Revista de Sistemas de Informação da FSMA
n. 5 (2010) pp. 3-13

SISTEMAS DE INFORMAÇÃO
http://www.fsma.edu.br/si/sistemas.html

Otimização em Teste de *Software* com Aplicação de Metaheurísticas

Fabrício Gomes de Freitas[1], Camila Loiola Brito Maia[2], Gustavo Augusto Lima de Campos[3].
Jerffeson Teixeira de Souza[4]

Grupo de Otimização em Engenharia de Software da UECE (GOES.UECE)
Laboratório de Computação Natural e Inteligente (LACONI)
Universidade Estadual do Ceará (UECE)
Avenida Paranjana, 1700 – Fortaleza, CE – Brasil
fabriciogf@uece.br, camila.maia@gmail.com, gustavo@larces.uece.br, jeff@larces.uece.br

***Resumo*— Durante a fase de Teste de *Software* são encontrados alguns problemas que não podem ser resolvidos com técnicas tradicionais da Engenharia de *Software*. Contudo, tais problemas podem ser modelados matematicamente e serem resolvidos através de otimização matemática, especialmente com uso de metaheurísticas. Nesse contexto, surgiu um recente campo denominado *Search-based Software Engineering* (SBSE) que trata da resolução de problemas de Engenharia de *Software* por meio de técnicas de otimização. Dada a importância da fase de Teste de *Software*, uma subárea denominada *Search-Based Software Testing* (SBST) se destaca no campo. Este artigo objetiva apresentar o estado atual desta recente área de pesquisa e mostrar seu potencial. Para isso, inicialmente são descritas as principais metaheurísticas empregadas na área. Em seguida, o estado da arte da SBST é apresentado através da descrição dos principais problemas já modelados e dos resultados já conseguidos. A partir dos resultados, indicamos que os resultados são promissores.**

***Index Terms*— Teste de *Software*, Otimização Matemática**

I. Introdução

A relevância da Engenharia de *Software* é notável para o importante estágio atual de desenvolvimento de sistemas. Desde a sua criação na década de 1970, importantes avanços na qualidade do *software* produzido foram conseguidos devido à pesquisa em modelos, normas e metodologias de suporte ao desenvolvimento de sistemas [1]. A aplicação de tais métodos acontece nos mais diversos aspectos relacionados ao processo de desenvolvimento, como Planejamento de Projeto, Análise de Requisitos e Teste de *Software*.

Infelizmente, em alguns casos, as normas e metodologias convencionais desenvolvidas pela comunidade científica especializada não são capazes de resolver certos problemas que acontecem durante o processo de desenvolvimento de *software*, ou o fazem de modo insatisfatório. Isso acontece com problemas intrinsecamente complexos, como aqueles envolvendo a seleção de uma solução em um conjunto proibitivamente grande de possibilidades. Em problemas desse tipo são exigidas formas automatizadas de resolução para que o problema possa ser resolvido de forma eficiente.

Na fase de Teste de *Software*, especificamente, são encontrados problemas que podem ser modelados dessa forma. Por exemplo, considere a atividade de priorização de casos de teste. Esta atividade trata da determinação da melhor ordem de execução dos casos de teste de um sistema. A definição da qualidade de uma ordem é realizada por meio de uma métrica de cobertura definida matematicamente para calcular o quanto tal ordem executa cedo todo o sistema. Apesar de tal métrica poder ser descrita em texto, a resolução do problema em si se mostra em demasia custosa dada a quantidade de permutações possíveis entre os casos de teste. Assim, a solução para o problema de priorização de casos de teste não pode ser facilmente descrita por meio de regras textuais ou passos escritos em documentos e normas. Entretanto, a modelagem matemática do mesmo como um problema de otimização se mostra conveniente dada a existência de características matemáticas especiais deste problema.

Percebeu-se que de fato é possível unir essas duas áreas da computação de forma a possibilitar a resolução de problemas que antes não poderiam ser solucionados de forma satisfatória. Assim, inúmeros trabalhos foram realizados nos últimos anos demonstrando a aplicabilidade de técnicas de otimização matemática em diversos problemas da Engenharia de *Software*, especialmente em problemas da fase de testes. Recentemente, percebeu-se que de fato é possível unir essas

[1] Fabrício Gomes de Freitas é graduando em Ciências da Computação na Universidade Estadual do Ceará (UECE).
2 Camila Loiola Brito Maia é mestranda em Ciências da Computação na Universidade Estadual do Ceará.
3 Gustavo Augusto Lima de Campos é Doutor em Engenharia Elétrica pela Universidade Estadual de Campinas, Professor Permanente do Mestrado Acadêmico em Ciências da Computação da UECE.
4 Jerffeson Teixeira de Souza é PhD em Ciência da Computação pela University of Ottawa, Professor Permanente do Mestrado Acadêmico em Ciências da Computação da UECE.

duas áreas da computação de forma a possibilitar a resolução de problemas que antes nem poderiam ser solucionados e os termos "*Search-based Software Engineering* (SBSE)" e "*Search-based Software Testing* (SBST)" foram definidos para representar as áreas. O uso do termo "busca" (*search*, em Inglês) se dá porque as técnicas de otimização utilizadas na resolução são conhecidas como algoritmos de busca. Em especial, as técnicas de otimização utilizadas são metaheurísticas devido ao fato de as mesmas geralmente apresentarem boas soluções para os problemas, independente de particularidades da instância ou do seu tamanho.

