# Knapsack e NSGA-II: Uma abordagem híbrida para a seleção de casos de teste

**Lorena do Carmo Caldas**
GSORT[1], Instituto Federal da Bahia, 1
Rua Emídio dos Santos, S/N - Barbalho, Salvador - BA, 40.301-015, 2
lorecaldas@ifba.edu.br

**Antônio Maurício da S. Pitangueira 1, 2**
antoniomauricio@ifba.edu.br

**RESUMO**

Este trabalho discute a combinação entre dois algoritmos de alocação de recursos para selecionar casos de teste manuais para regressão de forma ótima. Isso envolve a aplicação dos algoritmos NSGA-II e Knapsack sobre a ferramenta TSuite. Esse novo método compõe um candidato à solução do problema NP-Completo da seleção de casos de teste. O problema envolvido nesta seleção inclui as condições de tempo, prioridade e complexidade dos casos de teste e as restrições de tempo total disponível no projeto de teste e prioridade das suítes de teste. A abordagem combinada aumenta a quantidade de casos de teste relevantes selecionados em 3,2% por oferecer justiça aos casos de complexidade baixa.

**PALAVRAS CHAVE. Seleção de casos de teste, Teste de regressão, Método Híbrido.**

**Área Principal: (ADM – Apoio à Decisão Multicritério, MH –Metaheurísticas, OC – Otimização Combinatória)**

**ABSTRACT**

This paper discusses the combination of allocation research algorithms to select manual test cases for regression in optimal form. It envolves the application of NSGA-II e Knapsack algorithms on TSuite's tool. Its new method compose a candidate to solving the NP-Complete test cases selection problem. The problem envolved in this selection includes conditions of time, priority and test cases complexity and constraints total time available in test project and test sets priority. The combined approach increases the amount of relevant test cases selected in 3,2% by offers justice to cases of low complexity.

**KEYWORDS. Test Cases Selection, Regress Test, Hybrid Method.**

**Main Area: (MDS - Multicriteria Decision Support, MH – Metaheuristics, CO – Combinatorial Optimization)**

[1]Grupo de Pesquisa em Sistemas Distribuídos, Otimização, Redes e Tempo Real

## 1. Introdução

Teste de *software* é o processo de constante revisão de produtos de trabalho realizado com o intuito de assegurar o correto funcionamento de programas. Esses programas podem ser de computador (*desktop*), de internet (*web*) ou estar embutidos em dispositivos móveis (*mobile*) ou equipamentos operacionais (embarcados), a exemplo de aparelhos industriais, carros, eletrodomésticos, entre outros. Os programas citados, também denominados *softwares* ou sistemas, são conjuntos de rotinas computacionais próprias ao funcionamento de serviços de um domínio específico. Eles podem ser utilizados para diversos fins, entre os quais são comuns: gerenciamento de recursos, automatização de atividades, comunicação em rede, entre outros usos.

Para que um *software* seja construído é necessário realizar minimamente as etapas de: especificação, projeto e codificação. As atividades de teste de *software* ainda que presentes no processo de construção, são consideradas atividades complementares. Elas consomem boa parte dos esforços despendidos na construção de um programa, Black (2002). Isso porque tais atividades podem permear diversas fases do desenvolvimento do projeto, por incluir os fatores de verificação e validação. Esses fatores são necessários à realização do controle de qualidade do produto, Bastos et. al. (2007). Isso gera custos ao projeto, pois é necessário que, para cada atividade desempenhada ou produto de trabalho gerado para o projeto, seja realizada revisão e análise de impacto no programa. Sendo assim, qualquer forma de reduzir o tempo de execução das atividades de revisão, sem degradar o processo de garantia da qualidade ou a qualidade do próprio produto é considerável. Essa redução implica na economia de pessoal e custos para a realização do projeto.

Quando um *software* passa pelo processo de revisão de qualidade com o teste de *software*, é adequado avaliá-lo sob várias perspectivas, como: estrutura interna (caixa branca) e estrutura externa (caixa preta). A análise do sistema sob essas perspectivas envolvem a avaliação: do código-fonte, persistência dos dados, *layout*, usabilidade, entre outros aspectos. Isso, por vezes, implica na execução repetida de uma rotina de atividades de teste. Este é o caso da tarefa de teste de regressão, o tipo de teste que é realizado sobre um módulo já testado em versões anteriores do sistema. Ela é necessária quando vários cenários de teste precisam ser reconduzidos para considerar se uma parte relevante do produto foi danificada por alterações aplicadas a esta região ou outra que dependa desta para funcionar.

Os cenários de teste são geralmente modelados como casos de teste: documentos que contém o passo a passo referente à execução de cenários de teste sobre o programa. Um caso de teste pode ser manual, quando escrito em linguagem natural, ou automático, quando contém instruções de operação próprias à uma ferramenta do tipo *record and play* (gravação e ação). Os casos de teste podem ser agrupados de acordo com as características que apresentam. Essa ação contribui na organização do projeto e por isso facilita pesquisas e ação posteriores relacionadas à recuperação desses dados. Uma suíte de testes é um conjunto de casos de teste e um projeto de teste é constituído por várias suítes de teste. Quando o projeto possui grande volume de dados, ainda que organizados, implica em dificuldade e demora em consultá-los. A utilização de uma ferramenta que extraia e automatize a busca por critérios constitui importante auxílio nas atividades que envolvam a manipulação desses dados, pois reflete diminuição dos esforços em executá-las.

