CM)		
Coesão Relacional	H	Coesão relacional do projeto por meio do número médio de relacionamentos entre classes dentro de um componente.
Acoplamento	DepPack	Número de pacotes dos quais as classes e interfaces de um dado pacote dependem.
	CDepIn	Número de elementos arquiteturais que dependem dessa classe.
	CDepOut	Número de elementos arquiteturais dos quais essa classe depende.
	DepIn	Número de dependências UML em que o pacote é o fornecedor.
	DepOut	Número de dependências UML em que o pacote é o cliente.
Tamanho	NumOps	Número de operação por interface.
Métrica de Extensibilidade [OliveiraJr et al. 2013] (Função Ext)		
Extensibilidade	Extens	Soma da capacidade de extensão da LPS em termos de abstração de cada um dos componentes da ALP
Métricas de Elegância [Simons e Parmee 2012] (Função Eleg)		
Elegância	NAC	Elegância de números entre classes, representada pelo desvio padrão dos números de atributos e métodos entre as classes de um projeto.
	EC	Elegância de acoplamentos externos das classes de um projeto
	ATMR	Desvio padrão da razão entre atributos e métodos dentro das classes de um projeto

As métricas para avaliar princípios básicos de projeto são denominadas convencionais. A função objetivo CM é composta pela soma de várias métricas convencionais (apresentadas na Tabela 1). A equação da função é apresentada em Eq. 1, onde **c** representa o número de componentes, **itf** representa o número de interfaces e **cl** representa o número de classes de uma dada ALP. CM utiliza a média de DepPack de todos os componentes da ALP e a média de NumOps de todas as interfaces do projeto.

$$\boldsymbol{CM(pla)} = \sum_{i=1}^{c} DepIn + \sum_{i=1}^{c} DepOut + \sum_{i=1}^{cl} CDepIn \sum_{i=1}^{cl} CDepOut + \frac{\sum_{i=1}^{c} DepPack}{c} + \frac{\sum_{i=1}^{itf} NumOps}{Itf} + \frac{1}{\sum_{i=1}^{c} H} \quad (1)$$

A função FM é usada para avaliar o grau de modularização de uma ALP em termos de características. FM consiste do somatório da soma de cada métrica dirigida a características (Tabela 1). A função FM, proposta por Colanzi et al. (2014), é apresentada a seguir, onde **i** representa o número de interfaces e **f** representa o número de características de uma dada ALP.

$$\boldsymbol{FM(pla)} = \sum_{i=1}^{c} LCC + \sum_{i=1}^{f} CDAC + \sum_{i=1}^{f} CDAI + \sum_{i=1}^{f} CDAO + \sum_{i=1}^{f} CIBC + \sum_{i=1}^{f} IIBC + \sum_{i=1}^{f} OOBC \quad (2)$$

A função objetivo Ext apresenta o valor invertido da métrica Extens. Esse cálculo é feito dessa maneira para que seja possível maximizar a extensibilidade da ALP, uma vez que os objetivos são minimizados na MOA4PLA. A seguir, a função proposta:

$$\boldsymbol{Ext(pla)} = \frac{1}{Extens(pla)} \quad (3)$$

O objetivo da função Eleg é melhorar a elegância do projeto de ALP alcançado em cada solução gerada pela MOA4PLA. Para isso, devem-se minimizar o somatório dos valores das métricas NAC, EC e ATMR conforme apresentado em Eq. 4.

$$\boldsymbol{Eleg(pla)} = NAC(pla) + EC(pla) + ATMR(pla) \quad (4)$$

Como mencionado anteriormente, as funções CM e FM envolvem métricas que tratam de diferentes propriedades arquiteturais e com diferentes grandezas. Resultados experimentais obtidos por Guizzo et al. (2014) mostram que algumas soluções obtidas no processo de otimização tinham melhor acoplamento que outras, porém isso não foi identificado no valor de CM. Logo é preciso dispor de um modelo de avaliação mais preciso, capaz de capturar a qualidade do projeto da ALP. Por isso, foi realizado um refinamento do modelo originalmente proposto por Colanzi et al. (2014). As novas funções objetivo, propostas a partir de CM e FM, são apresentadas na próxima seção.

3. Proposta das Novas Funções Objetivo

As novas funções multiobjetivo foram desenvolvidas utilizando as mesmas métricas utilizadas nas funções CM e FM [Colanzi et al. 2014], porém separadas por propriedades arquiteturais com a intenção de se obter avaliações mais precisas sobre o projeto de ALP. A Tabela 2 apresenta as funções objetivo (seu nome e acrônimo) bem como as métricas utilizadas em cada função.

Tabela 2. Novas Funções Objetivo

Função Objetivo	Métricas
Acoplamento de Componentes (ACOMP)	DepIn, DepOut
Acoplamento de Classes (ACLASS)	CDepIn, CDepOut
Tamanho (TAM)	NumOps
Coesão (COE)	H, LCC
Difusão de Características (DC)	CDAI, CDAO, CDAC
Entrelaçamento de Características (EC)	CIBC, IIBC, OOBC

Três das funções propostas ACOMP, ACLASS e TAM são uma agregação de várias métricas convencionais. DC e EC são uma agregação de várias métricas dirigidas a características. As métricas contam o número de elementos arquiteturais associados a cada característica, avaliam as dependências entre características causadas por características que não estejam bem modularizadas. A função COE avalia a coesão do projeto mesclando a métrica convencional H com a métrica dirigida a características LCC. Abaixo são apresentadas as funções desenvolvidas e a descrição de cada uma:

1. **Acoplamento de Componentes:** composta pelas métricas de dependência de entrada e saída. Esta função mede o acoplamento entre os componentes de uma ALP.

2. **Acoplamento de Classes:** mede o número de elementos arquiteturais que dependem de outras classes do projeto mais o número de elementos dos quais cada classe depende.

$$\boldsymbol{ACOMP(pla)} = \sum_{i=1}^{c} \boldsymbol{DepIn} + \sum_{i=1}^{c} \boldsymbol{DepOut} \quad (5)$$

$$\boldsymbol{ACLASS(pla)} = \sum_{i=1}^{cl} \boldsymbol{CDepIn} + \sum_{i=1}^{cl} \boldsymbol{CDepOut} \quad (6)$$

3. **Tamanho:** composta pela métrica que mede o tamanho de operações de uma classe. Esta métrica mede o número de interfaces de uma classe.

4. **Coesão:** composta por métricas que medem a coesão do projeto de ALP em termos dos relacionamentos internos entre as classes dos componentes e do número de características com os quais cada componente está associado.

$$\boldsymbol{TAM(pla)} = \frac{\sum_{i=1}^{itf} \boldsymbol{NumOps}}{\boldsymbol{itf}} \quad (7)$$

$$\boldsymbol{COE(pla)} = \sum_{i=2}^{c} H + \sum_{i=1}^{c} LCC \quad (8)$$

5. **Difusão de Características:** composta por métricas de difusão de características. Esta função é um somatório de métricas que contam o número de elementos arquiteturais associados a cada característica.