O presente artigo apresenta uma revisão bibliográfica de trabalhos relacionados ao uso de metaheurísticas em problemas de Teste de *Software*. Nesse sentido, esta versão desenvolve uma versão anterior relacionada à Engenharia de *Software* de forma geral [2] com o intuito de apresentar mais detalhes aos trabalhos descritos. Dessa forma, pretende-se alcançar os seguintes objetivos:

- Difundir este recente campo de pesquisa para engenheiros de *software* para que os mesmos tenham conhecimento da possibilidade de resolução de problemas complexos durante a fase de desenvolvimento;
- Instigar pesquisadores relacionados com sistemas de informação que com a experiência podem, a partir do levantamento realizado, adicionar conhecimento ou ideias para a área, como a modelagem de novos problemas;
- Servir como base de conhecimento e guia para futuros pesquisadores interessados no campo, funcionando como um panorama do estado da arte da área e indicando avanços que ainda podem ser conseguidos.

O artigo inicia com a exposição sucinta da teoria de otimização matemática, incluindo a otimização multiobjetiva. Além disso, são apresentadas cinco metaheurísticas amplamente utilizadas na resolução de problemas de SBST. Em seguida, indicamos os fatores que favorecem a utilização das técnicas de otimização pelas empresas de desenvolvimento de sistemas como complemento às técnicas convencionais atuais. Após isso, a seção principal apresenta o estágio atual da pesquisa em SBST, incluindo cinco áreas, a saber: geração de dados de teste; seleção de casos de teste; priorização de casos de teste; testes não funcionais; e testes funcionais. Ao final, a conclusão apresenta considerações finais sobre o cenário geral de aplicações de metaheurísticas em Teste de *Software* e indica trabalhos futuros na área.

II. Otimização por Metaheurísticas

A atividade de otimização representa maximizar ou minimizar uma função matemática definida a partir de coeficientes e variáveis. As variáveis que definem a função podem estar sujeitas a restrições, ou seja, as variáveis devem satisfazer um conjunto de equações definidas de acordo com cada instância do problema. Entre as técnicas que podem ser utilizadas para resolver problemas de otimização se encontram as heurísticas. Tais métodos são caracterizados por, em geral, serem específicos para o problema a ser resolvido.

O termo metaheurísticas foi introduzido por Glover [3] e representa uma classe de algoritmos genéricos de busca. Assim, metaheurísticas são heurísticas de forma genérica, ou seja, que podem ser utilizadas em diferentes tipos de problemas. Estes métodos utilizam ideias de diversos domínios como inspiração para realizar o processo de busca da solução para problemas de otimização. Por exemplo, a metaheurística *Têmpera Simulada* [4] é baseada em um processo físico da metalurgia. Outros exemplos são os *Algoritmos Genéticos* [5] que são fundamentados em conceitos da evolução populacional.

O processo geral de execução de uma metaheurística trata da busca de soluções a partir da visitação de regiões do espaço de soluções guiando-se por funções de satisfação [6]. A função de satisfação é uma função matemática que atribui um valor a cada solução do espaço de busca. Para comparação, considere uma técnica exaustiva: para determinar a melhor solução de um problema, todas as soluções existentes são visitadas e ao final a melhor é retornada. Outro tipo de técnica que pode ser usada para a resolução de problemas de otimização são os métodos exatos. Em tal abordagem a busca é efetuada a partir de decisões fundamentadas em teoremas matemáticos.

O funcionamento das metaheurísticas faz com que apenas algumas das soluções existentes sejam visitadas para a determinação de uma solução. A maneira como o processo de busca segue é específico de cada metaheurística, mas é uma maneira que tenta de forma inteligente encontrar boas soluções. Contudo, não existe garantia de a solução retornada por uma metaheurística ser a melhor possível.

A utilização de metaheurísticas pode ser justificada devido a alguns fatores, entre eles [7]:

- Complexidade interna do problema que impede a aplicação de técnicas exatas;
- Quantidade muito grande de possíveis soluções que impede a utilização de técnicas exaustivas.

Para maior compreensão do tema do presente trabalho, a seguir descrevemos a teoria de otimização com mais detalhes. Além disso, cinco metaheurísticas utilizadas nos trabalhos em SBST citados neste artigo são explicadas. As metaheurísticas apresentadas estão divididas em duas seções: otimização mono-objetivas e otimização multiobjetivas. A diferença entre essas classes reside apenas na quantidade de funções atacadas pelas técnicas. No caso da otimização mono-objetiva, as metaheurísticas realizam o processo de busca seguindo apenas uma função de satisfação. Em técnicas de otimização multiobjetiva, como o nome sugere, a busca deve considerar mais de uma função ao mesmo tempo. Os algoritmos descritos a seguir são quatro mono-objetivos (*Hill-Climbing*, Têmpera Simulada, GRASP e Algoritmos Genéticos) e uma metaheurística multiobjetiva (NSGA-II).

