Uma Abordagem Integrada, Interativa e Multi-Objetiva para os Problemas de Seleção, Priorização e Alocação de Casos de Teste

Camila Loiola Brito Maia[1], Jerffeson Teixeira de Souza[2]

[1]Serviço Federal de Processamento de Dados (SERPRO)
Av. Pontes Vieira, 832 – Fortaleza, CE - Brasil

[2]Universidade Estadual do Ceará (UECE)
Fortaleza, 60740-903, Ceará, Brasil

camila.maia@gmail.com, jeff@larces.uece.br

Resumo. *Os problemas de seleção, priorização e alocação de casos de teste podem ser considerados como difíceis, devido ao grande número de soluções possíveis que devem ser consideradas em sua resolução e os diversos fatores que podem influenciar na busca dessas soluções. Existem vários trabalhos que utilizam técnicas de otimização na busca por soluções para problemas difíceis da engenharia de software, na recente área de pesquisa conhecida como Search-Based Software Engineering. Dentro desse contexto, este trabalho propõe uma abordagem integrada, interativa e multi-objetiva para a seleção, priorização e alocação de casos de teste. Dois experimentos são detalhados, mostrando a efetividade das técnicas de otimização nos três problemas.*

Abstract. *The test cases selection, prioritization and allocation problems can be considered difficult because of the large number of possible solutions that should be considered in solving these problems and the many factors that can influence the search for these solutions. There are several studies that use optimization techniques in the search for solutions to difficult problems of software engineering at the recent area of research known as Search-Based Software Engineering. In this context, this work proposes an integrated, interactive and multi-objective approach to the test cases selection, prioritization and allocation problems. Two experiments are detailed, showing the effectiveness of optimization techniques in the three problems.*

1. Introdução

Diversos problemas relacionados à Engenharia de Software podem ser considerados como complexos, como, por exemplo, alocação de tarefas, escolha de requisitos que devem ser implementados no próximo release e seleção de casos de teste. Esses problemas podem apresentar um número alto de possíveis soluções e alta complexidade, tornando-os grandes candidatos a serem resolvidos por métodos automáticos ou semi-automáticos.

Uma área relativamente nova, unindo Engenharia de Software e Pesquisa Operacional, chamada *Search-Based Software Engineering* (HARMAN e JONES, 2001), ou SBSE, tem obtido grande interesse recente da comunidade acadêmica. A idéia principal é aplicar técnicas de otimização em problemas da Engenharia de Software que podem ser formulados matematicamente. Para isso, uma ou mais funções objetivo são

definidas, bem como as restrições do problema, e as técnicas de busca são aplicadas a fim de encontrar soluções próximas da solução ótima. No caso de haver mais de uma função objetivo, para selecionar as soluções utilizamos o conceito de Frente de Pareto, que representa o conjunto de soluções eficientes para o problema. Essas soluções pertencentes à Frente de Pareto são igualmente boas porque não existe relação de dominância entre elas.

O termo *Search-Based Software Engineering* foi criado em 2001 por Harman e Jones (HARMAN e JONES, 2001), e desde então uma grande quantidade de trabalhos têm sido desenvolvidos nesta área. Alguns exemplos de áreas da Engenharia de Software que já possuem aplicação destas técnicas são: engenharia de requisitos (BAGNALL et al., 2001), análise e projeto (LI et al., 2010), codificação (KUPERBERG e OMRI, 2009), teste de software (LI et al., 2007), gerência de projetos (DI PENTA et al., 2007) e métricas (ANTONIOL et al., 2009).

Segundo Bastos (BASTOS et al., 2007), o principal objetivo do processo de testes é encontrar o maior número de erros possível, para que possam ser corrigidos e o cliente possa receber uma versão confiável do sistema. Um dos fatores determinantes para que isso ocorra é um bom planejamento da execução dos testes, de modo a garantir que os testes mais significativos sejam executados. O planejamento da execução dos testes consiste em selecionar e/ou priorizar os casos de teste a serem executados, e alocá-los aos testadores disponíveis para sua execução.

Nos trabalhos (COLARES et al., 2009), (WALCOTT et al., 2006) e (LI et al., 2007) foi demonstrado que as técnicas de busca superam outras técnicas já utilizadas para resolução de problemas da engenharia de software modelados matematicamente. O trabalho (SOUZA et al., 2010) mostra que as técnicas de busca superam também a resposta humana para alguns problemas da engenharia de software, comprovando a competitividade dessas técnicas naqueles cenários.

A proposta deste trabalho é formular matematicamente, de forma multi-objetiva, integrada e iterativa, os problemas de seleção, priorização e alocação de casos de teste e aplicar técnicas baseadas em busca, mais especificamente metaheurísticas, na resolução dos mesmos. A formulação será multi-objetiva porque leva em consideração diversos fatores que podem ser otimizados, nos três problemas. É também integrada porque os problemas são modelados de forma a serem executados sequencialmente, ou seja, o resultado de um é entrada para a execução do processo seguinte. Além disso, a abordagem é iterativa, pois quando cada um desses processos exibe um conjunto de soluções para o usuário, é este usuário que escolherá qual das soluções apresentadas será a entrada para o processo seguinte.

2. Trabalhos Relacionados

Nesta seção serão relatados alguns trabalhos relacionados à aplicação de técnicas de otimização aos problemas de seleção e priorização de casos de teste. Existem diversos trabalhos que tratam de alocação de tarefas utilizando-se dessas técnicas, porém não tratam especificamente de alocação de casos de teste para testadores. Há alguns trabalhos que tratam de alocação de recursos para a execução de testes, porém esses recursos englobam outros fatores além de mão de obra, como tempo de CPU e número de casos de teste.