6. **Entrelaçamento de Características:** composta por métricas que analisam a interação entre características. Esta função é a soma dos somatórios de métricas que objetiva avaliar as dependências entre características causadas por alguma característica que não esteja bem modularizada.

$$\boldsymbol{DC(pla)} = \sum_{i=1}^{f} CDAI + \sum_{i=1}^{f} CDAO + \sum_{i=1}^{f} CDAC \quad (9)$$

$$\boldsymbol{EC(pla)} = \sum_{i=1}^{f} CIBC + \sum_{i=1}^{f} IIBC + \sum_{i=1}^{f} OOBC \quad (10)$$

4. Descrição do Estudo Exploratório

Esta seção descreve o estudo experimental, concentrando-se na definição, planejamento e execução do experimento. Considerando a motivação do estudo apresentada na introdução, o objetivo do experimento é: Analisar os conjuntos de soluções obtidos pelos MOEAs NSGA-II e PAES usando seis objetivos; com o propósito de avaliar a diversidade e o melhor custo benefício *(trade-off)* entre os objetivos do ponto de vista do pesquisador no contexto do projeto de ALP baseado em busca. Somente as novas funções objetivo foram utilizadas no estudo. As variáveis independentes são NSGA-II e PAES, e as dependentes são o *fitness* das soluções, o melhor *trade-off* entre os objetivos e a diversidade das soluções. Os indicadores de qualidade Hipervolume (HV) e Distância Euclidiana à solução ideal (DE) foram usados para medir, respectivamente, a diversidade das soluções e o *trade-off* entre os objetivos. . O teste de hipótese de Wilcoxon foi utilizado com significância de 95% (p-value $\leq$ 0,05), a fim de verificar a diferença estatística entre os MOEAs com relação à diversidade de soluções.

A hipótese nula do estudo é a hipótese H0, em que não existe diferença significativa entre o PAES e o NSGA-II considerando Hipervolume (HV) e Distância Euclidiana (DE). As duas hipóteses alternativas do estudo são H1 e H2. H1 assume que o PAES é melhor em termos de Hipervolume e Distância Euclidiana que o NSGA-II. Na hipótese H2, o NSGA-II é melhor que o PAES. Essas hipóteses são definidas abaixo em termos dos indicadores utilizados na comparação: HV e DE.

H_0 : $HV_{NSGA\text{-}II} = HV_{PAES}$ **and** $DE_{NSGA\text{-}II} = DE_{PAES}$
H_1 : $HV_{NSGA\text{-}II} < HV_{PAES}$ **and** $DE_{NSGA\text{-}II} > DE_{PAES}$
H_2: $HV_{NSGA\text{-}II} > HV_{PAES}$ **and** $DE_{NSGA\text{-}II} < DE_{PAES}$

4.2 ALPs Utilizadas no Estudo

O experimento envolveu três ALPs, cujas principais informações são apresentadas na Tabela 3. Todas as ALPs são baseadas em componentes e seguem o estilo em camadas. Arcade Game Maker (AGM) é uma LPS criada pelo SEI que inclui três jogos de arcade. Duas versões dessa ALP foram utilizadas: AGM-1 [SEI 2015] e AGM-2 [Contieri Jr et al. 2011]. A Mobile Media (MOM), versão MOM-2 [Contieri Jr et al. 2011], é uma LPS que apoia o gerenciamento de diferentes tipos de mídias.

Observa-se que o valor da função objetivo ACLASS é zero. Isto se justifica pela forma de medição das métricas CDepIn e CDepOut que contam o número de elementos que dependem de uma determinada classe e dos quais essa classe depende, respectivamente, por meio de dependências UML ou dependências de uso. As ALPs utilizadas não possuem esses tipos de dependências, o que justifica os valores zerados.

Tabela 3. Informações sobre os projetos de ALP utilizados no estudo

ALP	# Componentes	# Interfaces	# Classes	# Variabilidades	# *Características*	*Fitness* Original (ACOMP, ACLASS, TAM, COE, DC, EC)
AGM-1	9	14	20	4	9	(28; 0; 35,71; 25,25; 300; 66)
AGM-2	9	14	21	5	11	(31; 0; 3,93; 29,25; 359; 112)
MOM-2	8	13	10	7	13	(27; 0; 6,153; 22,33; 405; 107)

4.3. Operação e Execução do estudo

A OPLA-Tool [Colanzi et al. 2014], ferramenta que apoia a aplicação da MOA4PLA, foi utilizada na execução do experimento. Os MOEAs NSGA-II e PAES foram executados 25 vezes para cada ALP. Os algoritmos foram configurados usando os mesmos parâmetros adotados por Colanzi e Vergilio (2014a). Esses parâmetros são: tamanho da população= 100, taxa de mutação= 0,9 e número de gerações= 300. Nenhum operador de cruzamento foi aplicado no NSGA-II por restrições da ferramenta.

4.4. Ameaças à validade

A principal questão que se deve levar em conta como um risco que pode afetar a validade do experimento é o tamanho da amostra e o tamanho das ALPs utilizadas já que foram utilizados somente três projetos de ALPs acadêmicas e, com ALPs maiores, a solução do problema tende a ficar mais complexa. Todas as métricas utilizadas nas funções objetivo e, por consequência, no experimento, já foram utilizadas em estudos anteriores [Colanzi e Vergilio 2014a, Colanzi et al. 2014, Guizzo et al. 2014]. Outra possível ameaça está relacionada com a aleatoriedade dos algoritmos de busca. Para mitigar essa ameaça, cada algoritmo foi executado 25 vezes, pois isso garante resultados consistentes mesmo com resultados diferentes a cada execução. Além disso, foram adotados os mesmos parâmetros de configuração adotados por Colanzi e Vergilio (2014a). Todas essas ameaças podem ser tratadas em estudos futuros.

5. Análise e Interpretação dos Resultados

Esta seção apresenta os resultados obtidos com a realização do experimento. A Tabela 4 apresenta o número de soluções que formam os conjuntos PF_{true} e PF_{known} para cada ALP. Observa-se que o NSGA-II encontrou um número menor de soluções que o PAES, porém, escolher manualmente um projeto de ALP dentre um conjunto numeroso de soluções encontradas é uma tarefa árdua para o arquiteto. Nota-se que o PF_{true} é formado por soluções encontradas pelos dois algoritmos. O NSGA-II demorou em média 10 minutos em cada execução, enquanto o PAES demorou cerca de 120 minutos.

Na Tabela 5 pode-se observar os valores de *fitness* e de DE das soluções com menor DE encontradas pelo NSGA-II e pelo PAES, além do *fitness* da solução ideal. Os valores de *fitness* são apresentados na seguinte ordem: ACOMP, ACLASS, TAM, COE, DC e EC. Pode-se notar que as soluções encontradas pelo NSGA-II (valores destacados em negrito na tabela) são melhores que as encontradas com o PAES já que têm menor DE em relação à solução ideal. A solução ideal é composta pelos valores mínimos de cada função objetivo alcançados nas soluções que compõem o PF_{true}.

Os valores médios de hipervolume, calculados com base nos resultados de cada uma das 25 execuções de cada um dos MOEAs, podem ser vistos na Tabela 6. As funções objetivo foram normalizadas para o cálculo do hipervolume. Pode-se constatar que os valores de hipervolume do NSGA-II também foram melhores do que os valores do PAES. Os maiores valores de hipervolume estão destacados em negrito.