A ferramenta TSuite foi criada com o objetivo de automatizar a tarefa de seleção dos casos de testes manuais. Ela é discutida no trabalho Caldas e Pitangueira (2013) e serviu de base para os avanços descritos neste artigo. A função do TSuite é priorizar os cenários mais importantes para a atividade de teste de regressão, no momento atual do projeto, considerando o tempo que se tem para executar o ciclo de regressão. O artigo atual propõe a adequação do método de seleção utilizado pelo TSuite para elevar o seu desempenho.

O TSuite faz uso de técnicas da SBSE (*Search Based Software Engineering*) para priorizar e selecionar casos de teste. SBSE é um ramo da Engenharia de *Software* que aplica os métodos da PO (Pesquisa Operacional) na resolução de problemas matemáticos com suporte computacional. Por vezes isso ocorre com base em algoritmos computacionais de busca e de alocação de recursos,

Freitas et. al. (2010). O TSuite utilizava um método híbrido, composto por um algoritmo próprio, o TSuite, e por um algoritmo ótimo, o NSGA-II.

O desenvolvimento do trabalho descrito neste artigo evoluiu o método de seleção original do TSuite. Atualmente são utilizados os algoritmos Knapsack e NSGA-II para compor seu método híbrido. O algoritmo NSGA-II foi escolhido por ser um algoritmo ótimo (metaheurística) adequado ao problema que é atacado pelo TSuite, Farzat e Barros (2010). Ele é um algoritmo genético: um algoritmo que aplica filtros sucessivos sobre uma população de soluções para escolher aquelas mais propícias e variadas, conforme contexto do problema. Os algoritmos genéticos foram criados com inspiração na teoria da evolução pela escolha seletiva proposta pelo cientista Charles Darwin. O algoritmo NSGA-II representa importante contribuição aos trabalhos SBSE já consolidados – Pratap et. al. (2002), Pitangueira, A. M., et. al (2013). – e por isso ele foi aplicado no TSuite para oferecer variedade de soluções na priorização dos cenários de teste.

O algoritmo de alocação de recursos Knapsack é largamente utilizado na computação como forma de solucionar o problema da Mochila, um problema de alocação de recursos já difundido no campo da Engenharia de *Software*, Molina et. al (2002). Ele é utilizado inclusive para resolver questões de escalonamento de processos dos sistemas operacionais. Sobre o TSuite, o Knapsack cumpre o papel de alocar os casos de testes da melhor forma possível, considerando o tempo disponível para realizar a atividade de teste de regressão. Através da união desses dois algoritmos, Knapsack e NSGAII, passou a ser possível priorizar os casos mais importantes e selecioná-los da maneira mais proveitosa à situação atual do projeto (recursos) no TSuite.

No trabalho Caldas e Pitangueira (2013), testes foram conduzidos para comprovar a validade do método utilizado no motor de seleção do TSuite. Ele apresenta o benefício da redução de aproximadamente 50% do tempo de execução da atividade de seleção de casos de teste para regressão, em comparação com o meio natural: a execução manual. No entanto, foi necessário evoluir a sua forma de priorizar e selecionar os casos de teste manuais para melhorar os resultados apresentados pela ferramenta. A aplicação da alocação dinâmica disponibilizada pelo algoritmo Knapsack, no método de seleção do TSuite, representa avanços ao seu desempenho e resultado ainda mais relevante do que aquele encontrado em seu método de seleção anterior.

Testes realizados com base no método Knapsack-NSGAII, proposto neste artigo, comprovam o aumento de 3,2% do resultado alcançado pela ferramenta TSuite, em relação ao método que ele utilizava anteriormente, o TSuite-NSGAII. Isso significa que o uso combinado entre essas duas técnicas efetivamente amplia a possibilidade de encontrar melhores resultados neste tipo de problema de alocação de recursos. Juntos, os algoritmos Knapsack e NSGA-II puderam prover este resultado pelo aumento da quantidade de casos de testes em relação aos testes realizados no trabalho Caldas e Pitangueira (2013) em iguais condições.

Este artigo está organizado como descrito a seguir. A seção Problema contextualiza o leitor no cenário a que se deve aplicar o trabalho aqui proposto. A seção Trabalhos Correlatos apresenta os artigos que descrevem contexto de problema semelhantes ao citado neste. A seção Método define o meios e os algoritmos utilizados para alcançar a solução do problema relatado. A seção Resultados exibe os números obtidos com a execução do método descrito na seção anterior. E por fim, a seção Conclusão resume este artigo e apresenta trabalhos futuros.

## 2. O Problema

No trabalho publicado por Chen, Rosenblum e Vo (1994) um estudo de caso baseado no problema da seleção de casos de teste é definido como “encontrar uma suíte reduzida para as alterações do programa”. Yoo e Harman (2010) e Rothermel e Harrold (1997) explicam que o problema da seleção de casos de teste para regressão deve estar relacionado a um projeto que contenha diversas versões a ser comparadas. Segundo Fazart e Barros (2010) este é um problema de classe matemática NP-Completo porque dele participa a decisão multicritério que necessita de elevado tempo computacional polinomial para ser solucionado. Esse gasto de tempo ocorre por

conta das várias combinações que são realizadas a ritmo exponencial à medida que as características do problema são dispostas dinamicamente em um algoritmo.