O trabalho (YOO e HARMAN, 2007) foi o primeiro a tratar o problema de seleção de casos de teste como um problema multi-objetivo. Os autores propõem duas abordagens: a primeira considera cobertura de código e custo como funções objetivo, e a segunda considera cobertura de código, custo e histórico de detecção de falhas dos casos de teste como funções objetivo. Foram utilizados, para efeitos de comparação, programas pequenos e grandes. Para os programas pequenos, o algoritmo NSGA-II obteve o melhor desempenho, seguido pelo algoritmo vNSGA-II. Para os programas maiores, o algoritmo guloso teve o melhor desempenho.

Em (MAIA et al., 2009), foi apresentada uma nova abordagem multi-objetivo para o problema de seleção de casos de teste. As funções objetivo otimizadas neste trabalho são o tempo total de execução, o risco dos casos de teste e a importância dos casos de testes. Como restrições, são levados em consideração o tempo de execução dos casos de teste selecionados e o tempo disponível para testes de cada testador. O algoritmo NSGA-II foi comparado apenas com um algoritmo randômico, obtendo melhor desempenho.

Os autores de (WALCOTT et al., 2006) propuseram uma técnica de priorização de casos de teste baseada em algoritmo genético que reordena a suíte de teste considerando dois objetivos: restrições de tempo para testes e cobertura de código. A função objetivo implementa a técnica *Average Percentage Block Coverage* (APBC). A técnica proposta foi comparada com as seguintes técnicas: ordem inicial, onde os casos de teste são ordenados de acordo com a ordem em que foram escritos, ordem reversa, onde os casos de teste são ordenados de forma descendente de acordo com a ordem em que foram escritos, e fault-aware, onde os casos de teste são ordenados de acordo com a função APBC, até que o tempo limite para execução seja atingido. A técnica proposta pelos autores obteve melhor desempenho que as outras técnicas para um dos estudos de caso. Para o outro estudo de caso, este fato não ocorreu porque os casos de teste eram mais intercambiáveis (trocados facilmente por outros). Segundo os autores, é mais fácil para os métodos de priorização randômicos ter valores de APFD maiores à medida que mais casos de teste da suíte original possam ser executados.

Em (LI et al., 2007), os autores comparam cinco algoritmos para o problema da priorização de casos de teste: Algoritmo Guloso, Additional Greedy, 2-Optimal, Hill Climbing, e Algoritmo Genético. A análise foi realizada separadamente para programas pequenos e grandes. Como função objetivo, foram consideradas as seguintes métricas de cobertura, separadamente (três abordagens mono-objetivo): *Average Percentage Block Coverage* (APBC), *Average Percentage Decision Coverage* (APDC) e *Average Percentage Statement Coverage* (APSC). Para os programas pequenos, o algoritmo genético foi melhor que os demais, e o Algoritmo Guloso obteve o pior desempenho. O algoritmo *Additional Greedy* teve o segundo melhor desempenho, e os autores mostraram que a diferença entre ele e o Algoritmo Guloso, que foi o melhor, não foi significante. Para os programas grandes, os dois melhores algoritmos foram *Additional Greedy* e *2-Optimal*, porém não houve diferença significativa entre eles.

Uma nova abordagem para o problema de alocação de tarefas é apresentado em (DI PENTA et al., 2009), baseada em dependências, fragmentação, conflitos e especialização das tarefas. Os algoritmos implementados para a avaliação da solução foram: Têmpera Simulada, Algoritmo Genético mono e multi-objetivo e Hill Climbing. Nos experimentos foram utilizados dados de projetos reais. Os autores buscavam

minimizar o tempo de conclusão das tarefas e também sua fragmentação, respeitando as dependências (executadas pelo mesmo time) e experiência das pessoas envolvidas. Os algoritmos Têmpera Simulada e Algoritmo Genético obtiveram melhor desempenho que os demais.

Em geral, para os problemas de seleção e priorização de casos de teste, a maioria dos trabalhos utiliza a cobertura de código como função objetivo, ou seja, é necessário conhecer todo o código do sistema, e seu relacionamento com os casos de teste. Nem sempre isto é possível, pois os testadores podem não conhecer a linguagem do software que está sendo desenvolvido (e isto não é um requisito para testar) ou uma fábrica de teste pode ter sido contratada, simplesmente para testar em modo caixa-preta. A abordagem proposta nesse artigo considera funções objetivo que dependem de dados dos requisitos e casos de teste, não dependendo do código da aplicação. Considera também valores como quantidade de execuções dos casos de teste por testadores. Todos os valores considerados nesta abordagem são facilmente coletados em ferramentas automáticas existentes no mercado. Porém, para os experimentos citados neste trabalho, foram considerados dados gerados randomicamente.

3. Abordagem Integrada dos Problemas

Nesta seção é apresentada a formulação para os três seguintes problemas de teste de software, de forma integrada: seleção, priorização e alocação de casos de teste.

3.1. Visão Geral da Abordagem Integrada

A abordagem proposta consiste em selecionar, priorizar e alocar casos de teste, de forma integrada, interativa e multi-objetiva.

A solução é composta por três fases distintas, onde a saída da primeira fase (seleção de casos de teste) é entrada para a segunda fase (priorização de casos de teste), e a saída desta é entrada para a terceira fase (alocação dos casos de teste). A Figura 1 detalha a abordagem proposta.

A saída da primeira fase é um conjunto de soluções, onde cada solução possui um subconjunto de casos de teste, selecionados para a execução. O usuário seleciona uma das soluções sugeridas e então o processo de ordenação dos casos de teste, a fase 2, é iniciado. A saída da segunda fase é um conjunto de soluções, onde cada solução contém os mesmos casos de teste selecionados na fase 1, porém ordenados para execução. Novamente, o usuário seleciona uma das soluções sugeridas (uma ordenação de casos de teste), e a fase 3 inicia. A saída desta última fase é um conjunto de soluções, onde cada solução contém a alocação dos casos de teste que já foram selecionados e priorizados, ou seja, conterá a informação de quem testará o que.