Tabela 4: Número de Soluções Encontradas

ALP	PF_{true}	PF_{known} do NSGA-II	PF_{known} do PAES
AGM-1	471	52	423
AGM-2	811	161	657
MOM-2	696	119	577

Tabela 5: Soluções de Menor Distância Euclidiana à Solução Ideal

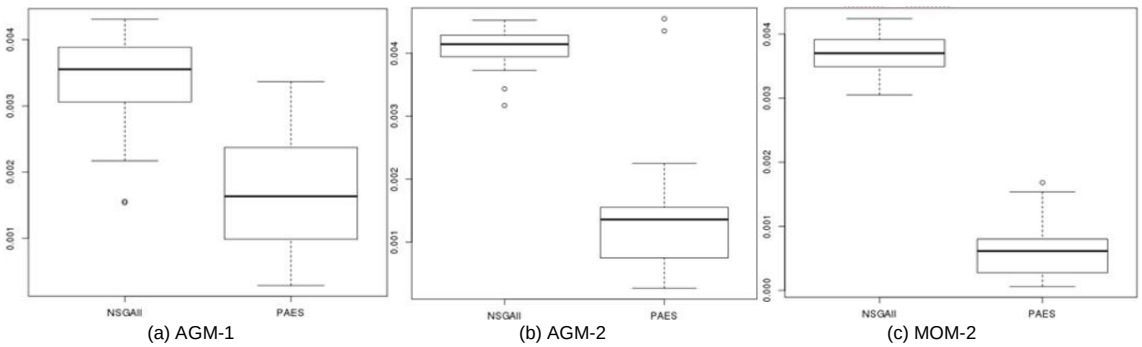
ALP	Solução Ideal (ACOMP, ACLASS, TAM, COE, DC, EC)	NSGA-II		PAES	
		DE	*Fitness* da solução	DE	*Fitness* da solução
AGM-1	(15; 0; 1,09; 15,1; 220; 46)	**8,8**	**(17; 0; 3,8; 22,2; 223; 49)**	226,0	(27; 0; 2,1; 43,0; 430; 124)
AGM-2	(18; 0; 1,08; 23,1; 262; 74)	**8,5**	**(22; 0; 3,6; 27,1; 265; 79)**	199,0	(24; 0; 3,2; 38,1; 455; 120)
MOM-2	(15; 0; 1,16; 17,5; 340; 70)	**13,7**	**(19; 0; 5,7; 20,2; 345; 81)**	419,6	(45; 0; 1,9; 56,0; 740; 187)

Tabela 6: Hipervolume

ALP	NSGA-II		PAES		*p-value*
	Média	Desvio Padrão	Média	Desvio Padrão	
AGM-1	**0,00338**	0,00734	0,00173	0,00084	8,14507
AGM-2	**0,00408**	0,00031	0,00139	0,00105	9,31647
MOM-2	**0,00369**	0,00031	6, 16777	0,00042	1,14507

Após o cálculo dos indicadores de qualidade apresentados anteriormente, o teste de normalidade Shapiro-Wilk foi aplicado sobre os resultados de HV cujo resultado aponta que os dados não seguem uma distribuição normal. Todos os resultados obtiveram relevância estatística (95% de confiança no teste). Então, foi aplicado o teste de Wilcoxon para avaliar a validade da hipótese nula adotada no presente trabalho. Os resultados apontam que existe diferença entre os conjuntos de dados encontrados pelos dois algoritmos com 95% de confiança (vide valores de *p-values* na Tabela 6). A análise dos *boxplots* permite constatar a consistência dos dados obtidos pelos MOEAs bem como qual deles obteve a maior mediana em termos de HV. Como é possível observar na Figura 2, o NSGA-II alcançou a maior mediana para os três conjuntos de dados.

Com base na média de HV e na análise dos *boxplots*, observa-se que o NSGA-II encontra maior diversidade de soluções do que o PAES. Como apresentado anteriormente, NSGA-II também encontra soluções com menores DE do que PAES e, portanto, consegue otimizar mais os objetivos definidos para o processo de otimização. Portanto, com nível de significância de 95% a hipótese nula foi rejeitada. Ainda, o NSGA-II é melhor que PAES e, portanto, fornece indícios de que a hipótese alternativa H2 deve ser aceita, a qual afirma que o NSGA-II é melhor que PAES em termos de HV e DE. Apesar disso, vários pontos foram observados durante este estudo no que diz respeito à aplicação dos MOEAs com muitos objetivos. A partir dessa observação algumas lições foram aprendidas, as quais são abordadas na próxima seção.



(a) AGM-1 (b) AGM-2 (c) MOM-2

Figura 2: Boxplots dos resultados de hipervolume

6. Lições Aprendidas

A seguir são apresentadas as lições aprendidas a partir da condução do experimento. Essas lições servem como base para a realização de futuros estudos.

Número de Soluções Encontradas: O tamanho dos conjuntos de soluções encontrados pelos dois MOEAs fornece evidências de que ambos não conseguem resolver o problema de projeto de ALP na presença de seis objetivos de forma satisfatória do ponto de vista do arquiteto, já que encontram muitas soluções, confirmando as evidências relatadas por Deb e Jain [2012]. Nesse caso, seria melhor conduzir experimentos com um número menor de objetivos.

Tempo de Execução: Considerando que o NSGA-II teve um tempo de execução médio de 10 minutos e o PAES de 120 minutos, pode-se dizer que o tempo de execução do NSGA-II na presença de muitos objetivos é mais viável.

Otimização das Funções: Uma comparação entre os valores de *fitness* das soluções de menor DE (Tabela 5) com os valores de *fitness* originais (Tabela 3) permitiu observar que o NSGA-II conseguiu otimizar grande parte dos objetivos. Por exemplo, o valor original da função ACOMP para a ALP AGM-1 era 28,0. A solução de menor DE encontrada pelo NSGA-II diminuiu o valor do acoplamento entre componentes para 17,0 enquanto o PAES somente reduziu o valor em 1 unidade (27,0). Em outros casos, como o da MOM-2, o NSGA-II conseguiu otimizar o valor enquanto o PAES obteve uma solução de pior valor que o original para essa função. O mesmo ocorreu com as funções de coesão (COE), difusão e entrelaçamento entre características (funções DC e EC). Por outro lado, o PAES conseguiu obter melhores resultados que o NSGA-II em todas as ALPs ao otimizar o tamanho das interfaces do projeto (função TAM). Considerando o acoplamento de classes (ACLASS) os algoritmos não conseguiram otimizar o valor da função uma vez que os valores originais de todas as ALPs eram zero. Isso indica que para estas ALPs a função ACLASS não tinha o que otimizar.

Soluções Próximas do Ideal: O NSGA-II conseguiu encontrar soluções próximas da solução ideal mesmo diante do desafio de otimizar seis objetivos simultaneamente.

Número de Funções: Apesar de usar seis objetivos, o NSGA-II conseguiu valores de *fitness* melhores que os originais. Isso evidencia tanto a viabilidade de aplicação das funções propostas como a sua sensibilidade às mudanças nas propriedades arquiteturais que elas pretendem medir.