Este artigo apresenta um método ótimo candidato à resolução do problema da seleção dos casos de testes para regressão. Ele está inserido no contexto da ferramenta TSuite e por isso considera as variáveis informadas nela. Para a proposta apresentada neste trabalho serão consideradas as variáveis de: complexidade, tempo de execução e relevância da funcionalidade associada, por unidade de caso de teste; e as restrições de: prioridade das suítes de teste e tempo total disponível para a execução da atividade. As variáveis citadas compõem o peso do caso de teste, que é a característica utilizada para a priorizá-los. A restrição de tempo total disponível é usada para alocar os casos de maior relevância. Todas elas participam das equações dispostas no modelo matemático exibido na Figura 1.

No modelo de problema matemático apresentado na Figura 1, a premissa 1 visa minimizar o tempo de execução do somatório dos casos de teste. A premissa 2 visa minimizar o risco da ocorrência de falhas na ferramenta, através da cobertura relacionada aos casos selecionados. Isso é conseguido através da análise de impacto, adquirida pela atribuição de prioridades às suítes que se relacionam em primeiro e segundo grau, no modelo arquitetural do sistema. A premissa 3 visa maximizar a importância dos casos selecionados, algo que está relacionado à relevância dos casos. É este fator que proporciona maior evidência aos casos pertencentes às suítes prioritárias no momento atual do projeto e que portanto necessitam passar pelo ciclo de regressão. Esses são os casos associados aos novos módulos do sistema ou aqueles modificados para compor a sua versão atual. A restrição do modelo, a inequação a), é baseada no tempo máximo disponível para a execução dos casos de teste relevantes. Ou seja, o tempo de projeto relacionado à realização do ciclo de testes de regressão, dentro do processo de teste de *software*.

1. Minimizar $\sum_{j=1}^{k} TempoExecução(t_j)$
2. Minimizar $\sum_{j=1}^{k} Risco(t_j)$
3. Maximizar Importância($t_j$)

Sujeito à:
a) $\sum_{j=1}^{k} TempoExecução(t_j) \leq T_{max}$

Figura 1: Modelo Matemático do Problema TSuite

A ferramenta TSuite considera o problema da seleção de casos de teste para regressão no contexto dos projetos de teste mantidos na ferramenta *opensource* (código aberto) TestLink. TestLink é uma ferramenta *web* para o registro e persistência dos dados pertinentes à execução de testes manuais ou automatizados FST Community (2012). Ela possibilita a importação e exportação de projetos de testes, em formato XML, o que permite que esses planos sejam manipulados em um ambiente exterior, assim como o TSuite o faz. Para tanto, os projetos de teste cadastrados no TestLink devem possuir execuções realizadas anteriormente ao momento atual do projeto e novas suítes a serem executadas neste novo período. Eles também precisam disponibilizar tempo para a execução do ciclo de regressão sobre o sistema maior que zero.

Através do TSuite, o executor pode informar um projeto de teste no formato TestLink e então realizar a seleção dos casos prioritários. Para que isso aconteça, é necessário compor um modelo arquitetural do *software* em teste, no TSuite, pela associação dos módulos do sistema que se relacionem. Essa associação deve acontecer para consolidar a atribuição de prioridades às suítes de teste, que à esta altura representam cada qual um módulo do sistema, em um relacionamento do tipo

um para um. As principais funcionalidades que precisam ser selecionadas também são informadas na ferramenta para fazer prevalecer os casos mais importantes ao momento atual do projeto. Através do uso do TSuite é possível selecionar os casos de teste manuais para regressão, contando com a independência de experiência e conhecimento total do modelo do sistema, por parte do executor da atividade.

## 3. Trabalhos Correlatos

No trabalho de Chen, Rosenblum e Vo (1994), é apresentado um método para selecionar casos de teste, chamado TestTube. Ele foi escrito em linguagem de programação C e é próprio aos casos de teste unitários definidos para as funções também escritas nesta linguagem. Quando uma classe passa por alterações, todas as funções associadas a ela são selecionadas para novo teste unitário.

O trabalho de Rothermel e Harrold (1997) apresenta a ferramenta DejaVu. Ela implementa uma família de algoritmos própria à seleção dos casos de teste. A ferramenta realiza análise de impacto com base na teoria dos grafos. No entanto, com as mudanças pontuais realizadas sobre uma parte do sistema, todo o módulo é selecionado para passar pela regressão, assim como o TestTube o faz. O TestTube e DejaVu são ferramentas para selecionar casos de teste unitários, o que difere do TSuite, que é próprio à seleção dos casos de teste de sistema e de integração. Seus métodos de seleção são próprios e o TSuite atualmente utiliza-se dos algoritmos Knapsack e NSGA-II para apresentar evolução de desempenho. Também, o TSuite visa cobrir somente os casos prioritários ao momento atual do projeto, sem que seja necessário testar todo um módulo de sistema por conta de mudanças pontuais sobre uma de suas funções.

No trabalho de Medeiros (2008) são descritos alguns métodos para seleção de casos de teste baseados em risco, incluindo a proposta de método inserida neste mesmo trabalho. Ele cita o método definido por Besson (2004), que tenta priorizar as funcionalidades mais importantes do sistema para as testar e atribui quantitativo de esforço despendido na execução dos casos de teste. Os casos de teste de menor esforço são mais relevantes, dada a utilização mínima dos recursos disponíveis. O TSuite, ampliou a justiça sobre os casos de teste de complexidade baixa, assim como o método Besson propõe, pela utilização do método Knapsack-NSGAII. O método próprio, proposto por Medeiros, visa maximizar a cobertura dos casos de teste e minimizar os esforços no ciclo de regressão, premissas essas aplicadas no modelo de problema proposto pelo TSuite. Porém, seu experimento é baseado em uma planilha eletrônica que reúne todas as características necessárias à sua seleção específica. O TSuite utiliza os projetos de teste mantidos na ferramenta TestLink e utiliza os recursos oferecidos por ela para compor as características necessárias à priorização e seleção dos casos, validando a sua utilização em ambiente corporativos e outras realidades de execução de teste de *software*.