Na fase 1, para selecionar os casos de teste, os seguintes objetivos são levados em consideração: a importância dos requisitos do sistema para o cliente, o tempo máximo permitido para a execução dos testes e a precedência dos casos de teste. A seleção dos requisitos é baseada na visão do cliente, visto que o valor de importância dado a cada requisito é atribuído pelo cliente.

A priorização de casos de teste, fase 2, consiste em ordenar os casos de teste para execução, e esta ordenação é baseada na volatilidade e complexidade dos requisitos. Há uma restrição de precedência dos casos de teste: caso exista um caso de

teste x que possui um caso de teste precedente y, este precedente y deve ser executado antes de x, logo y deverá estar posicionado antes de x na solução ordenada.

Por último, a fase de alocação de casos de teste, fase 3, consiste em distribuir os casos de teste aos testadores disponíveis, e leva em consideração a habilidade dos testadores, a preferência dos testadores e o histórico de execução dos testes, além da restrição de precedência, que afirma que um caso de teste y que é precedente do caso de teste x deve ser executado pelo mesmo testador que executar o caso de teste x.

A saída final é um conjunto de casos de teste selecionados, priorizados e alocados para os testadores.

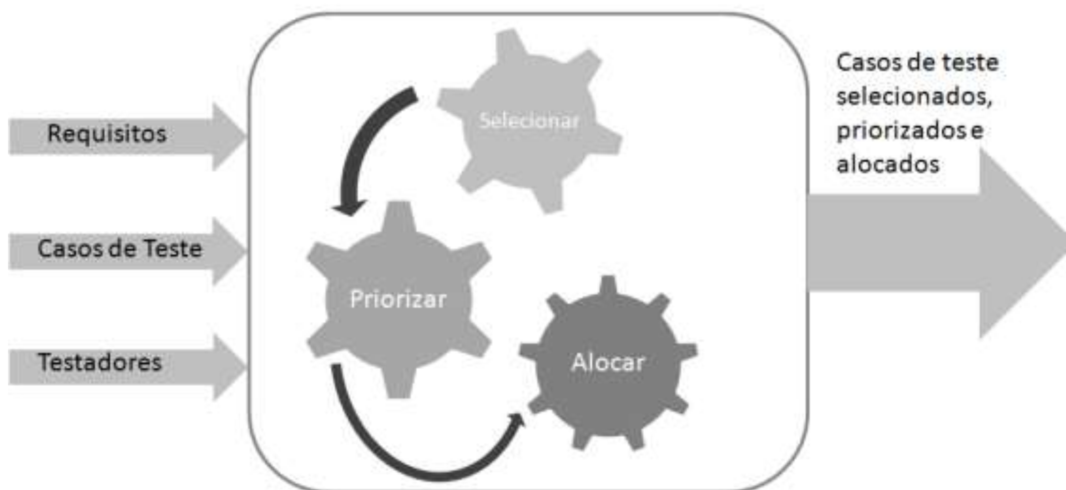

Figura 1. Visão Geral da Solução Proposta (MAIA, 2011)

3.2. Formalização dos Problemas

A seguir são apresentadas algumas definições formais que permitirão a formulação matemática da abordagem proposta.

Seja R o conjunto de requisitos de um sistema. O conjunto R contém N requisitos, ou seja, $R = \{r_i \mid i = 1, \dots, N\}$, e cada requisito possui os seguintes atributos: $importance_i$, a importância associada pelo cliente para cada requisito, $volatility_i$, que representa o quanto o requisito já foi alterado, e $complexity_i$, que representa a complexidade do requisito. Nesta abordagem, pode-se atribuir valores de 1 a 100 para os três atributos.

O conjunto de casos de teste C possui todos os casos de teste do sistema. C possui M casos de teste, ou seja, $C = \{\, c_j \,|\, j = 1, \dots, M\}$. Cada caso de teste possui os seguintes atributos: $requiredExperience_j$, que representa o nível de experiência mínimo exigido do testador (valores entre 1 e 100), $executionTime_j$, seu tempo previsto para execução, $precedence_j$, que é o caso de teste precedente, e $coverage_j$, que é o conjunto de requisitos cobertos por ele. O tempo previsto para execução, nesta abordagem, seria uma média do tempo de execução até o momento ou, caso seja um novo caso de teste, uma estimativa de tempo para sua execução.

Os testadores são representados pelo conjunto T, que possui P elementos, ou seja, $T = \{t_w \mid w = 1, \ldots, P\}$. Cada testador possui $experience_w$, que representa o nível de experiência do testador, e pode assumir valores de 1 a 100 nesta abordagem. A idéia é utilizar testadores mais experientes na execução de casos de teste mais complexos.

A partir do relacionamento dos casos de teste com os testadores podemos citar os seguintes atributos: $preference_{w,j}$, que representa a preferência do testador w pelo caso de teste j, e $numberOfExecutions_{w,j}$, que indica quantas vezes um testador w executou o caso de teste j.

As suítes de teste são representadas de duas formas nesta abordagem. A primeira é um vetor binário chamado STC, de tamanho M, onde cada posição representa um caso de teste do sistema. Se o valor para uma determinada posição em STC for 1, o caso de teste correspondente àquela posição está selecionado. Caso o valor seja 0, o caso de teste correspondente não está selecionado. O vetor STC representa a solução para a fase 1 da abordagem proposta (seleção de casos de teste).