Pode-se concluir que o NSGA-II consegue encontrar melhores projetos de ALP do que os projetos originais, porém ele não consegue explorar adequadamente o espaço de busca diante de muitos objetivos. Desse modo, parece ser mais promissor o uso de um número menor de funções objetivo em se tratando da otimização de ALPs.

7. Considerações Finais

Neste estudo foram relatados os resultados de uma avaliação do comportamento dos MOEAs NSGA-II e PAES diante de muitos objetivos. Para isso, utilizou-se a abordagem multiobjetivo MOA4PLA. Os resultados fornecem indícios de que NSGA-II supera PAES para resolver problemas com seis objetivos considerando as ALPs utilizadas. No entanto, o desempenho do NSGA-II ainda assim não é adequado, pois produz uma grande quantidade de soluções, dificultando a seleção de um projeto de ALP por parte do arquiteto. Nesse sentido, pode-se concluir que é viável a utilização das novas funções

propostas neste trabalho, porém, seria melhor utilizá-las em conjuntos variados de funções objetivos utilizando um menor número de objetivos.

Como trabalhos futuros serão realizados estudos utilizando as funções propostas com um número menor de objetivos a serem otimizados simultaneamente, além de uma análise qualitativa das soluções conduzida com arquitetos.

Agradecimentos

Os autores agradecem ao DInf/UFPR e ao Giovani Guizzo pelo apoio na execução dos experimentos. Agradecem também à CAPES pelo apoio financeiro.

Referências

Aleti, A.; Buhnova, B.; Grunske, L.; Koziolek, A. and Meedeniya I. (2013). "Software architecture optimization methods: A systematic literature review". IEEE Trans. on Software Engineering, 39(5):658–683.

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

Colanzi, T. E. e Vergilio, S. R. (2014a) "A Comparative Analysis of Two Multi-Objective Evolutionary Algorithms in Product Line Architecture Design Optimization" In: ICTAI, pp.681-688.

Colanzi, T. E. e Vergilio, S. R. (2014b) "A Feature-Driven Crossover Operator for Product Line Architecture Design Optimization". In: Computer, Software and Applications Conference, pp.43-52.

Colanzi, T. E.; Vergilio, S. R.; Gimenes, I. and Oizumi, W. N. (2014) A search-based approach for software product line design. In: Int. Software Product Line Conference, pp. 237-241, Florence, Italy.

Contieri Jr, A. C.; Correia G. G.; Colanzi, T. E.; Gimenes, I.; OliveiraJr, E.; Ferrari, S.; Masiero, P. C.; Garcia, A. F. (2011) "Extending UML Components to Develop Software Product-Line Architectures: Lessons Learned", In: European Con. on Software Architecture, pp. 130-138, Essen, Alemanha.

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

Deb, K. and Jain, H. (2012) "Handling many-objective problems using an improved NSGA-II procedure", In: IEEE Congress on Evolutionary Computation (CEC 2012), pp.1,8.

Guizzo, G.; Colanzi, T. E. and Vergilio, S. R. (2014) "Applying Design Patterns in Search-Based Product Line Architecture Design," In: SSBSE, pp 77-91, Fortaleza, Brasil.

Knowles, J. D. and Corne, D. W. (2000). "Approximating the non-dominated front using the Pareto archived evolution strategy". Evolutionary Computation, 8:149–172.

Linden, F.; Schmid, K. and Rommes, E. (2007) "Software Product Lines in Action - The Best Industrial Practice in Product Line Engineering". Springer, 2007.

Lopez-Herrejon, R. E.; Batory D. and Cook, W. (2005). "Evaluating support for features in advanced modularization technologies". In: European Conf. on Object-Oriented Programming, pp. 169–194.

OliveiraJr, E.; Gimenes, I. ; Maldonado, J. C.; Masiero, P. C.; Barroca, L. (2013). "Systematic evaluation of software product line architectures". Journal of Universal Computer Science, 19:25 – 52.

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

Sant'Anna, C. N. (2008). "On the Modularity of Aspect-Oriented Design: A Concern-Driven Measurement Approach". PhD thesis, PUC-Rio, Rio de Janeiro/RJ, Brasil.

SEI (2015). Arcade Game Maker Pedagogical Product Line. in: http://www.sei.cmu.edu/productlines/ppl/

Simons, C. L. and Parmee, I.C. (2012) "Elegant object-oriented software design via interactive, evolutionary computation". IEEE Trans. Systems, Man and Cybernetics, 42(6):1797-1805.

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

Wust, J. (2014). SDMetrics. in: http://www.sdmetrics.com/.

Utilizando Otimização por Colônia de Formigas na Seleção de Produtos para o Teste de Mutação do Diagrama de Características

Thiago do Nascimento Ferreira e Silvia Regina Vergilio

[1] Departamento de Informática – Universidade Federal do Paraná (UFPR)
Caixa Postal 19.081 – 81.531-980 – Curitiba – Paraná – Brasil

{tnferreira,silvia}@inf.ufpr.br

Abstract. *The variability mutation testing of the featue model has been found promissing results, due to its capability to reveal faults. To derive a set of test cases, that is, products, with a high mutation score, associated to a minimal cost, is a hard task, which has been solved by search based approaches. However, existing approaches use evolutionary algorithms. This work proposes an approach based on Ant Colony Optimization (ACO). A mathematical formulation is introduced, considering the mutation score and the dissimilarity between generated products, in order to reduce the number of redundant products, that is, products that kill the same set of mutants. Results comparing the approach with Genetic Algorithm, Simulated Annealing and Random Search show that ACO can find the best solutions and with the best convergence, even in feature models with higher number of products.*

Resumo. *O teste de mutação de variabilidades do diagrama de características mostra-se bastante promissor, devido a sua capacidade em revelar defeitos. Derivar um conjunto de casos de teste, ou seja, de produtos, com alto escore de mutação associado a um custo mínimo é uma tarefa complexa para a qual soluções baseadas em busca são propostas. Entretanto, as soluções existentes utilizam algoritmos evolutivos. Neste trabalho uma solução baseada em Otimização por Colônia de Formiga (ACO) é proposta. Uma modelagem é introduzida, que considera o escore de mutação e a dissimilaridade dos produtos gerados, a fim de reduzir o número de produtos redundantes, ou seja, que matam os mesmos mutantes. Resultados obtidos, comparando a abordagem com o Algoritmo Genético, Têmpera Simulada e Busca Aleatória, mostram que o ACO consegue encontrar as melhores soluções, além gerar a melhor convergência mesmo em diagramas com elevada quantidade de produtos.*

1. Introdução

A abordagem de Linha de Produto de Software (LPS) é bastante utilizada na indústria na construção de famílias de produtos, usando os conceitos de reúso de artefatos de software. Neste contexto, o Diagrama de Características (ou *Feature Model* - FM) tem uma tarefa importante pois é o artefato que representa informações sobre todos os produtos possíveis de uma LPS, em termos de características e relacionamentos que estas podem ter [Sayyad et al. 2013]. Uma característica pode ser definida como sendo uma funcionalidade ou atributo do sistema que é visível ao usuário final onde, cada característica, pode ou não ser selecionada de maneira a formar um determinado produto [Linden et al. 2007].