O trabalho de Farzat e Barros (2010), apresenta igual problema ao definido neste artigo de maneira reduzida. Sua proposta utiliza o algoritmo NSGA-II como meio de selecionar os casos de teste. O TSuite, emprega este algoritmo, na combinação Knapsack-NSGAII para priorizar e trazer maior variedade aos casos de teste selecionados.

## 4. Descrição da Abordagem Proposta

SBST (*Search Based Software Testing*) é uma vertente da SBSE voltada somente aos problemas de busca relacionadas ao teste de *software*. O problema do TSuite se enquadra no campo da SBST e esta ferramenta utiliza seus métodos para prover a melhor seleção possível ao grupo de casos de teste de um projeto de teste.

Para este trabalho foi considerada a combinação entre esses dois algoritmos: Knapsack e NSGA-II para observar os resultados obtidos em relação a esta nova forma de selecionar casos de teste para regressão. Foi alcançada a melhoria de 3,2% com o uso combinado dos algoritmos citados em relação ao método que utiliza somente o NSGA-II como algoritmo ótimo na seleção dos casos sobre o TSuite. O uso do algoritmo genético NSGA-II propicia variedade na priorização

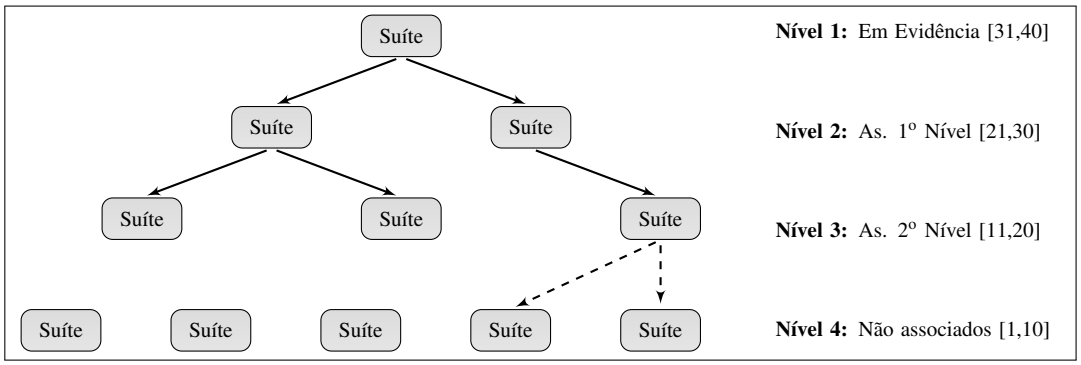


Figura 2: Esquema de associação de suítes e atribuição de pesos do TSuite

e posterior seleção dos casos de teste. Já o algoritmo de alocação Knapsack provê o esquema de seleção mais proveitoso, com base nos recursos disponibilizados para consumo sobre o projeto. Ao aplicá-los juntos, sobre o TSuite, foi oferecida justiça aos casos de teste menos custosos ao projeto. Esta é a forma de tornar a seleção dos casos direta e proporcional aos recursos associados a cada categoria de casos de teste. Desta maneira, foi possível ampliar a cobertura de testes sobre o sistema sem que houvesse aumento dos recursos consumidos para tanto.

O algoritmo Knapsack realiza a alocação dos recursos conforme o espaço disponível para eles no contexto geral do problema. No caso do TSuite, este espaço é representado pelo tempo total que se tem para realizar a tarefa de regressão. Utilizados em conjunto, ele e o NSGA-II, é possível obter a melhor seleção possível com base na alocação mais proveitosa ao problema que se quer atacar.

A escolha dos casos é feita com base em um *ranking* realizado pelo TSuite, no momento da associação entre as suítes do projeto de teste. Todos os casos, antes de passar pela seleção, ganham pesos, que correspondem à soma do valor de cada característica envolvida na unidade de caso de teste e da suíte em que está inserida. Quando o executor da ferramenta realiza as associações entre as suítes, o projeto de teste passa a apresentar o modelo definido na Figura 2 para o TSuite. O esquema é constituído por quatro níveis, cada qual correspondendo a um intervalo de valores que são os pesos dos casos. Esses pesos simbolizam a relevância dos casos enquadrados nas suítes.

Do Nível 1 participam os casos mais prioritários ao momento atual do projeto. Eles fazem parte da nova versão do sistema e a partir dele deve ser traçada a análise de impacto com base na associações desses casos com aqueles contidos nas suítes dos demais níveis do projeto. O segundo nível possui os casos dispostos nas suítes que se associam em primeiro grau com o Nível 1. O terceiro nível contém aqueles que se associam em segundo grau. Eles se relacionam diretamente com os casos do Nível 2, mas indiretamente com o Nível 1. O Nível 4 contém os casos que não se associam com nenhum dos outros pertencentes aos outros níveis do projeto ou que se associam com aqueles do segundo nível, mas que não são contabilizados como associados porque tem rastreabilidade fraca como aqueles do Nível 1.