A segunda forma de representação de uma suíte de teste é o vetor de inteiros PTC, cujo tamanho é o número de casos de teste selecionados na primeira fase da abordagem. Os números inteiros contidos em suas células representam o número dos casos de teste que serão executados, na respectiva ordem. O caso de teste contido na célula 1 do vetor PTC será executado primeiro, o caso de teste contido na célula 2 deve ser executado após o primeiro, e assim por diante. Esta é a representação da solução da fase 2 (priorização).

As suítes de teste possuem as seguintes informações: $suiteCoverage_{STC}$, que é o conjunto de requisitos cobertos por STC, $suiteImportance_{STC}$, a importância da suíte na visão do cliente, $suiteVolatility_{PTC}$, a volatilidade ponderada da suíte, $suiteComplexity_{PTC}$, a complexidade ponderada da suíte, e $testCasePosition_{j,PTC}$, que representa a posição do caso de teste j na suíte PTC.

As alocações dos casos de teste para os testadores são representadas pelo vetor de inteiros ATC. Cada célula de ATC possui o número do testador que executará o caso de teste que está na posição correspondente no vetor PTC. O vetor ATC possui as seguintes informações: $numberOfExecutions_{ATC}$, que representa a soma da quantidade de vezes que um caso de teste foi testador pelo testador que está alocado a ele (para todos os casos de teste), $preference_{ATC}$, que é o mesmo para a preferência do testador pelo caso de teste alocado para ele, e $experience_{ATC}$, que é a soma das experiências dos testadores os casos de teste alocados para o mesmo. Neste último caso, uma penalidade é aplicada se um caso de teste for alocado a um testador que não possua a experiência mínima para executá-lo. Mais detalhes da representação do problema podem ser vistos em (MAIA, 2011).

3.3. Formulação Matemática

Esta seção mostra a formulação matemática integrada para os três problemas tratados nesta pesquisa.

3.3.1. Seleção de Casos de Teste

Deseja-se encontrar um vetor binário *STC* de tamanho *M*, onde *STC* representa um subconjunto de casos de testes, no sentido de atingir os objetivos *OS1* e *OS2* e respeitando as restrições *RS1* e *RS2*, a seguir:

$(\mathbf{OS1})\ Max\ \ |suiteCoverage_{STC}|$, onde $suiteCoverage_{STC} = \{r_i \mid r_i \in coverage_{j,} STC_j = 1\}$

$(\mathbf{OS2})\ Max\ suiteImportance_{STC} = \sum(importance_i), \forall\, r_i \in suiteCoverage_{STC}$

Sujeito a:

$(\boldsymbol{RS1}) \sum_{j=1}^{M} executionTime_j * STC_j \leq\ maximumTimeAllowed$

$(\boldsymbol{RS2})\ \forall\ t_{j1}\ e\ t_{j2}\ \in C,\ \left(\left(precedence_{t_{j2}} =\ \mathrm{t_{j1}}\right) \mathrm{e}\ (\mathrm{X_{j2}} = 1)\right) \rightarrow\ (\mathrm{STC_{j1}} = 1)$

Onde a equação (*OS1*) deseja maximizar a quantidade de requisitos cobertos pelo subconjunto de casos de testes representado por *STC*, ou seja, a cardinalidade deste subconjunto. A cobertura pode ser calculada da seguinte forma:

$$suiteCoverage_{STC} = \{r_i \mid r_i \in coverage_{j,} STC_j = 1\}$$

A equação (*OS2*) expressa a importância de *STC*, cujo valor é a soma da importância dos requisitos cobertos por *STC*. Quanto mais requisitos cobertos por *STC* que possuem valor alto para a importância, maior será a importância de *STC*.

A equação (*RS1*) é uma restrição e não permite que a soma dos tempos dos casos de teste selecionados seja maior que o tempo máximo permitido. A constante *maximumTimeAllowed* representa o tempo máximo para os testes.

A equação (*RS2*) é a segunda restrição do problema de seleção de casos de teste, e determina que, caso um caso de teste selecionado possua um caso de teste precedente, este precedente também deve estar presente na suíte de testes representada por *STC*.

3.3.2. Priorização de Casos de Teste

Dado o vetor binário *STC* originado da seleção, deseja-se encontrar um vetor de inteiros *PTC*, onde *PTC* representa um subconjunto ordenado de casos de teste que contém os casos de teste selecionados na fase anterior, no sentido de atingir os seguintes objetivos *OP1* e *OP2* e respeitando a restrição *RP1*, a seguir:

$(\boldsymbol{OP1})\ Max\ \ suiteVolatility_{PTC} = \sum\ (volatility_i * coverageWeight_i)\ , \forall\, r_i\ \in suiteCoverage_{STC})$

$(\boldsymbol{OP2})\ Max\ \ suiteComplexity_{PTC} = \sum\ (complexity_i * coverageWeight_i)\ , \forall\ i\ \in suiteCoverage_{STC})$

Sujeito a:

$(\boldsymbol{RP1})\ \forall\ t_{j1},\ t_{j2} \in C,$
$\left(precedence_{t_{j2}} =\ t_{j1}\right) \rightarrow (testCasePosition_{t_{j1},PTC} < testCasePosition_{t_{j2},PTC})$

A equação (*OP1*) deseja maximizar a volatilidade da suíte de testes *PTC*, que pode ser calculada da seguinte forma:

$$suiteVolatility_{PTC} = \sum (volatility_i * coverageWeight_i)\, , \forall\, i\ \in suiteCoverage_{STC})$$

Para cada requisito *i* pertencente ao conjunto $suiteCoverage_X$, ou seja, para cada requisito coberto pela suíte de testes representada por *STC*, seu valor de volatilidade é acumulado para representar o valor de volatilidade da suíte de teste, ponderado pela posição do primeiro caso de teste na suíte representada por *PTC* que cobre este requisito ($fistTestCasePosition_{PTC,i}$), como pode ser visto abaixo.

$$coverageWeight_i = Y.lenght - firstTestCasePosition_{PTC,i} + 1$$

Quando o primeiro caso de teste que cobre o requisito *i* está no início da suíte de testes *Y*, o valor para $coverageWeight_i$ é maior, tornando maior, nesse caso, a ponderação para sua volatilidade (equação *OP1*) e sua complexidade (equação *OP2*). Isso faz com que as suítes que cobrem os casos de teste mais voláteis mais cedo tenham maior valor para sua volatilidade. A mesma afirmação vale para a complexidade, constante na equação *OP2*.