A. *Otimização mono-objetiva*

Na otimização mono-objetiva, como apresentado, a busca de soluções é realizada de acordo com os valores de uma função de satisfação. Assim, se considerarmos um problema de maximização e tivermos uma solução com valor maior que outra, tal solução com valor maior de função de satisfação é melhor que a segunda. O processo continua seguindo este conceito até que algum critério de parada, como tempo máximo de execução, seja satisfeito.

1) *Hill-Climbing*

A técnica *Hill-Climbing* é definida como um método de busca local de soluções [8]. O seu nome, "subida de colina", em uma tradução livre, representa a atividade que é efetuada durante o processo de busca: a partir da solução corrente a próxima solução é tomada a partir da vizinhança local de forma que esta próxima solução seja melhor que a anterior. O uso do termo "subida" é uma referência a problemas de maximização, onde o objetivo é encontrar soluções que apresentem o maior valor para a função de satisfação. A Figura 1, a seguir, ilustra este conceito.

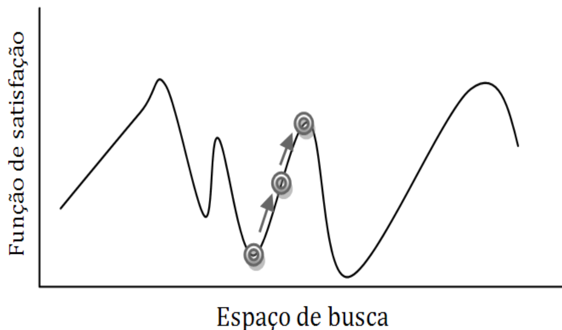

Fig. 1. Processo de busca do algoritmo *Hill-Climbing*.

Pode-se perceber que o método *Hill-Climbing* apresenta uma estratégia simples de busca. De fato, esse é um dos mais simples algoritmos de otimização conhecidos. Contudo, tal simplicidade representa uma desvantagem para o processo de busca. Isto acontece porque esta técnica é capaz apenas de encontrar a melhor solução localmente. Em outros termos, o método *Hill-Climbing* não explora o possível grande espaço de busca que pode ter soluções melhores das encontradas apenas na vizinhança da solução atual.

É importante destacar que o método *Hill-Climbing* pode ser utilizado tanto em problemas de maximização quanto em problemas de minimização. No caso de minimização, o algoritmo faria o mesmo processo de busca local, mas as soluções encontradas dariam menores valores para a função de satisfação

2) *Têmpera simulada*

A metaheurística *Têmpera Simulada* é considerada a mais conhecida entre as metaheurísticas [4]. Seu procedimento de busca é baseado em um processo físico real que ocorre na metalurgia de ligas e metais. Especificamente, o processo tomado como base é originalmente definido como *annealing* e traduzido como "têmpera". Outro termo utilizado é "resfriamento", tendo em vista a atividade que este processo efetua sobre o material.

No processo de têmpera, um material é aquecido com altas temperaturas e, após, é resfriado de forma que ao final de todo o processo o material se encontre em um estado cristalizado de energia mínima. Na relação com a otimização matemática, o objetivo é minimizar o valor da função de satisfação a partir das soluções encontradas durante o processo de busca. Novamente, perceba que a Têmpera Simulada pode ser aplicada tanto a problemas de maximização quanto a problemas de minimização. Se a função deve ser minimizada, o algoritmo já está pronto. No caso de o desejado ser a maximização da função, basta considerar o oposto da mesma, pois a minimização do oposto de uma função corresponde à maximização da mesma.

O algoritmo *Têmpera Simulada*, diferentemente da técnica *Hill-Climbing*, permite a aceitação de soluções que não melhoram o valor da função de satisfação. Por exemplo, em problemas de minimização é possível a aceitação para a próxima iteração de uma solução com valor maior que a solução corrente. Estas aceitações contra o objetivo são controladas a fim de definir um processo de busca inteligente. A forma de controle de aceitação deriva de funções estatísticas definidas para o processo real. Em resumo, o algoritmo funciona da seguinte forma: se a nova solução encontrada é melhor que a solução corrente, então ela é aceita; caso contrário, se a nova solução encontrada piora o objetivo, então ela é aceita com certa probabilidade definida em termos da diferença entre as soluções, do valor atual da variável de temperatura e de uma constante física. Ilustrativamente para um problema de minimização com função de satisfação 'f', o processo é como mostra a Figura 2.

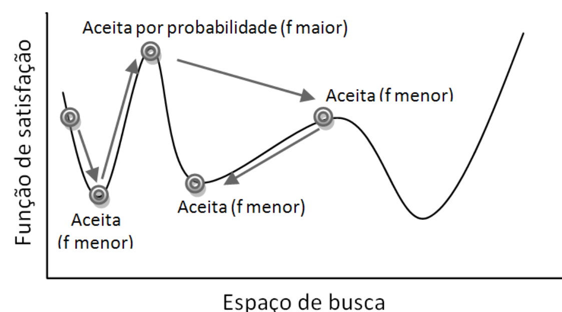

Fig. 2. Processo de busca do algoritmo Têmpera Simulada.