O tempo unitário dos casos de teste é utilizado como critério para o consumo do tempo total do projeto destinado à realização do ciclo de regressão. Ele está associado à complexidade dos casos, que é dividida em três categorias. São elas: complexidade Baixa, quando possui até cinco passos para se executar; complexidade Média, quando possui entre 6 e 10 passos para se executar; e complexidade Alta, quando ele possui mais que 10 passos para se executar. O tempo é atribuído dinamicamente sobre a ferramenta pelo executor dela e conta para todos os casos que se enquadre àquela categoria de complexidade.

É, também, com base nos pesos dos casos que o algoritmo NSGA-II retorna seus resultados ótimos. Eles são utilizados para selecionar os casos que correspondam àquele valor atribuído. O Knapsack oferece suporte ao consumo do tempo disponível, por agrupar aqueles casos que caibam no espaço destinado a essa restrição. O peso total atribuído aos casos é 40. Do nível 4 participam

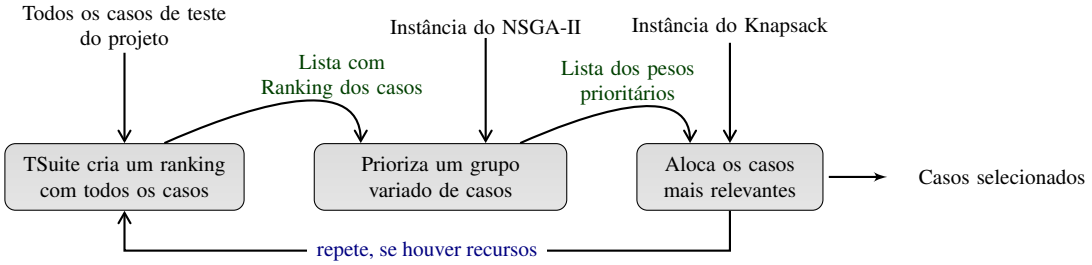


Figura 3: Fluxo de execução do método Knapsack-NSGAII

os casos com peso do intervalo 1 a 10. Do nível 3, do intervalo 11 a 20. Do nível 2, do intervalo 21 a 30. Do nível 1, participam os casos mais prioritários, são aqueles com peso contido no intervalo de valores 31 a 40. Com base neste *ranking* é que ocorre a seleção do método Knapsack-NSGAII.

A seleção utilizada no TSuite era divida em duas etapas: a seleção estática e a seleção dinâmica. A seleção estática contava com método específico do TSuite. Todos os casos dos níveis 1 e 2 eram selecionados, conforme tempo total disponível. Caso restasse tempo para selecionar novos casos, a seleção dinâmica, aquela que utilizava o método ótimo , o algoritmo NSGA-II, era realizada. Dela participavam os casos dos níveis 3 e 4. O método híbrido, apresentado neste artigo, visa substituir esta forma de seleção. Nele, o NSGA-II prioriza os casos e o Knapsack os aloca para que ocorra a seleção dos casos contidos em todos os níveis de peso. Ele seguirá o fluxo descrito na Figura 3.

O método híbrido exibido no Algoritmo 1 descreve três entradas para sua execução: todos os casos de teste ainda não selecionados, tempo total restante e uma instância do algoritmo NSGA-II. O algoritmo é originalmente escrito em linguagem Java, respeitando os princípios da Orientação a Objetos, por isso ele necessita da instância de um objeto para executar suas operações. O algoritmo deve retornar seu resultado em um vetor, contendo todos os casos alocados pelo Knapsack. Durante este processo todos os casos disponíveis são avaliados e caso tenham sido escolhidos, participam do consumo do tempo total. Na condição em que todos os casos alocados tenham sido selecionados e ainda reste tempo disponível para selecionar, ao menos um caso de complexidade baixa, então a rotina de seleção ótima é invocada novamente. Isto torna o método Knapsack-NSGAII recursivo.

Antes que ocorra a alocação e assim a seleção dos casos, o método híbrido Knapsack-NSGAII invoca a priorização dos casos com base no algoritmo NSGA-II. Isso acontece conforme apresentado no Algoritmo 2.

Essa parte do algoritmo recebe como entrada uma lista dos casos ainda disponíveis para seleção. O resultado deve seguir a forma de uma lista preenchida com todas as soluções ótimas encontradas pelo algoritmo genético. Internamente, o algoritmo cria uma geração, seleciona os melhores indivíduos daquela geração, realiza combinações entre eles, aplica um novo fator determinante, se necessário, para trazer maior variedade de soluções à geração e, por fim, seleciona os melhores indivíduos com a conclusão de todo o processo. Esse ciclo é repetido diversas vezes, até que o objetivo do problema seja atingido. No caso do TSuite, a meta é encontrar os melhores casos de teste presentes no projeto, contando com suas características representadas como peso. Essa parte do fluxo de operação do método Knapsack-NSGAII retorna os melhores e mais variados pesos selecionados, para que o Knapsack os aloque da melhor forma possível, conforme apresentado no Algoritmo 1. A relação entre esses dois algoritmos combinados está descrito na Figura 3. Ela exibe todos os passos que são realizados pelo método híbrido para selecionar os casos mais relevantes à

situação atual do projeto.