A equação (*OP2*) expressa a complexidade da suíte de testes *Y*, que pode ser calculada da seguinte forma:

$$suiteComplexity_{PTC} = \sum (complexity_i * coverageWeight_i)\, , \forall\, i\ \in suiteCoverage_{STC})$$

A restrição (*RP1*) reforça a precedência dos casos de teste: na situação onde um caso de teste t_{j1} seja precedente do caso de teste t_{j2}, t_{j1} deve ser executado antes de t_{j2}, ou seja, deve estar antes de t_{j2} na suíte de testes.

3.3.3. Alocação de Casos de Teste

Dado o vetor de inteiros *PTC* originado da priorização, deseja-se encontrar um vetor de inteiros *ATC*, que representa os testadores alocados para cada caso de teste e a ordem de execução desses casos de teste, no sentido de se atingir os objetivos *OA1*, *OA2* e *OA3* e respeitando a restrição *RA1*, a seguir:

$$(\boldsymbol{OA1})\ Min\ numberOfExecutions_{ATC} = \sum_{w=1}^{P} \sum_{a=1}^{Y.lenght} numberOfExecutions_{w,PTC_a} * isAllocated_{w,PTC_a,ATC}$$

$$(\boldsymbol{OA2})\ Max\ preference_{ATC} = \sum_{w=1}^{P} \sum_{a=1}^{PTC.lenght} preferences_{PTC_a,w} * isAllocated_{w,PTC_a,ATC}$$

$$(\boldsymbol{OA3})\ Max\ experience_{ATC} = \sum_{w=1}^{P} \sum_{a=1}^{PTC.lenght} \left(experience_w - penalty_{w,PTC_a}\right) * isAllocated_{w,PTC_a,ATC}$$

Sujeito a:

$(\boldsymbol{RA1})\ \forall\, t_{j1},\ t_{j2} \in C,$

$$\left(\left(precedence_{t_{j2}} = t_{j1}\right) \; and \; (isAllocated_{w,t_{j2},ATC} = 1)\right) \rightarrow (isAllocated_{w,t_{j1},ATC} = 1)$$

A equação (*OA1*) deseja atribuir para cada testador os casos de teste que ele executou menos vezes. A abordagem proposta considera casos de teste que estão sendo testados pela primeira vez, além de casos de teste antigos, já executados (teste de regressão).

A função $isAllocated_{w,PTCa,ATC}$ informa se um caso de teste PTC_a está alocado para o testador w na solução gerada *ATC*. PTC_a representa o caso de teste que será executado na ordem a, contido em *PTC*. Esta função retorna 1 quando o testador w está alocado para o caso de teste PTC_a ou 0, caso contrário.

Então, para cada alocação dada pela função *isAllocated*, o número de vezes que o testador executou o caso de teste correspondente é computado na função *suiteNumberOfExecutions*.

A equação (*OA2*) deseja atribuir aos testadores os casos de teste que eles preferem executar. Para cada alocação dada pela função *isAllocated*, o valor de preferência do testador pelo caso de teste ao qual está alocado é computado na função *suitePreference*.

A equação (*OA3*) deseja alocar os casos de teste a testadores que tem no mínimo a experiência exigida pelo caso de teste. Cada caso de teste possui um atributo chamado "experiência mínima", que informa qual a experiência mínima do testador que deve ser alocado a este caso de teste. Deve ser possível alocar testadores com experiência menor que a requerida pelo caso de teste, porém quando isto acontecer uma penalidade deve ser aplicada à função que representa este objetivo, para torná-la menos competitiva. Para cada alocação dada pela função *isAllocated*, o valor de experiência do testador é computado na função *suiteExperience*, e diminuído da penalidade se for o caso.

A equação (*RA1*) é uma restrição do problema e garantirá que, para dois casos de teste t_{j1} e t_{j2}, se t_{j1} for precedente de t_{j2}, este deve ser executado pelo mesmo testador que executará t_{j2} (na equação, o testador w).

4. Avaliação

Neste capítulo serão detalhados dois experimentos que foram planejados para responder as seguintes questões:

Q1) Qual metaheurística é mais eficiente para a abordagem proposta, dentre as três selecionadas para os experimentos (NSGA-II, MOCell e SPEA2)?

Q2) O uso de técnicas de otimização, conforme a abordagem integrada, interativa e multi-objetiva, é competitiva em relação à respostas de gerentes de teste?

No primeiro experimento, três algoritmos multi-objetivos serão aplicados aos problemas de seleção, priorização e alocação de casos de teste, e serão comparados. No segundo experimento, as respostas dos algoritmos multi-objetivos são comparadas com as respostas de possíveis usuários da abordagem proposta, ou seja, gerentes de teste. Para cada problema, elaborou-se um questionário contendo a descrição do problema e os dados da instância. Além disso, o tempo de resposta dos usuários foi registrado, por problema (seleção, priorização e alocação de casos de teste).