Existem várias variações do algoritmo original. Entre as mais usadas destaca-se aquela na qual durante o processo de busca o valor da variável de temperatura é decrescido. Isso é feito para simular o resfriamento realizado no processo real de têmpera. Esta diminuição do valor da temperatura faz com que a probabilidade de aceitação de soluções piores definida em fórmulas físicas diminua durante o processo de busca. Dessa forma, o algoritmo apresenta um fator de convergência teórica ao aceitar soluções melhores no limite de tempo e ao mesmo tempo permitir a diversificação do processo de busca ao aceitar certas vezes soluções piores.

3) GRASP

O algoritmo GRASP (*Greedy Randomized Adaptive Search Procedure*) [9] é uma técnica que parte de uma estratégia gulosa para o processo de busca. Nessa metaheurística, a busca é antecedida por um processo de construção através da inclusão de componentes de solução. O termo '*Randomized*' ("aleatório") do nome desse algoritmo representa a aleatoriedade da escolha dos componentes que poderão ser selecionados na fase de construção. Após a construção de uma solução, o processo de busca é iniciado. Ao final, a solução encontrada é comparada com as já encontradas em iterações anteriores. A melhor solução desse conjunto é tomada como a solução atual. A Figura 3, a seguir, ilustra o algoritmo.

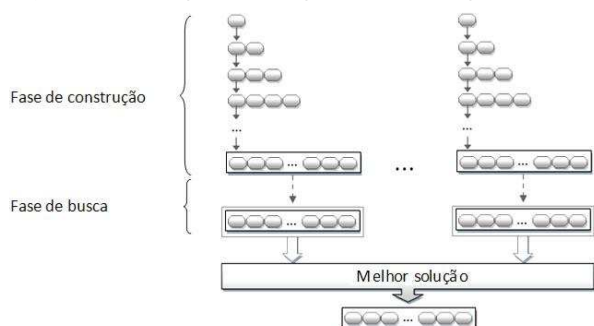

Fig. 3. Esquema de funcionamento da metaheurística GRASP.

Uma alteração no GRASP original apresentado é a técnica GRASP Reativo [10]. A alteração em tal abordagem em relação ao GRASP original reside na alteração no fator de aleatoriedade da seleção dos componentes realizada na fase de construção. Assim, durante a execução do algoritmo, a seleção dos componentes varia entre a possibilidade de escolher todos os componentes ou apenas o melhor.

4) Algoritmos genéticos

A metaheurística *Algoritmos Genéticos* [5] é um método bastante empregado para a resolução de problemas de otimização. Como seu nome indica, essa técnica usa conceitos da Genética, como população e mutação.

O funcionamento da metaheurística se resume basicamente ao emprego de duas operações genéticas: o *crossover*, no qual as informações estruturais de duas soluções são cruzadas a fim de gerar duas novas soluções; e a mutação, processo pelo qual algumas alterações aleatórias podem acontecer nas soluções geradas. Após isso, as soluções atuais são avaliadas para a determinação de quais continuam para a próxima iteração. Dessa forma, as soluções vão sendo selecionadas de acordo com o desempenho em relação à função de satisfação. Como as novas soluções são geradas a partir dessas soluções que foram selecionadas, o processo evolui com o intuito de gerar soluções cada vez melhores. Para que o processo não gere necessariamente as mesmas soluções e para que o processo explore vários espaços de busca, a operação de mutação é aplicada com uma pequena taxa sobre as soluções.

B. Otimização multiobjetiva

As metaheurísticas apresentadas anteriormente são mono-objetivas. Isto significa que elas resolvem problemas com apenas uma função. Nesses problemas a escolha entre soluções é feita simplesmente pela comparação dos valores de cada uma para o objetivo considerado. Por exemplo, se o problema é de maximização, uma solução com maior valor de função de satisfação é melhor do que outra com valor inferior.

Alternativamente, o problema pode ser modelado com mais de uma função. Nesses casos a comparação entre soluções não é um processo trivial já que existe mais de um valor para comparação. Para resolver tal questão, as abordagens de otimização multiobjetivas apresentam o conceito chamado de f*rente de Pareto*. Este conceito representa um conjunto com as melhores soluções possíveis para o problema considerando todos os objetivos ao mesmo tempo. Em outras palavras, este conjunto é formado por soluções que não são piores que nenhuma outra considerando todos os objetivos. Este conceito de ser pior ou melhor em relação a outras soluções, conhecido como "dominância", pode ser usado para uma definição formal da *frente de Pareto*: as soluções pertencentes à *frente de Pareto* são aquelas que não são dominadas por nenhuma outra solução, ou seja, são soluções melhores do que todas as outras em no mínimo um aspecto.

1) NSGA-II

A metaheurística NSGA-II (*Non-dominated Sorting Genetic Algorithms II*) [11] é uma metaheurística multiobjetiva baseada em *Algoritmos Genéticos*. Os conceitos de cruzamento e mutação são correspondentes aos já apresentados de forma recorrente na área dos AGs.