Com esta crescente utilização na indústria, existe também uma demanda por técnicas específicas para teste no contexto de LPS como, por exemplo, verificar se os produtos gerados estão de acordo com o desejado, ou se o produto gerado consiste em uma derivação válida das características presentes em uma LPS. Para isto, o ideal seria poder testar todos os produtos possíveis descritos pelo FM. Entretanto, considerando n características e que cada produto demora $O(m)$ para ser executado, no pior caso testar todos os produtos demandaria um tempo de $O(2^n \times m)$, o que pode ser impraticável [Ensan et al. 2012]. Diante disso, critérios de teste são aplicados como forma de selecionar e avaliar um conjunto de casos de teste (ou conjunto de produtos) de modo a aumentar a possibilidade de revelar defeitos e com isso, aumentar o nível de confiança de que a LPS (ou FM) foi especificada corretamente.

Dentre esses critérios, o teste de mutação vem se mostrando promissor devido a sua capacidade de relevar defeitos, apesar do seu custo computacional ser relativamente alto [Jia and Harman 2011]. A partir de um programa original, um conjunto de programas modificados por operadores de mutação inserem pequenos desvios sintáticos no programa (código fonte ou objeto). Este programa modificado é chamado de programa mutante. Desse modo um caso de teste é gerado e então executado com o programa original e com o mutante. Caso as saídas sejam diferentes, o mutante é dito morto.

No contexto do diagramas de características, diferentes abordagens utilizando critérios baseados no teste de mutação foram propostas [Henard et al. 2013, Ferreira et al. 2013]. Dependendo do tamanho do diagrama de característica a ser testado, uma quantidade elevada de produtos pode ser gerada para satisfazer o critério de teste de mutação. Nesta situação, técnicas para selecionar os melhores conjuntos de produtos são necessárias. Os melhores conjuntos são aqueles que contenham um número pequeno de produtos (casos de teste) e que apresentem alto escore de mutação (cobertura do critério). Para isso é necessário que produtos que matem os mesmos mutantes sejam evitados no conjunto a fim de diminuir testes redundantes e o esforço computacional.

Para este problema o uso de algoritmos de busca pode ajudar. Dentre estes, o algoritmo de Otimização por Colônia de Formiga (*Ant Colony Optimization* - ACO) [Dorigo and Stützle 2004] tem sido bastante utilizado devido a sua capacidade de gerar boas soluções em poucas iterações. Entretanto, a maioria das soluções propostas para o problema de teste de LPS a partir das variabilidades do FM, são baseadas em algoritmos evolutivos [Harman et al. 2014, Matnei Filho and Vergilio 2014]. Além disso, a maioria destes trabalhos não considera o teste de mutação.

Considerando o exposto acima este trabalho tem dois objetivos principais: i) avaliar o desempenho do ACO no teste de mutação de variabilidades de LPS, a partir do FM; e ii) propor uma modelagem matemática para selecionar um conjunto de produtos do FM, que mate a maior quantidade possível de mutantes diferentes, ou seja, produtos dissimilares, a fim de alcançar cobertura, mas com um número mínimo de casos de teste redundantes.

Para avaliar a proposta, os resultados do ACO foram comparados com os resultados gerados pela Têmpera Simulada, Algoritmo Genético e Busca Aleatória em quatro instâncias reais. A avaliação mostra que o ACO consegue gerar as melhores soluções em todas instâncias, com rápida convergência, e em menor tempo em instâncias maiores.

Este trabalho está organizado da seguinte forma: na Seção 2 são apresentados alguns aspectos da Análise de Mutantes em Diagramas de Características; na Seção 3 a abordagem baseada no algoritmo ACO é proposta; a Seção 4 descreve o estudo experimental realizado; a Seção 5 discute trabalhos relacionados; e por fim, na Seção 6 apresentam-se a conclusão e trabalhos futuros.

2. Análise de Mutantes em Diagramas de Características

O trabalho de Stephenson *et al.* [2004] foi um dos primeiros a identificar tipos de defeitos em produtos da LPS, sem, entretanto, propor uma abordagem para aplicar o critério análise de mutantes. Hernard *et al.* [2013] propuseram dois operadores de mutação para gerar produtos dissimilares, considerando defeitos no FM. Um conjunto maior de operadores foi proposto por Ferreira *et al.* [2013] , descrevendo defeitos no FM que incluem: atribuição incorreta da cardinalidade de uma característica solitária; atribuição incorreta de elementos de uma relação agrupada; atribuição incorreta da cardinalidade de uma característica agrupada; e restrições de dependência incorretas. A Figura 1 apresenta um exemplo de aplicação do operador DFL (*Decrease Solitary Feature Lower Bound*) em um diagrama de característica. Este operador decrementa o valor mínimo da cardinalidade de uma característica solitária.

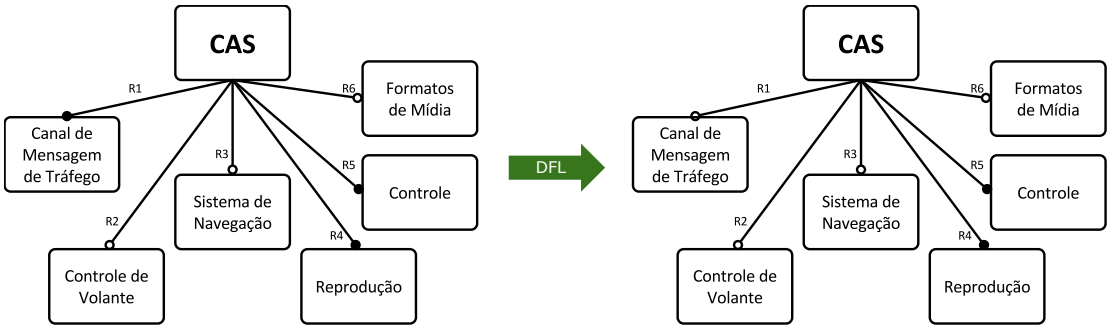


Figura 1. Exemplo de aplicação do operador DFL. Fonte: [Ferreira et al. 2013]

Na figura é possível perceber que o operador gerou um novo diagrama onde a característica "Canal de Mensagem de Tráfego", que é obrigatória no original, virou opcional no diagrama mutante.

Neste trabalho o conjunto de operadores de Ferreira *et al.* [2013] é utilizado. Além de ser um conjunto mais completo, que considera a maioria dos defeitos presentes no FM, este conjunto pode ser utilizado em um processo de teste automatizado pela ferramenta chamada FMTS (*Feature Mutation Test Suite*). FMTS realiza a geração de mutantes para um FM e calcula o escore de mutação de um dado conjunto de produtos gerados. Assim como no teste de programas tradicionais, um FM mutante é morto quando se comporta diferentemente do FM em teste. Para tanto, um FM mutante deve validar um produto que o FM não valida, ou, ao contrário, deve considerar como inválido, um produto que é válido de acordo com o FM original. O escore de mutação $MS(FM, P')$ é calculado da seguinte forma:

$$MS(FM, P') = \frac{MM(P')}{MG(FM) - ME(FM)} \quad (1)$$

onde P' é o conjunto de produtos (ou conjunto de casos de teste), FM é o diagrama de características em teste, $MM(P')$ é o número de mutantes mortos por P', $MG(FM)$ é número total de mutantes gerados e $ME(FM)$ é o número de mutantes equivalentes a FM.