4.1. Algoritmos

Os algoritmos utilizados nos experimentos são: NSGA-II (DEB et al., 2002), MOCell (NEBRO et al., 2009) e SPEA2 (ZITZLER et al., 2001), todos baseados em algoritmos genéticos (HOLLAND, 1975), porém multi-objetivos. Esses algoritmos foram escolhidos devido ao seu bom desempenho em relação aos demais algoritmos multi-objetivos, como pode ser visto em (NEBRO et al., 2008), (ZITZLER et al., 2000) e (ZITZLER et al., 2001).

Por conta da limitação de espaço, não serão apresentados detalhes sobre os algoritmos. Mais detalhes podem ser visualizados em (MAIA, 2011).

Inicialmente, foi realizado um estudo das melhores taxas de recombinação e mutação a serem utilizadas para cada algoritmo. Nesse processo, foram criadas combinações de taxas, e cada uma foi testada para cada algoritmo. Os detalhes sobre as taxas e tipos de recombinação e mutação são detalhados em (MAIA, 2011).

4.2. Instâncias

Para a execução dos experimentos, foram criadas 29 instâncias, de modo aleatório, porém neste artigo são mostrados os resultados de apenas 10. O número de requisitos varia de 20 a 140, o número de casos de teste varia de 40 a 300, o número de testadores de 2 a 15, o percentual de precedência varia entre 20 e 80%, e o tempo máximo da suíte de testes varia de 30 a 90% do tempo total dos casos de teste.

4.3. Indicadores

Nos experimentos, utilizaremos dois indicadores para a comparação: o *hypervolume* (DEB, 2008) e o tempo de execução. O *hypervolume*, que avalia convergência e diversidade, é um indicador que calcula o volume coberto por um conjunto de soluções, usando um ponto de referência (DEB, 2008).

4.4. Experimento 1: Aplicação das Técnicas de Otimização

As metaheurísticas NSGA-II, MOCell e SPEA2, além de um algoritmo randômico, foram aplicados a todas as instâncias. Os resultados desses experimentos são apresentados nas tabelas a seguir. Foram, no total, 348 execuções (3 problemas x 4 algoritmos x 29 instâncias). Neste trabalho publicamos apenas 10 resultados para cada problema, por questões de espaço. O resultado completo pode ser visualizado em (MAIA, 2011).

A Tabela 1 mostra o valor do *hypervolume* para cada instância e algoritmo, para os problemas de seleção, priorização e alocação de casos de teste, respectivamente. Os melhores valores foram destacados em negrito.

De uma maneira geral, o SPEA2 obteve melhor *hypervolume*. Para o problema de seleção de casos de teste, o SPEA2 obteve os melhores valores para esta métrica em 68,97% das instâncias, seguido pelo NSGA-II, com 27,59%. O bom desempenho do SPEA2 se repete para os problemas de priorização de casos de teste, onde obteve melhor *hypervolume* para 72,41% das instâncias, e alocação de casos de teste, obtendo melhor *hypervolume* para 44,82% das instâncias. Como esperado, os piores valores foram obtidos pelo algoritmo randômico, para os três problemas.

Tabela 1. *Hypervolume*

INSTÂNCIA	PROBLEMA	NSGA-II	MOCell	SPEA2	Randômico
BASIC_1	SEL	**0,9754**	0,6777	0,8599	0,2889
	PRI	0,9900	0,9837	**0,9908**	0,8504
	ALO	0,000787	**0,000796**	0,000813	0,000433
BASIC_2	SEL	**0,9592**	0,0000	0,9560	0,3923
	PRI	**0,9974**	0,9081	0,9857	0,7076
	ALO	**0,000049**	0,000045	0,000045	0,000015
SEL_TC_M	SEL	0,9687	0,6808	**0,9974**	0,3510
	PRI	**0,9992**	0,9219	0,9843	0,5141
	ALO	0,000126	0,000161	0,000202	**0,013550**
SEL_TC_L	SEL	0,9171	0,7304	**0,9900**	0,4384
	PRI	0,9680	0,9279	**0,9997**	0,7139
	ALO	0,000110	0,000099	**0,000142**	0,000041
SEL_REQ_S	SEL	0,9901	0,7800	**0,9993**	0,3331
	PRI	0,9792	0,9330	**0,9995**	0,5636
	ALO	**0,000385**	0,000272	0,000257	0,000058
SEL_REQ_M	SEL	0,9815	0,6296	**0,9985**	0,3255
	PRI	**0,9960**	0,9011	0,9889	0,5929
	ALO	0,000188	**0,000255**	0,000125	0,000019
SEL_REQ_L	SEL	0,0000	0,0000	**1,0000**	0,3369
	PRI	**0,9970**	0,9484	0,9758	0,7301
	ALO	0,000031	0,000024	**0,000063**	0,000001
SEL_TIME_S	SEL	**0,9771**	0,4995	0,9637	0,4033
	PRI	0,9970	0,9796	**0,9998**	0,7699
	ALO	0,000328	0,000337	**0,000358**	0,000020
SEL_TIME_M	SEL	0,9759	0,0000	**0,9992**	0,2015
	PRI	**0,9988**	0,9526	0,9900	0,6589
	ALO	**0,000112**	0,000065	0,000093	0,000000
SEL_TIME_L	SEL	0,9547	0,0000	**0,9996**	0,1440
	PRI	0,9754	0,9416	**0,9903**	0,6520
	ALO	0,000096	0,000089	**0,000113**	0,000007

A Tabela 2 apresenta os valores de tempo de execução dos algoritmos. O algoritmo randômico obteve os melhores tempos de execução, porém gerou soluções bem piores que os demais algoritmos. Não considerando o algoritmo randômico, o NSGA-II obteve menores valores para o tempo de execução para os problemas de seleção, em 93,10% das instâncias, e alocação de casos de teste, em 86,21%. Já para os problemas de priorização de casos de teste, os dois melhores foram o SPEA2, em 51,72%, e o MOCell, em 41,38%.