O NSGA-II inicia com a geração, de forma aleatória, de um conjunto P_0 de N soluções (uma população de soluções). Em seguida, um segundo conjunto Q_0, também de tamanho N, é gerado com o uso dos operadores de permutação e mutação. As soluções dos dois conjuntos formam a população R_0 de tamanho 2N. Então, R_0 é ordenado usando o conceito de dominância da seguinte maneira: as soluções que não são dominadas por nenhuma outra são colocadas na primeira *frente de Pareto*, F_1; as soluções que são dominadas por uma solução são colocadas na segunda *frente de Pareto*, F_2; e assim sucessivamente até que todas as soluções de R_0 estejam em alguma *frente*. Então, uma população P_1 é formada considerando as *frentes de Pareto* iniciais até o limite de N soluções. Quando este limite impede que todas as soluções de uma *frente* sejam tomadas, um operador de distância de soluções é utilizado na seleção de quais serão tomadas para P_1. Este operador tem o objetivo de selecionar soluções que não se encontrem próximas de outras no espaço de busca. Esta operação é realizada com o intuito de tomar soluções diferentes. A população P_1 é então utilizada como ponto inicial na iteração seguinte, ou seja, uma população será formada a partir da aplicação dos operadores genéticos, e as soluções das duas populações serão ordenadas de acordo com a dominância para a seleção de quais participarão da próxima geração, como mostra a Figura 4.

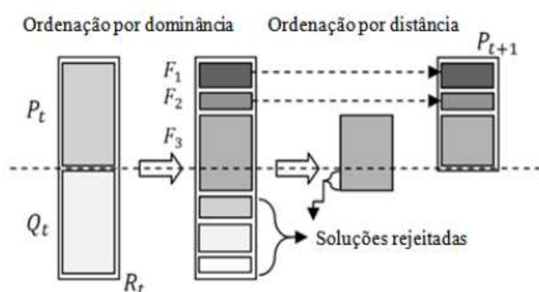

Fig. 4. Procedimento geral do NSGA-II (Adaptação [11]).

III. OTIMIZAÇÃO EM ENGENHARIA DE *SOFTWARE*

A Engenharia de *Software*, como uma disciplina de engenharia, é uma área na qual existem problemas que podem ser resolvidos matematicamente [12]. Tais problemas podem ser caracterizados pela busca da solução em um espaço de possíveis soluções. Os problemas dessa forma apresentam, em geral, elevada quantidade de possibilidades e alta complexidade de resolução. Assim, o processo de busca para estes problemas deve ser feito por meio de um método automatizado e não exaustivo, permitindo que os envolvidos possam realizar atividades onde a capacidade humana é mais conveniente [13] [14].

TABELA I
ÁREAS E PROBLEMAS JÁ ATACADOS EM SBSE

Área da Engenharia de *Software*	Principais Problemas
Engenharia de Requisitos	Análise de Requisitos Seleção de Requisitos Planejamento de Requisitos
Teste de *Software*	Geração de Dados de Teste Seleção de Casos de Teste Priorização de Casos de Teste Testes Não Funcionais
Estimativa de *Software*	Estimativa de Tamanho Estimativa de Custo
Planejamento de Projeto	Alocação de Recursos Alocação de Pessoal
Otimização de Código-Fonte	Paralelização Otimização para Compilação
Manutenção de *Software*	Re-engenharia de *Software* Automated Maintenance
Otimização de Compilador	Alocação em Heap Tamanho de Código
Projeto de *Software*	Modularização

O primeiro passo nessa direção aconteceu em 1976 [15]. Neste trabalho, o objetivo foi gerar dados de teste a partir de maximização numérica. Contudo, foi em um segundo momento, mais precisamente em 2001, que um trabalho de apresentação da área [16] indicou esta área de pesquisa e denominou o termo "*Search-based Software Engineering* (SBSE)". A partir deste momento, diversos pesquisadores em parceria com engenheiros de *software* e desenvolvedores de sistemas têm modelado e resolvido diversos problemas da Engenharia de *Software* utilizando técnicas de busca, especialmente metaheurísticas. A Tabela 1 demonstra a amplitude da aplicação de metaheurísticas em problemas desta área.

Nesse contexto, uma das áreas da Engenharia de *Software* mais destacadas em SBSE é a de Teste de *Software*. Este destaque é dado principalmente pela quantidade de problemas dessa área que já se mostraram possíveis de serem modelados e resolvidos através de técnicas de otimização matemática. Dado o maior destaque da área Teste de *Software* no campo SBSE, um subcampo específico para ela foi criado: *Search-based Software Testing*.

Neste ponto, é importante salientar que a SBST, assim como a SBSE, complementa as técnicas convencionais da Engenharia de *Software* já utilizadas no desenvolvimento de sistemas. Nesse sentido, o caso é que alguns problemas que antes não eram completamente resolvidos, resolvidos de forma não satisfatória ou sequer considerados pelas metodologias e métodos convencionais da Engenharia de *Software* tradicional, passam a ser estudados e solucionados indicando a relevância do campo.