Em algumas situações não é possível matar todos os mutantes. Alguns mutantes podem ser semanticamente equivalentes ao original, ou seja, apesar de diferentes sintaticamente derivam o mesmo conjunto de produtos. Então não existe caso de teste capaz de diferenciá-los. Estes mutantes precisam ser descartados no cálculo do escore.

3. Abordagem Proposta

Nesta seção é apresentada a modelagem matemática e o algoritmo ACO utilizado.

3.1. Definição do Problema

Sabendo que $P = \{p_1, p_2, p_3, ..., p_n\}$ é o conjunto de produtos disponíveis para seleção e $M = \{m_1, m_2, m_3, ..., m_m\}$ o conjunto de mutantes que devem ser mortos, a matriz MC de tamanho $m \times n$ especifica quais produtos matam quais mutantes. Se $MC_{ij} = 1$ então o mutante i é morto pelo produto j. Caso contrário, $MC_{ij} = 0$.

Para maximizar a quantidade de produtos que matam mutantes diferentes, utiliza-se uma medida de dissimilaridade entre dois produtos p_j e p_k definida por:

$$dissimilaridade(p_j, p_k) = 1 - jaccard(p_j, p_k) \tag{2}$$

onde $0 \leq jaccard(p_j, p_k) \leq 1$ é a medida de similaridade de Jaccard [Jaccard 1908] onde 0 representa que os produtos p_j e p_k matam mutantes diferentes e 1 os produtos p_j e p_k matam os mesmos mutantes. A função pode ser descrita como:

$$jaccard(p_j, p_k) = \frac{|D(p_j) \cap D(p_k)|}{|D(p_j) \cup D(p_k)|} \tag{3}$$

onde $D(p_j)$ retorna o conjunto dos mutantes mortos por um produto p_j. Se $D(p_j)$ e $D(p_k)$ são ambos vazios, $jaccard(p_j, p_k) = 1$. A função $D(p_j)$ é definida como:

$$D(p_j) = \{i \mid MC_{ij} = 1, \forall m_i \in M\} \tag{4}$$

Sabendo que $P' \subset P$ é um subconjunto de produtos selecionados, a dissimilaridade média de P' é dada por:

$$DM(P') = \begin{cases} \frac{\sum_{p_j \in P'} \sum_{p_k \in P'} dissimilaridade(p_j, p_k)}{|P'| * (|P'|-1)}, \forall j \neq k & \text{Se } |P'| > 1 \\ 1 & \text{Caso contrário} \end{cases} \tag{5}$$

onde $DM(P') = 1$ caso os produtos selecionados matem todos mutantes diferentes e $DM(P') = 0$ caso os produtos selecionados em P' matem os mesmos mutantes.

Com isso, a formulação do problema pode ser definida como:

$$\text{maximizar} \quad w_1 * MS(FM, P') + w_2 * DM(P') \tag{6}$$

onde $MS(FM, P')$ é escore de mutação do conjunto P', $DM(P')$ a dissimilaridade do conjunto P', w_1 e w_2 representam os pesos das funções MS e DM respectivamente.

3.2. Algoritmo de Otimização por Colônia de Formiga

O algoritmo de Otimização por Colônica de Formigas (*Ant colony optimization* - ACO) é uma metaheurística desenvolvida para resolver problemas de otimização combinatória e foi proposto por Dorigo e Stützle [2004]. Este algoritmo simula o comportamento das formigas na busca pelo menor caminho do seu ninho até a fonte de comida. Esta tarefa é guiada por uma substância chamada feromônio que é responsável por fazer a comunicação entre as formigas. Esta substância é deixada em uma trilha e quanto maior a sua concentração, maior será a atratividade daquela trilha por outra formiga.

O *Ant Colony System* (ACS) é um algoritmo de otimização baseado no comportamento das formigas [Dorigo and Stützle 2004]. Neste algoritmo, o problema é codificado como um grafo completo e direcionado $G = (V, A)$ onde, para esse trabalho, cada vértice representa um produto possível de ser selecionado. No algoritmo, cada formiga constrói a sua solução percorrendo o grafo G visitando cada vértice somente uma vez. Um valor τ_{ij} é associado a cada aresta de G e representa a quantidade de feromônio depositado entre o vértice i e o vértice j. Inicialmente, o valor do feromônio em todas as arestas é definido para τ_0.

Quando uma formiga k está no vértice i, o vértice j seguinte é selecionado estocasticamente dentro de um conjunto de vértices disponíveis a partir de i. Esse conjunto é chamado de $vis_k(i)$ e é definido como:

$$vis_k(i) = \{j \mid dissimilaridade(p_i, p_j) > DM(P)\} \tag{7}$$

O processo de construção da solução pela formiga é feito segundo uma regra de proporção pseudoaleatória. Baseado em uma variável q aleatória uniformemente distribuída entre $[0, 1]$, se $q \leq q_0$, o vértice j é selecionado segundo:

$$j = argmax_{l \in vis_k(i)} \tau_{il} * \eta_{il}^{\beta} \tag{8}$$

onde β representa a importância da informação heurística entre o vértice i e o vértice j. A informação heurística é determinada por :

$$\eta_{ij} = \mu * \left(dissimilaridade(p_i, p_j) + \frac{|D(p_j)|}{MG(FM)} \right) \tag{9}$$

onde $MG(FM)$ é a quantidade de mutantes gerados e μ uma constante. Se $q > q_0$ a formiga k escolhe o vértice j segundo uma probabilidade p_{ij}^k dada por:

$$p_{ij}^k = \begin{cases} \frac{\tau_{ij}^{\alpha} * \eta_{ij}^{\beta}}{\sum_{h \in vis_k(i)} \tau_{ih}^{\alpha} * \eta_{ih}^{\beta}} & \text{Se } j \in vis_k(i) \\ 0 & \text{Caso contrário} \end{cases} \tag{10}$$

onde os parâmetros α e β representam respectivamente a importância do feromônio e a importância da informação heurística.

A trilha de feromônio é atualizada localmente e globalmente. A atualização local ocorre sempre que uma formiga sai do vértice i e vai para o vértice j. Essa atualização é definida como:

$$\tau_{ij} = (1 - \varphi) * \tau_{ij} + \varphi * \tau_0 \tag{11}$$

onde φ representa o fator de decaimento do feromônio e τ_0 a quantidade de feromônio inicial. Para este trabalho, $\tau_0 = 0,5$.

No final do processo, somente a formiga que gerou a melhor solução atualiza as arestas que pertencem a solução. A equação de atualização é dada por :

$$\tau_{ij} = (1 - \rho) * \tau_{ij} + \rho * \Delta\tau_{ij} \tag{12}$$

onde $\Delta\tau_{ij} = \frac{f_{best}}{2}$ e f_{best} é o valor da melhor solução encontrada.

4. Estudo Experimental

Esta seção apresenta os experimentos realizados a fim de avaliar a modelagem e o algoritmo ACO proposto.