Pode-se notar, porém, que o algoritmo SPEA2 é mais lento que os demais algoritmos. O SPEA2 chegou a ser 36,14 vezes mais lento que o NSGA-II, na instância

BASIC_1, para o problema de seleção de casos de teste. Mas, no contexto geral, o SPEA2 foi 2,1 vezes mais lento que o NSGA-II e 1,32 vezes mais lento que o MOCell. Este tempo é medido em milisegundos.

Tabela 2. Tempo de Execução (ms)

INSTÂNCIA	PROBLEMA	NSGA-II	MOCell	SPEA2	Randômico
BASIC_1	SEL	**578**	2142	20891	31
	PRI	**3394**	7217	10516	140
	ALO	**16048**	21056	27766	141
BASIC_2	SEL	**2406**	4535	5094	78
	PRI	57385	60182	**56593**	719
	ALO	**1494160**	1944050	2254896	4828
SEL_TC_M	SEL	**3828**	6604	9609	125
	PRI	98141	**88010**	**88010**	953
	ALO	**483142**	627135	583538	5250
SEL_TC_L	SEL	**8282**	9901	13719	141
	PRI	177328	**150837**	157801	2266
	ALO	**1660313**	2233863	2252546	15735
SEL_REQ_S	SEL	**2390**	3822	7922	62
	PRI	34297	33786	**32407**	547
	ALO	**362875**	437923	526906	2297
SEL_REQ_M	SEL	**2734**	3997	8109	63
	PRI	**45672**	47870	48625	562
	ALO	**276063**	317276	356098	2656
SEL_REQ_L	SEL	**3281**	5180	6672	63
	PRI	94797	89856	**83892**	953
	ALO	**513658**	910719	821391	2688
SEL_TIME_S	SEL	**3469**	5027	7578	2046
	PRI	**39406**	42203	41125	484
	ALO	**83938**	86797	84767	1984
SEL_TIME_M	SEL	**2438**	4506	7109	47
	PRI	84453	77183	**71125**	906
	ALO	**550560**	798633	712565	3781
SEL_TIME_L	SEL	**2953**	4507	7094	78
	PRI	108031	96532	**89469**	1125
	ALO	**835907**	1252056	1267570	5234

4.5. Experimento 2: Competitividade Humana

O objetivo deste experimento foi comparar o desempenho dos algoritmos multi-objetivos com respostas fornecidas por possíveis usuários desta abordagem.

Alguns possíveis usuários foram selecionados e elaboraram uma resposta para os problemas de seleção, priorização e alocação de casos de teste. Esses usuários são pessoas graduadas em Ciência da Computação, engenheiros de software, sendo que uma parte (cerca de 60%) tem experiência em testes no mercado de trabalho e uma parte tem experiência apenas acadêmica. Para o problema de seleção de casos de teste há 10 respostas, para o problema de priorização há 7 respostas, e para o problema de alocação de casos de teste há 5 respostas. Os usuários não tiveram acesso à formulação sugerida neste trabalho, portanto suas respostas foram intuitivas e baseadas em sua experiência.

Para comparar as soluções, foi calculado o *hypervolume* das soluções geradas pelos usuários, bem como o tempo médio para a resolução dos problemas de seleção, priorização e alocação de casos de teste. Os resultados podem ser vistos na Tabela 3. A média do valor do *hypervolume* e tempo dos algoritmos é em relação à instância BASIC_1, a mesma utilizada pelos usuários.

Tabela 3. Comparação do *Hypervolume* e Tempo de Resposta (em segundos)

	Seleção de Casos de Teste		**Priorização de Casos de Teste**		**Alocação de Casos de Teste**	
	hypervolume	**tempo**	**hypervolume**	**tempo**	**hypervolume**	**Tempo**
Resposta humana	0,0905	2490,00	0,3299	1885,8	0,00003	1656,00
NSGA-II	0,8943	3,39	0,9781	8,25	0,00188	568,21
MOCell	0,3469	5,29	0,9336	7,32	0,00150	778,39
SPEA2	0,9122	8,90	0,9884	7,45	0,00184	824,74

Como pode ser visto na Tabela 3, as soluções geradas pelos algoritmos multi-objetivos gerou soluções mais próximas à Frente de Pareto, devido ao valor bem maior do *hypervolume*.

Para a seleção de casos de teste, houve uma tendência dos usuários em escolher mais requisitos (cobertura maior), não priorizando a importância dos requisitos escolhidos. Já para a alocação de casos de teste, curiosamente, os usuários não priorizarm a preferência dos testadores.

Para validar os algoritmos como ferramenta viável para a busca de soluções dos problemas de seleção, priorização e alocação de casos de teste, o tempo de resposta também foi comparado ao tempo de resposta humana. O tempo de resposta humano foi, em média, 279,78 vezes maior que o tempo que o algoritmo mais lento (SPEA2) usou para encontrar um conjunto de soluções, para o problema de seleção de casos de teste. Para o problema da priorização de casos de teste, o tempo de resposta humano foi 228,58 vezes mais lento que o algoritmo NSGA-II, e 2,01 vezes mais lento que o SPEA2 no problema de alocação de casos de teste. Para este último problema, o tempo de execução dos algoritmos foi bem maior por conta da complexidade da geração da população inicial dos mesmos. O tempo de geração desta população chegou a ocupar mais de 90% do tempo da execução do algoritmo.

Finalmente, é válido destacar que algumas soluções geradas pelos usuários (humanos) foram inválidas, não respeitando as restrições dos problemas. Para o problema de seleção de casos de teste, a metade das soluções (5 soluções) foram inválidas, e para o problema de priorização de casos de teste, 2 das 7 soluções foram inválidas. O cálculo do *hypervolume* e do tempo levou em consideração apenas as soluções válidas geradas pelos usuários.