IV. OTIMIZAÇÃO EM TESTE DE *SOFTWARE*

Nesta seção são apresentados diversos trabalhos e resultados em otimização em Teste de *Software*. Entre os problemas atacados estão a geração de dados de teste e a seleção e priorização de casos de teste. A Tabela 2, a seguir, indica alguns problemas já atacados em SBST, suas principais características e contribuições, e as referências.

TABELA II
PRINCIPAIS PROBLEMAS ATACADOS EM SBST E REFERÊNCIAS

Problema de Teste de *Software*	Observações	Referência
Geração de Dados de Teste	Uso de algoritmos genéticos	[17]
	Formalização do problema	[18]
	Maximização dos caminhos	[19]
	Quantidade de vezes do teste	[20]
	Uso de tempera simulada	[21]
Seleção de Casos de Teste	Análise de cinco técnicas	[22]
	Abordagem multiobjetiva inicial	[26]
	Nova formulação multiobjetiva	[27]
	Seleção segura de casos de teste	[28]
Priorização de Casos de Teste	Relaçãode priorização e seleção	[26]
	Formalização do problema	[29]
	Tempo de execução como fator	[30]
	Comparação de técnicas	[31]
	Uso do GRASP Reativo	[32]
Testes Não Funcionais	Teste de performance	[33]
	Teste de tempo de execução	[34]
	Testes em contextos reais	[35]
Testes Funcionais	Teste de interação	[36]

A. Geração de dados de teste

A geração de dados de teste é o processo de identificação de conjuntos de dados de entrada válidos para um programa de acordo com os critérios de teste. Quanto maior e mais complexo um programa, mais difícil é gerar tais dados de entrada. Dessa forma, a utilização de técnicas automáticas para a geração destes conjuntos se mostra adequada. Além disso, esta atividade de geração apresenta, por exemplo, a meta de atingir a maior quantidade possível de testes no sistema com os dados gerados. Nesse aspecto, a utilização de metaheurísticas, enquanto técnicas de otimização, também se mostra adequada tendo em vista a impossibilidade de realizar tal atividade satisfatoriamente a não ser pelo uso de técnicas matemáticas. De outra forma, por exemplo, outra possível resolução seria utilizar técnicas aleatórias de geração. Entretanto, como já demonstrado [17], tal abordagem não é satisfatória.

Korel [18] propõe um gerador de dados de teste para o qual o programa a ser testado serve como a entrada. Dessa forma, o programa para o qual os conjuntos serão gerados é executado e sua execução é monitorada a fim de identificar os dados necessáros. O gráfico de controle de fluxo do programa é gerado e os caminhos possíveis são percorridos e, finalmente, os dados de teste de entrada necessários para percorrer tais caminhos possíveis são gerados. Este artigo tem a contribuição de ter indicado o potencial da aplicação de otimização no problema da geração de dados de teste. A formulação genérica do problema é apresentada na Figura 5, onde os termos 'n' indicam os pontos no código (branchs) e D é o domínio de dados.

$$\text{Seja } P = < n_{k_1}, n_{k_2}, \ldots, n_{k_q} > \text{ um caminho em um programa.}$$

$$\text{Encontrar uma entrada } x \in D \text{ para o programa, tal que } P \text{ seja satisfeito}$$

Fig. 5. Formalização do problema da geração de dados de teste.

Quando é percebido um caminho diferente do definido como controle, e por isso não desejado, uma função de minimização é executada para encontrar os valores dos dados de entrada correspondentes. O autor mostra que a técnica proposta apresentou melhoria significativa em relação aos demais métodos utilizados.

1) Geração de dados de teste por evolução

A metaheurística *Algoritmos Genéticos* também é empregada na geração de dados [17]. Uma das contribuições do trabalho é a utilização de tal metaheurística para este problema, já que antes apenas algoritmos de busca local como o *Hill-Climbing* tinham sido usados. A abordagem utilizada no trabalho, derivada da geração dinâmica de dados de teste, é definida pelo particionamento do programa em blocos de forma que cada bloco possa ser avaliado como uma função de acordo com os dados de entrada. O objetivo nesse caso é determinar quais blocos possuem o valor mínimo na execução de forma que todos os pontos do código sejam cobertos. Os resultados do trabalho mostraram que de fato a abordagem com *Algoritmos Genéticos* é mais eficiente que a geração aleatória, e que com o aumento da complexidade do programa sob teste existe aumento direto da melhoria apresentada pela abordagem.

2) Melhorias de algoritmos genéticos para o problema

Algoritmos Genéticos foram usados em outros trabalhos para a geração de dados de teste [19]. O objetivo foi a maximização da cobertura de caminhos de fluxo do sistema. Os autores adicionaram características que melhoraram o seu desempenho especificamente para este problema, como normalização da função objetivo . O trabalho indica que com o uso das técnicas adicionais foi possível determinar os dados de teste com menos gerações (iterações) da metaheurística. Além

disso, os resultados encontrados demonstraram a eficiência da técnica.