4.1. Instâncias e Algoritmos Utilizadas

Para avaliar a proposta, um estudo experimental foi realizado com quatro LPS cuja as matrizes MC estão disponíveis em [Matnei Filho 2015]. A CAS (Car Audio System) é uma linha de produto de software para sistemas de som automotivo. JAMES é uma linha de produto de software para sistemas web colaborativos. Weather Station representa uma linha de produto de software para sistemas meteorológicos e por fim, a EShop é uma linha de produto de software para E-commerce.

Utilizando a ferramenta FMTS [Ferreira et al. 2013], diagramas mutantes e produtos foram gerados para cada LPS. Do conjunto de mutantes gerados, foram descartados todos os mutantes equivalentes e anômalos (diagramas mal formados). A Tabela 1 apresenta informações a cerca das instâncias utilizadas neste estudo[1].

Tabela 1. Informações das LPS utilizadas

Instâncias	Quantidade de Produtos	Mutantes Ativos	Características
JAMES	68	106	14
CAS	450	227	21
Weather Station	504	357	22
EShop	1152	394	22

No experimento, o ACO proposto foi comparado com o Algoritmo Genético (GA), Têmpera Simulada (SA) e Busca Aleatória (RS) desenvolvidos para este trabalho.

[1]Detalhes em http://www.inf.ufpr.br/tnferreira/supporting-pages/aco-for-feature-model/

4.2. Questões de Pesquisa

Buscando guiar a avaliação do experimento, as seguintes questões de pesquisa foram propostas:

RQ1: A abordagem utilizando ACO consegue gerar as melhores soluções em valores *fitness* e menor quantidade de produtos dentre os algoritmos avaliados?
RQ2: Qual dos algoritmos converge mais rapidamente?

Para responder RQ1, os resultados gerados durante 30 execuções pelas melhores configurações dos algoritmos foram comparados utilizando média, desvio padrão e testes estatísticos. No caso de RQ2, a cada geração do algoritmo a melhor solução até o momento foi guardada repetindo este processo durante 30 execuções.

4.3. Configuração dos Experimentos

Antes de avaliar os resultados encontrados pelos algoritmos, uma fase de *tunning* foi executada buscando encontrar a configuração que gerava as melhores soluções. Nesta fase, 688 combinações de parâmetros foram realizadas, sendo cada combinação executada por 30 vezes. Testes usando Kruskal Wallis com 95% de confiança foram realizados para verificar se existia alguma diferença estatística entre as melhores configurações de cada algoritmo. O teste mostrou que não existia diferença. Assim, a mesma configuração foi fixada para todas as instâncias. São elas:

- ACO: Formigas: 50, Avaliações: 50.000, α e β: 1 e $q_0 : 1$
- GA: População: 100, Avaliações: 50.000, Taxa de Mutação: 0.5%, Taxa de Cruzamento: 90%, Mutação: Bit Flip, Crossover: Uniforme e Seleção: Torneio Binário;
- SA: Temperatura Inicial: 1000, Decaimento: 0.995 e Avaliações: 50.000;
- RS: Avaliações: 50.000.

4.4. Resultados e Discussões

Esta subseção apresenta os resultados encontrados nos experimentos realizados.

4.4.1. Resultados para RQ1

A Tabela 2 apresenta os resultados encontrados de média e desvio padrão de *fitness* em 30 execuções de cada algoritmo em cada instância.

Tabela 2. Resultados encontrados de média e desvio padrão para os algoritmos

	Métrica	JAMES	CAS	EShop	Weather Station
ACO	fitness	**1.89482±0.00311**	**1.90266±0.00196**	**1.86929±0.00260**	**1.87761±0.00360**
	tempo (s)	18.067±2.074	6.671±0.738	28.907±1.929	7.883±0.738
GA	fitness	1.89195±0.00572	1.87675±0.00293	1.82181±0.00078	1.85491±0.00063
	tempo (s)	0.925±0.195	16.197±0.701	94.379±9.977	19.858±0.613
SA	fitness	1.88274±0.01060	1.89553±0.00379	1.86345±0.00293	1.87591±0.00248
	tempo (s)	1.229±0.239	16.314±1.227	87.676±4.425	16.060±1.141
RS	fitness	1.82096±0.00240	1.79710±0.00088	1.77814±0.00072	1.83324±0.00037
	tempo (s)	4.047±0.572	134.248±20.327	625.7005±41.999	282.503±15.366

Em todas as instâncias, o algoritmo ACO conseguiu encontrar, em média, as soluções que possuem os melhores valores de *fitness*. Além disso, o teste estatístico de

Kruskall Wallis mostrou que além de obter as melhores médias, com 95% de confiança os resultados gerados pelo ACO são estatisticamente diferente dos outros algoritmos. No tempo de execução, o ACO conseguiu encontrar nas instâncias EShop e Weather Station as melhores soluções gastando o menor tempo. Vale a pena ressaltar que estas LPS são as que possuem, respectivamente, o maior e segundo maior número de produtos.

Tabela 3. Quantidade média e desvio padrão de produtos selecionados

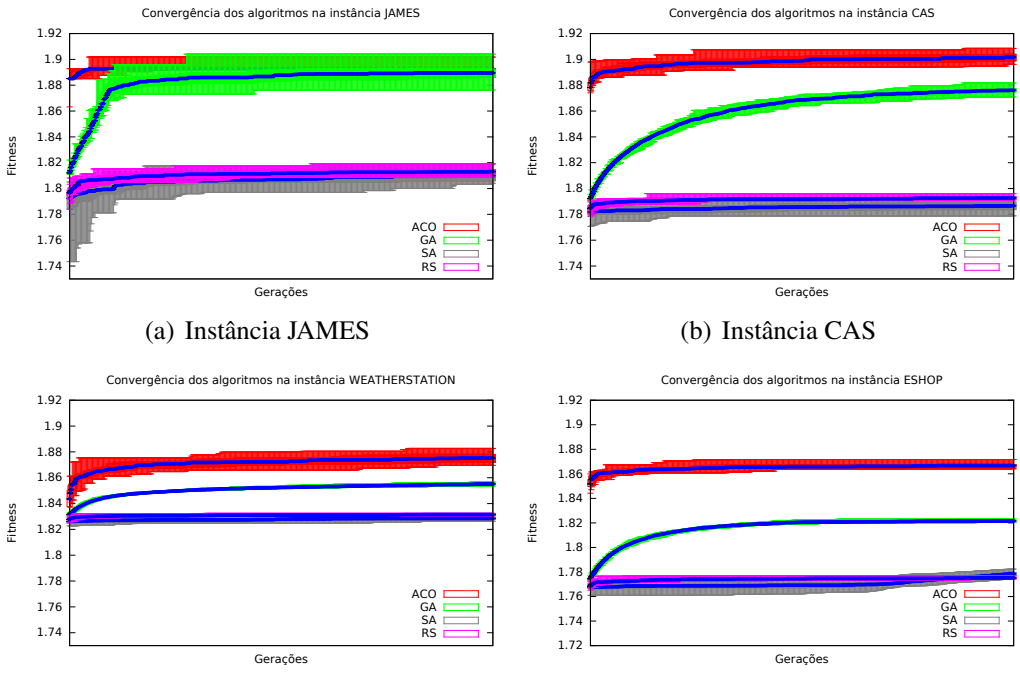
Instância	ACO	GA	SA	RS	Diferença
JAMES	**6.03 ± 0.18**	6.86 ± 0.86	7.06 ± 0.90	23.36 ± 3.15	-12.10%
CAS	**11.16 ± 0.79**	26.4 ± 2.96	13.83 ± 1.53	213.06 ± 11.62	-19.31%
ESHOP	**11.33 ± 1.24**	270.70 ± 15.69	14.80 ± 1.54	573.73 ± 16.86	-23.45%
WEATHERSTATION	**11.23 ± 0.72**	57.86 ± 4.78	17.20 ± 1.56	246.6 ± 9.59	-34.71%