**Algoritmo 1:** Fluxo principal do método Knapsack-NSGAII

```
Data: lista de casos de teste não selecionados
Data: tempo total disponível para alocação
Data: instância do algoritmo NSGA-II
Result: vetor contendo os casos selecionados
Cria e inicializa a lista casosSelecionados;
for (tem casos a alocar) do
    if (tempo disponível >= menor custo) then
        executa o algoritmo NSGA-II;
        preenche casosSelecionados;
    end
end
cria instância do algoritmo Knapsack;
if (casosSelecionados > 0) then
    executa o algoritmo Knapsack;
    aloca os casos de teste selecionados;
end
```

**Algoritmo 2:** Fluxo do algoritmo NSGA-II no método Knapsack-NSGAII

```
Data: lista de casos de teste não selecionados
Result: lista de casos priorizados
while tem recursos do
    cria uma geração;
    seleciona os melhores indivíduos;
    aplica crossover nos indivíduos;
    if (indivíduos muito parecidos) then
        aplica mutation na geração;
    end
    seleciona as melhores soluções;
end
retorna os pesos selecionados;
```

Ambos algoritmos, Knapsack e NSGAII atuam sobre um espaço de busca comum no método Knapsack-NSGAII. Para o problema do TSuite, o espaço de busca é um plano cartesiano que contém como eixos as características dos casos de teste (peso) e onde elas podem ser combinadas (peso x custo) para trazer as soluções propostas à resolução dele. Esse espaço é primeiramente utilizado pelo algoritmo NSGA-II. Quando o Knapsack opera sobre o espaço de busca, as características observadas passam a ser peso x benefício. Com a utilização dele, todas as soluções retornadas seguem uma sequência crescente gradual no espaço de busca. Elas são desenhadas desde o limite inferior até o limite superior do espaço, o que significa dizer que este algoritmo explora os extremos do espaço de busca e por isso ele não realiza apenas busca local. Já o algoritmo NSGA-II trabalha sobre esse mesmo espaço de busca de maneira cíclica. Ele realiza várias buscas locais, que combinadas contemplam boa parte deste plano. Juntos, o Knapsack e NSGA-II realizam uma busca cíclica gradual que reflete melhor utilização dos recursos disponibilizados, conforme será apresentado na próxima seção.

| Complexidade do Caso de teste | Tempo de Execução |
|---|---|
| *Baixa* | 12 minutos |
| *Média* | 24 minutos |
| *Alta* | 48 minutos |
| | |
| Tempo Total Disponível | 8h |

Tabela 1: Dados utilizados nos testes

## 5. Resultados

Testes foram conduzidos sobre a ferramenta TSuite contendo o método híbrido Knapsack-NSGAII para selecionar os casos de teste manuais para regressão. Utilizando como base as hipóteses 4 e 5 apresentadas no trabalho Caldas e Pitangueira (2013) é possível observar a evolução do resultado obtido pela ferramenta ao utilizar este novo método como meio de realizar a automatização da seleção dos casos.

Os testes foram realizados com base nos mesmos parâmetros para todos os métodos envolvidos. Os métodos de execução Manual e TSuite foram validados por uma dupla de analistas de testes com experiência de mais de 2 anos na atividade de execução de testes, de posse do modelo arquitetural do sistema em questão.

Um sistema *web* para cadastro e controle de rotinas de um estabelecimento comercial foi utilizado como base para os testes. Seu modelo serviu de fundamento para os testes de todos os métodos. O projeto de testes utilizado foi definido também conforme os moldes desse sistema. Ele possui 84 casos de teste, sendo o valor de 28 casos para cada complexidade de caso. Ambos os métodos foram testados utilizando os dados da Tabela 1, com tempo máximo de 30 minutos para a realização da atividade de seleção.

A Tabela 2 exibe a comparação entre esses valores. Os métodos: Manual, TSuite, Knapsack, NSGA-II e Knapsack-NSGAII foram avaliados para tanto. Na tabela, a linha Tempo apresenta os minutos que foram gastos para realizar a atividade. A linha Quantidade lista qual o montante alcançado por utilizar certo método de seleção. A linha Variedade dos Casos, especifica o montante obtido com base na complexidade dos casos. A letra A simboliza a complexidade Alta, a letra M simboliza a complexidade Média e a letra B simboliza a complexidade Baixa. Por fim, a linha Tempo não gasto apresenta quantos minutos não foram utilizados na seleção.

A seguir serão apresentadas e discutidas as hipóteses relativas ao método proposto para a solução do problema da seleção de casos de teste para regressão. São elas:

1. **Hipótese 1:** O método combinado Knapsack-NSGAII seleciona mais casos de testes do que aqueles algoritmos utilizados de forma isolada.
   *Prova:* O resultado apresentado na Tabela 2 para o método Knapsack-NSGAII é de maior valor do que aqueles exibidos para os métodos Knapsack e NSGA-II em separado. Isso acontece porque foi ampliada a justiça sobre os casos de teste de complexidade baixa em relação aos casos de complexidade média e alta. O montante de 15 casos de complexidade baixa selecionados implica, respectivamente, no crescimento de 25% e 7,14% do aproveitamento dos casos dessa categoria em relação àqueles selecionados pelos métodos usados isoladamente. Esse valor impacta na quantidade final de casos selecionados por método. Por possuir valor de consumo inferior aos casos de complexidade média e alta, é possível selecionar mais casos de teste de complexidade baixa com o tempo total disponível para executar a atividade de regressão.