3) Abordagem simplificada com algoritmos genéticos

A metaheurística *Algoritmos Genéticos* foi usada em outro trabalho como base para a formalização de uma abordagem para resolver este problema [20]. No trabalho, o conceito de cromossomos e evolução são tomados dos *Algoritmos Genéticos* para a geração de dados de testes. A seguir, os "indivíduos" sofrem a operação de cruzamento para a geração de novos dados de teste. A avaliação da população é realizada considerando a quantidade de vezes que o indivíduo aparece no teste. Os autores mostram que os resultados encontrados com esta técnica foram semelhantes aos de outras abordagens.

4) Utilização de têmpera simulada

Outra metaheurística usada no problema da geração de dados de teste é a *Têmpera Simulada* [21]. O trabalho foi realizado em alguns tipos de problemas da geração de teste, como o de teste de reuso de componentes. Este problema trata da verificação de que o novo sistema onde o componente será reusado tem as caracteríticas necessárias definidas quando o componente foi desenvolvido. Para validar a abordagem, foram gerados sistemas com condições que não permitiriam a utilização dos componentes de forma segura (isto é, sem erros). Os resultados mostraram que a abordagem proposta foi capaz de gerar dados que identificassem as falhas do sistema que impedem a reutilização dos componentes. Além disso, a eficiência do método no problema foi constatada, pois o tempo de retorno da solução foi na ordem de segundos.

B. Seleção de casos de teste

A atividade de seleção de casos de teste consiste da escolha de quais testes devem ser realizados em um sistema, seja para a primeira versão ou para versões posteriores. No último caso, tem-se o chamado teste de regressão. Essa última situação ocorre após a realização de alguma modificação necessária no sistema, como uma manutenção, que possa gerar alterações em funcionalidades que já estavam em funcionamento. Neste caso, a abordagem ideal seria testar todos os casos de teste previamente utilizados, o que significaria testar todo o sistema novamente. A necessidade dessa solução deriva do fato de que a modificação pode ter ocasionado algum impacto em outras funcionalidades do código desenvolvido. Entretanto, geralmente não há tempo ou recursos suficientes para testar todo o sistema novamente, sendo necessário selecionar alguns casos de teste para execução.

1) Análise de técnicas para a seleção de casos de teste

Mansour et al. [22] comparam cinco algoritmos de seleção de casos de teste, a saber: Têmpera Simulada, *Reduction* [23], *Slicing* [24], *Dataflow* [25], e *Firewall* [22] em relação à atividade de seleção de casos de teste de regressão. A função de satisfação usada foi a cobertura de código obtida pelos casos de teste. A comparação foi baseada em oito critérios quantitativos e qualitativos: número de casos de teste, tempo de execução, precisão, inclusividade (o quanto a técnica seleciona casos que cobrem falhas na nova versão), processamento dos requisitos, tipo de manutenção, nível de teste e tipo de abordagem. Os resultados mostraram que as técnicas apresentam resultados melhores dependendo do critério. Em relação ao algoritmo Têmpera Simulada, os resultados indicaram boas soluções nos critérios de número de casos de teste e precisão.

2) Abordagem multiobjetiva em seleção de casos de teste

O conceito de Pareto da otimização multiobjetiva é introduzido neste problema em um trabalho de 2007 [26]. Como já apresentado, a utilização de otimização multiobjetiva significa que mais de uma função de satisfação é considerada no proceso de resolução. Duas formulações para o problema foram utilizadas: a primeira combina as funções de satisfação de cobertura de código e de custo de execução, e a segunda versão usa as funções de satisfação de cobertura de código, de custo de execução e de histórico de falhas:

Maximizar a cobertura do código *Minimizar o custo computacional*
Maximizar a cobertura do código *Minimizar o custo computacional* *Maximizar taxa de detecção de falhas*

Fig. 6. Formulações do problema da seleção de casos de teste.

Foram utilizados três algoritmos: uma reformulação de técnica de algoritmo guloso, o NSGA-II, e uma variação denominada vNSGA-II. Nos experimentos foram utilizados quatro programas menores da suíte Siemens (726, 570, 412 e 374 linhas de código) e o programa *space* que tem 9.564 linhas de código.

Para a primeira abordagem, a metaheurística NSGA-II obteve o melhor desempenho, encontrando toda a *frente de Pareto*, seguido do algoritmo guloso e do vNSGA-II. Porém, para o programa *space* a técnica gulosa obteve o melhor desempenho. O vNSGA-II apresentou resultados semelhantes ao NSGA, mas com resultados ainda bem inferiores aos do algoritmo guloso. Os resultados para a segunda abordagem de funções de otimização se mostraram semelhantes.

3) Seleção de casos de teste com informação dos clientes

Uma abordagem multiobjetiva diferente para o problema de seleção de casos de teste foi apresentada em um trabalho de 2009 [27]. No trabalho são consideradas mais características do contexto do problema, como o risco de cada caso de teste e a importância dada pelo cliente aos requisitos que o caso de teste cobre. Dessa forma, o trabalho apresenta como principal contribuição a consideração da participação da visão do cliente como uma dos aspectos na atividade de seleção dos casos de testes.