A Tabela 3 apresenta a quantidade de produtos selecionados por cada algoritmo em cada instância. O ACO conseguiu encontrar, em média, 12% menos produtos na instância JAMES, 19% menos na instância CAS, 23% menos na instância EShop e 34% menos na instância Weather Station totalizando uma redução média de 22% na quantidade de produtos selecionados comparado com os outros algoritmos.

Por fim e respondendo da RQ1, o algoritmo ACO conseguiu gerar, em média, as melhores soluções com diferença estatística para as instâncias analisadas.

4.4.2. Resultados para RQ2

A Figura 2 apresenta a convergência dos algoritmos durante as primeiras 1.000 avaliações em cada instância, onde os valores de cada avaliação estão representados em um gráfico de caixa que representa o valor em 30 execuções naquela avaliação.



(a) Instância JAMES

(b) Instância CAS

(c) Instância Weather Station

(d) Instância EShop

Figura 2. Convergência da melhor solução durante a evolução dos algoritmos

Analisando a figura e respondendo a RQ2, para todas as instâncias, o algoritmo ACO conseguiu convergir mais rapidamente do que os outros algoritmos além de conseguir gerar a melhor solução logo nas primeiras gerações.

5. Trabalhos Relacionados

Segundo Harman *et al*. [2014] houve um crescente número de publicações que utilizam algoritmos de busca na área de LPS principalmente na área de evolução, customização e teste de uma LPS. Os trabalhos mais relacionados a esta proposta são os que visam a satisfazer o teste de mutação, e/ou que utilizam o algoritmo ACO.

Considerando a aplicação do ACO no contexto de LPS, Wang e Pang [2014] apresentam uma abordagem para seleção de características. O trabalho mostra que a abordagem proposta gera resultados cerca de 10% melhores do que o Algoritmo Genético para seleção de características (GAFES). Entretanto, este trabalho não considera critérios de teste.

Trabalhos que consideram o teste de mutação utilizam algoritmos evolutivos. Henard *et al.* [2014] utilizam um Algoritmo Evolucionário (1+1), que tem como objetivo gerar um conjunto de configurações para teste baseado no escore de mutação de cada configuração. Os operadores propostos em [Henard et al. 2013] foram utilizados.

O trabalho desenvolvido por Matnei Filho [2015] propõe o uso de algoritmos multiobjetivos buscando selecionar um conjunto de produtos que maximize o escore de mutação e a cobertura de pares de características (*parwise testing*), minimizando o tamanho do conjunto. Resultados avaliando os algoritmos NSGA-II, SPEA2 e IBAE mostram que todos os algoritmos obtiveram um bom desempenho com destaque para o SPEA2 em alguns casos que obteve as melhores soluções.

Nenhum dos trabalhos relacionados faz uso do ACO como técnica de busca para selecionar os produtos que devem ser utilizados no teste de mutação de variabilidades de uma LPS. Além disso, são poucos os trabalhos que consideram o teste de mutação [Henard et al. 2014, Matnei Filho 2015] e estes trabalhos não consideram uma medida de similaridade entre os produtos, que visa a diminuir o número de produtos redundantes que matam os mesmos mutantes. Como mostrado nos experimentos o ACO é uma boa alternativa principalmente para LPS com um número maior de produtos.

6. Conclusões e Trabalhos Futuros

Este trabalho apresentou uma modelagem matemática para resolver o problema de selecionar os melhores subconjuntos de produtos para satisfazer o teste de mutação aplicado ao diagrama de características, além de propor um algoritmo baseado no comportamento de colônia de formigas para resolver esse problema.

Os resultados encontrados no estudo experimental mostram que o algoritmo ACO conseguiu gerar, em média, as melhores soluções para o problema, selecionando a menor quantidade de produtos além de obter a melhor convergência.

Como trabalhos futuros, pretende-se aplicar a abordagem proposta a outras instâncias e LPS maiores, realizar testes com outros algoritmos de otimização, aplicar outras métricas de coberturas relacionadas por exemplo ao teste *pairwise* e, propor para o problema uma abordagem multiobjetivo baseada no ACO.

Referências

Dorigo, M. and Stützle, T. (2004). *Ant Colony Optimization*. Bradford Company, Scituate, MA, USA.

Ensan, F., Bagheri, E., and Gašević, D. (2012). Evolutionary search-based test generation for software product line feature models. In *Advanced Information Systems Engineering*, pages 613–628. Springer.

Ferreira, J. M., Vergilio, S. R., and Quinaia., M. (2013). A mutation approach to feature testing of software product lines. In *International Conference on Software Engineering and Knowlodge Engineering (SEKE)*.

Harman, M., Jia, Y., Krinke, J., Langdon, W., Petke, J., and Zhang, Y. (2014). Search based software engineering for software product line engineering: a survey and directions for future work. In *Proceedings of the 18th International Software Product Line Conference-Volume 1*, pages 5–18.

Henard, C., Papadakis, M., and Le Traon, Y. (2014). Mutation-based generation of software product line test configurations. In *Search-Based Software Engineering*, pages 92–106. Springer.

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

Jaccard, P. (1908). Nouvelles recherches sur la distribution florale. *Bulletin de la Societe Vaudoise des Sciences Naturelles*, 44(163):223-70.

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

Linden, F. J., Schmid, K., and Rommes, E. (2007). *Software product lines in action: the best industrial practice in product line engineering*. Springer Science & Business Media.

Matnei Filho, R. (2015). Gerando dados para o teste de mutação de linha de produto de software com algoritmos de otimização multiobjetivo. Master's thesis, Universidade Federal do Paraná - UFPR.

Matnei Filho, R. and Vergilio, S. (2014). Configuração baseada em busca de linha de produto de software: Resultados de uma mapeamento sistemático. In *V Workshop de Engenharia de Software Baseada em Busca (WESB)*.

Sayyad, A. S., Ingram, J., Menzies, T., and Ammar, H. (2013). 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. IEEE.

Stephenson, Z., Zhan, Y., Clark, J., and McDermid, J. (2004). Test data generation for product lines–a mutation testing approach. In *International Workshop on Software Product Line Testing*, volume 2004, page 13. Citeseer.

Wang, Y.-l. and Pang, J.-w. (2014). Ant colony optimization for feature selection in software product lines. *Journal of Shanghai Jiaotong University (Science)*, 19:50–58.