   Enquanto o método Knapsack-NSGAII selecionou 32 casos de teste em sua melhor execução, os métodos Knapsack e NSGA-II selecionaram 29 e 31, nesta sequência, em iguais condições. Já em sua primeira execução, o método híbrido obteve seu melhor resultado, algo que só foi obtido pelo NSGA-II e Knapsack em sua quarta execução. Analisando os resultados

| Variáveis | Ex. Manual | TSuite | TS. Knapsack | TS. NSGAII | Knap. NSGAII |
|---|---|---|---|---|---|
| *Tempo(min)* | 26 | *11 | 11 | 11 | 11 |
| *Quantidade* | 19 | 23 | 29 | 31 | 32 |
| *Variedade dos casos* | 6A 6M 7B | 9A 10M 4B | 7A 10M 12B | 6A 11M 14B | 7A 10M 15B |
| *Tempo não gasto (min)* | 204 | 0 | 0 | 0 | 0 |
| | | | | | *tempo médio |

Tabela 2: Resultado dos Testes

obtidos por todos os métodos operados sobre o TSuite, em relação ao método de seleção manual, todos se mostraram mais proveitosos do que aquele, por não desperdiçar recursos e por selecionar mais casos, inclusive em menor tempo. Esses aspectos que foram ampliados, em benefícios, com a implementação do método Knapsack-NSGAII.

2. **Hipótese 2:** A combinação entre os algoritmos Knapsack e NSGA-II permite melhor uso do espaço de busca próprio ao problema TSuite, em relação ao uso desses métodos isolados. *Prova:* A combinação entre os algoritmos ótimos, de busca: NSGA-II e de alocação: Knapsack proveu o melhor resultado na seleção dos casos para regressão, conforme exibem os dados contantes na Tabela 2. E isso pode ser considerado consequência do melhor aproveitamento do espaço do busca gerado para a resolução do problema, se comparado ao uso isolado deles sobre este mesmo espaço. Como mostra o gráfico de dispersão de valores, presente na Figura 5, os três métodos: TSuite-Knapsack, TSuite-NGSAII e Knapsack-NSGAII atuam de maneira diferente sobre o espaço de busca citado. O eixo 'x' do gráfico simboliza os pesos e o eixo 'y' os benefícios.

   Quando usado como algoritmo ótimo em separado, no método TSuite-Knapsack, o algoritmo Knapsack cresce exponencialmente sobre o espaço. Isso faz com que ele retorne valores que tangem os extremos do espaço de busca. No entanto, seus resultados não apresentam soluções variadas, já que o fator da distância euclidiana entre elas não é tratada por este método. O que significa dizer que o espaço que as separa não é explorado pelo algoritmo. Já o NSGA-II, quando utilizado em separado, no método TSuite-NSGAII, não explora as dimensões extremas do espaço de busca, mas realiza buscas cíclicas em um local estrito do espaço. O seu desenho sobre o gráfico sempre será em ondas (união das parábolas) e este é o aspecto que torna suas soluções mais variadas do que aquelas retornadas pelo método Knapsack. O NSGA-II considera as soluções vizinhas para garantir que aquelas selecionadas não sejam parecidas entre si (distância euclidiana).

   Considerando os resultados apresentados para o método Knapsack-NSGAII, na Figura 4, é possível perceber que ele alcança as dimensões extremas do gráfico, mas que também realiza buscas cíclicas que crescem de maneira constante. Isso significa que seu uso propicia a obtenção dos maiores e menores valores de peso sem que seja desconsiderada a variação dessas soluções.

   O gráfico apresentado na Figura 5 mostra de forma mais clara o desempenho do método Knapsack-NSGAII, comparando os resultados das suas 4 melhores execuções de 10. Todas as execuções seguem igual desenho, e inclusive alguns valores dispostos se repetem, no entanto, a 1ª execução apresenta uma forma contendo curvas mais acentuadas do que as demais, o que proporcionou maior variação na exploração do espaço de busca e por isso contribuiu para a maior quantidade de casos selecionados no total. A proximidade entre seus desenhos também aponta para a pouca possibilidade de degradação do desempenho desse método, além de coerência ao objetivo de resolução do problema.