5. Conclusões

Este trabalho mostrou a efetividade do uso de técnicas de otimização em três problemas de Teste de Software. Para a questão Q1 da pesquisa, o resultado do experimento 1 mostrou que o algoritmo SPEA2 obteve melhor desempenho, em relação ao *hypervolume*. A maioria das soluções geradas por ele compôs a Frente de Pareto ótima, formada pelas melhores soluções dos três algoritmos. O *hypervolume* do SPEA2 foi melhor em 68,97% das instâncias no problema de seleção de casos de teste, 72,41% no problema de priorização de casos de teste, e 58,62% no problema de alocação de casos de teste. A desvantagem deste algoritmo foi seu tempo de execução.

Para a questão Q2, pode-se destacar que o *hypervolume* e o tempo de execução das soluções formadas pelas respostas humanas foram bem piores que os valores para os algoritmos. O tempo de resposta humano foi aproximadamente 279,78 vezes maior que o do algoritmo mais lento (SPEA2) no problema de seleção, 228,58 vezes mais lento no problema de priorização, e 2,01 vezes mais lento no problema de alocação.

Pode-se citar duas ameaças à validade dos experimentos: a configuração dos algoritmos, visto que um número maior de combinações de parâmetros poderiam ser testados, e o uso de dados gerados, pois o algoritmo que os gerou pode tê-lo feito de maneira viciada.

Um possível trabalho futuro seria comparar os algoritmos utilizados com outros que não sejam algoritmos evolucionários. Um segundo trabalho seria realizar alterações da formulação matemática dos problemas, como, por exemplo, considerar vários precedentes para um caso de teste, ou limitar a quantidade de casos de teste por testador.

Referências

Antoniol, G., Kpodjedo, S., Ricca, F., Galinier, P. (2009) “Evolution and Search Based Metrics to Improve Defects Prediction”, In: *Proceedings of the 1st International Symposium on Search Based Software Engineering*, p. 23-32.

Bagnall, A. J., Rayward-Smith, V. J., Whittley, I. M. (2001) “The Next Release Problem”, Information and Software Technology, Elsevier, v. 43, n. 14, p. 883-890.

Bastos, A., Cristalli, R., Moreira, T., Rios, E. (2007), Base de Conhecimento em Teste de Software, Martins Fontes, segunda edição.

Deb, K., Agrawal, S., Pratap, A., Meyarivan, T. (2002) “A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II”, In: *Proceedings of the Parallel Problem Solving from Nature VI Conference*, Springer, p. 849-858.

Deb, K. (2008), Multi-Objective Optimization using Evolutionary Algorithms, John Wiley & Sons, 1st Edition.

Di Penta, M., Harman, M., Antoniol, G., Qureshi, F. (2007) "The effect of communication overhead on software maintenance project staffing: a search-based approach", IEEE International Conference on Software Maintenance, p. 315-324.

Di Penta, M., Harman, M., Antoniol, G. (2009) "The use of Search-Based Optimization Techniques to Schedule and Staff Software Projects: an Approach and an Empirical Study", Software – Practice and Experience.

Holland, J. H. (1975), Adaptation in Natural and Artificial Systems, The University of Michigan Press.

Kuperberg, M., Omri, F. (2009) "Using Heuristics to Automate Parameter Generation for Benchmarking of Java Methods", Electronic Notes in Theoretical Computer Science, v. 253, n. 1, p. 57-75.

Li, R., Chaudron, M. R. V., Ladan, R. C. (2009) "Towards Automated Software Architectures Design using Model Transformations and Evolutionary Algorithms", In: *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation (GECCO'10)*, p. 1333.

Li, Z., Harman, M., Hierons, R. M. (2007) "Search Algorithms for Regression Test Case Prioritization", IEEE Transactions on Software Engineering, v. 33, n. 4, P. 225-237.

Maia, C. L. B., Carmo, R. A. F., Freitas, F. G., Campos, G. A. L., Souza, J . T. (2009) "A Multi-Objective Approach for the Regression Test Case Selection Problem", In: *Proceedings of XLI Simpósio Brasileiro de Pesquisa Operacional*, p. 1824-1835.

Maia, C. L. B. (2011) "Uma Abordagem Integrada, Interativa e Multi-Objetiva para os Problemas de Seleção, Priorização e Alocação de Casos de Teste", Dissertação de Mestrado da Universidade Estadual do Ceará (UECE).

Nebro, A., Durillo, J., Coello, C. C., Luna, F, Alba, E. (2008) "A Study of Convergence Speed in Multi-Objective Metaheuristics", Parallel Problem Solving from Nature, Springer, p. 763-772.

Nebro, A. J., Durillo, J. J., Luna, F., Dorronsoro, B., Alba, E. (2009) "MOCell: A cellular genetic algorithm for multiobjective optimization", International Journal of Intelligent Systems, v. 24, n. 7, p. 726-746.

Walcott, K. R., Soffa, M. L., Kapfhammer, G. M., Roos, R. S. (2006) "Time-Aware Test Suite Prioritization", In: *Proceedings of the International Symposium on Software Testing and Analysis*, p. 1-12.

Yoo, S., Harman, M. (2007) "Pareto Efficient Multi-Objective Test Case Selection", In: *Proceedings of International Symposium on Software Testing and Analysis (ISSTA'07)*, p. 140-150.

Zitzler, E., Deb, K, Thiele, L. (2000) "Comparison of Multiobjective Evolutionary Algorithms: Empirical Results", Evolutionary Computation, v. 8, n. 2, p. 173-195.

Zitzler, E., Laumanns, M., Thiele (2001) "SPEA2: Improving the Strength Pareto Evolutionary Algorithm".