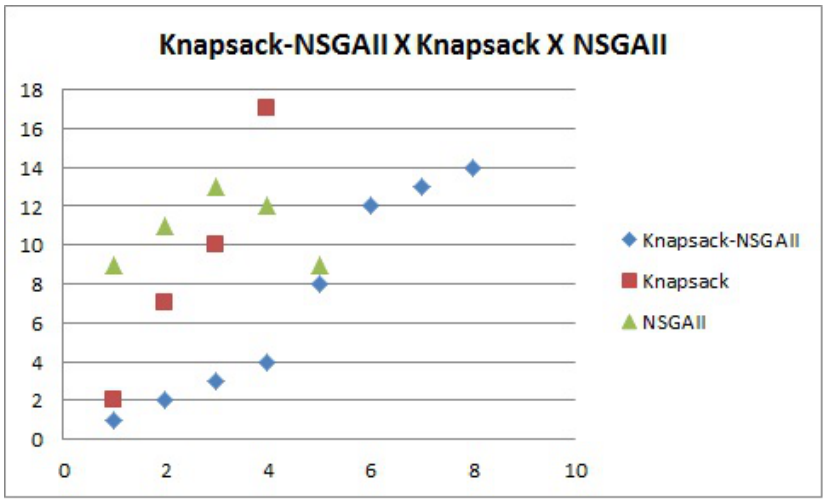


Figura 4: Comparação do desempenho entre os métodos testados

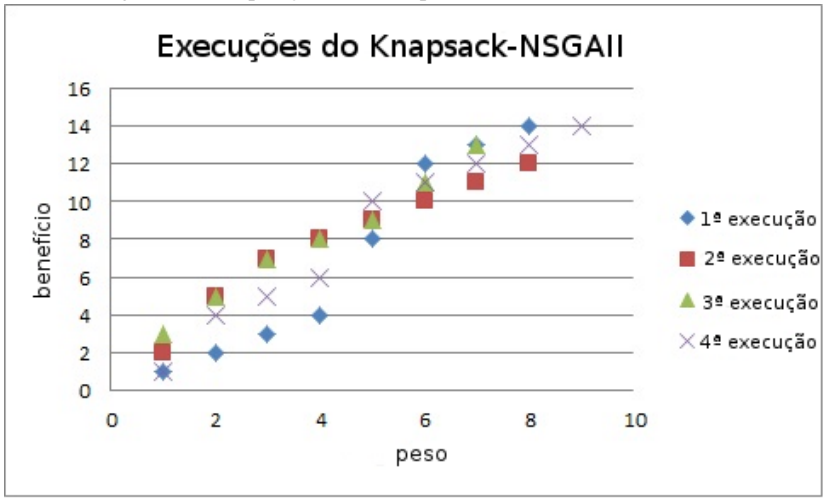


Figura 5: Execuções do método Knapsack-NSGAII

## 6. Conclusão

As atividades de teste de *software* permeiam várias fases do processo de desenvolvimento de sistemas e por isso elas são consideradas dispendiosas. Algumas tarefas precisam ser repetidas para que seja consolidada a garantia de qualidade sobre os produtos gerados. Este é o caso do teste de regressão. No entanto, é necessário selecionar os casos de teste mais relevantes ao momento atual do projeto para que não ocorra desperdício de tempo e seja alcançada a cobertura de testes ideal sobre o *software*.

Este trabalho apresenta uma ferramenta para auxiliar na resolução do problema NP-Completo da seleção de casos de teste. Neste problema estão envolvidos os aspectos de minimização do tempo de execução, minimização do risco e maximização da cobertura sobre os casos de teste. Está envolvida também a restrição de: tempo total disponível.

O contexto de seleção dos casos está contido no TSuite, uma ferramenta criada para automatizar a priorização e seleção dos casos de teste manuais para regressão. Seu método de seleção foi modificado para receber a combinação entre os algoritmos Knapsck e NSGA-II. Ambos tratam da alocação de recursos, no entanto, sobre o TSuite, o NSGA-II realiza a priorização dos casos e o Knapsack os aloca conforme tempo total disponível para a execução da atividade de regressão.

Com a mudança do método de seleção, o TSuite passou a apresentar 3,2% de melhoria na quantidade de casos de teste selecionados para regressão. Isso se deve à combinação Knapsack-NSGAII, que permite uma melhor exploração do espaço de busca do problema e por isso garante a

seleção de mais casos de complexidade baixa. Aumento esse que equivale a, respectivamente, 25% e 7,14% do montante dos casos de teste de complexidade baixa selecionados isoladamente pelos métodos TSuite-Knapsack e TSuite-NSGAII.

Com a implementação deste método, uma parte dos trabalhos futuros citados no trabalho Caldas e Pitangueira (2013) foi solucionada, quando ele relata a necessidade de melhoria ou mudança do seu método de seleção para que traga um resultado mais variado aos resultados que o TSuite apresenta. A adaptação do método TSuite-NSGAII para Knapsack-NSGAII proveu este tópico. No entanto, questões como análise da precedência e abertura ao tratamento dos casos de teste unitários ou automatizados ainda estão em aberto para implementações futuras.

**Referências Bastos, A., Rios, E., et. al.**, Base de conhecimento em teste de *software*, Martins, 2007.

**Besson, S.** (2004), A Strategy for Risk-Based Testing, http://www.stickyminds.com.

**Black, R.**, Managing the testing process: Practical tools and techniques for managing hardware and software testing, John Wiley & Sons, 2002.

**Caldas, L. do C. e Pitangueira, A. M. d. S.** (2013), A tool for optimal manual test case selection. 10th International Conference on Information Systems and Technology Management – CONTECSI.

**Chen, Y.; Rosenblum, D., et. al.** (1994), Testtube: A system for selective regression testing. pages 211–222.

**Farzat, F. de A. e Barros, M. d. O.** (2010), Método para seleção de casos de teste para alterações emergenciais. I Workshop Brasileiro de Otimização em Engenharia de *Software*.

**Free *Software* Test Community.**, User manual testlink version 1.9, http://www.teamst.org, 2012.

**Freitas, F. G. de, Maia, et. al.** (2010)., Otimização em teste de *software* com aplicação de metaheurísticas.

**Lina, Y.; Choua, C., et. al.** (2012). ,Test coverage optimization for large code problems. page 16–27. The publisher of Systems and Software, 85.

**Medeiros, A.** (2008), Método de Seleção de Casos de Teste de Regressão Baseado em Risco. Universidade Federal de Pernambuco.

**Molina, A. C., Flórez, C. A. C e Bolaños, A. R.** (2008), Algoritmo multiobjetivo NSGA-II aplicado al problema de la mochila.

**Pitangueira, A. M., et. al** (2013), A Systematic Review of Software Requirements Selection and Prioritization Using SBSE Approaches. Search Based Software Engineering, 8084, Lecture Notes in Computer Science, Springer Berlin Heidelberg.

**Pratap, A, Agarwal, S e Deb, K.** (2002), A fast anda elistic multiobjective genetic algorithm NSGA-II. IEEE transaction on Evolucionary Computation.

**Rothermel, G. e Harrold, M. J.** (1997), Experience With Regression Test Selection, Kluwer Academic Publishers, 1997. pages 178-188.

**Yoo, S. e Harman, M.** (2010), Regression testing minimization, selection and prioritization: a survey. Software Testing, Verification and Reliability.