A formulação multiobjetiva proposta usa três funções de satisfação, a saber: minimização do tempo de execução, minimização do risco e maximização da importância. O artigo também apresentou restrições para o problema: o tempo disponível e a existência de pré-requisitos entre os casos de teste. Tal existência de pré-requisitos indica que alguns casos

de teste só podem ser executados após a execução de seus precedentes. A formulação matemática completa é apresentada na Figura 7.

$$1.\ Minimizar \sum_{j=1}^{k} TempoExecução(t_j)$$

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

$$3.\ Maximizar\ Importância(t_j)$$

Sujeito a:

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

$$b) \sum_{h=1}^{p} TempoDisponível(p_h) \leq T_{max}$$

$$c) \forall r_i \in R, \forall t_j \in T,$$

$$(\exists r \in R \ni r \in precedente(r_i)\ and\ cobertura(t_j, r)) \rightarrow t_j \in testCases(r_i)$$

Fig. 7. Formulação com participação da visão do cliente e restrições para o problema da seleção de casos de teste.

Para a resolução do problema foi utilizado o algoritmo NSGA-II, e como base de comparação também foi utilizado um algoritmo randômico. Para o NSGA-II, diferentes tamanhos de população foram considerados: 10, 20, 30, 40, 50. A partir de 50 os resultados se mostraram semelhantes. O NSGA-II gerou resultados bem melhores que o algoritmo randômico, como pode ser visto na figura a seguir. O algoritmo randômico gerou 50 soluções válidas, mas apenas duas delas foram melhores que as soluções geradas pelo NSGA-II. A figura 8 a seguir apresenta os resultados do trabalho. Nela é possível identificar a frente de Pareto encontrada pela metaheurística NSGA-II com soluções com alta importância, baixo tempo de execução e baixo risco; e as soluções aleatórias dispersas no campo de busca.

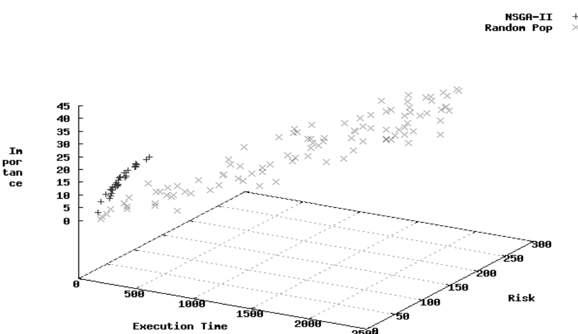

Fig. 8. Resultados de NSGA-II e técnica aleatória para a seleção de casos de teste - soluções do NSGA-II melhores que as geradas aleatoriamente [27].

4) Seleção segura de casos de teste

Rothermel e Harrold [28] apresentam uma abordagem denominada seleção de casos de teste de regressão segura. Este conceito trata da seleção dos casos de teste de forma que não seja permitida a exclusão de um caso de teste caso ele pudesse descobrir falhas na versão modificada do sistema. A principal contribuição do artigo é a validação da técnica com diversas instâncias de sistemas reais. Isso permitiu a realização de uma análise empírica dos resultados e a prova da eficiência da abordagem proposta.

C. Priorização de casos de teste

A atividade de priorização de casos de teste trata da ordenação dos casos de teste de modo que os mais significativos sejam executados primeiro, pois é possível que não exista tempo suficiente para executar todos. Em geral, a cobertura dos requisitos de teste é o principal objetivo a ser otimizado, ou seja, a ordenação dos casos de teste é feita de maneira que o número máximo de requisitos seja coberto no início. Dessa forma, mesmo na ocasião de necessidade de parada dos testes, seja por falta de tempo ou de recursos, pode-se garantir maior cobertura de teste no sistema. A formulação genérica desse problema é apresentada na Figura

T é uma suíte de testes (contém casos de teste)
PT é o conjunto de todas as permutações de T
f é uma função de PT aos números reais

Encontrar $T' \in PT$ tal que:
$$(\forall T''\ (T'' \in PT)(T'' \neq T')[f(T') \geq f(T'')].$$

9.

Fig. 9. Formalização do problema da priorização de casos de teste.

1) Formalização de funções de satisfação

Trabalhos iniciais [29] usaram algoritmos gulosos com o intuito de encontrar soluções ótimas. Para comparação foram usadas abordagens aleatórias, onde uma ordem randômica é definida, e de execução dos casos de teste sem ordenação, isto é, em que a execução segue a ordem de criação dos casos de teste. A contribuição do trabalho foi a formalização do problema e a definição das funções de satisfação como cobertura de sentenças no código, cobertura de sentenças não cobertas em testes anteriores e probabilidade de descoberta de falhas.

2) Considerando o tempo de execução para a priorização

Walcott et al. [30] propõem uma técnica de priorização de casos de teste utilizando Algoritmos Genéticos com o uso da cobertura de código gerada pela suíte de testes e do tempo para executá-la como parâmetros. Tal abordagem foi comparada com técnicas de ordem inicial, onde os casos de teste são executados na ordem em que foram escritos e ordem reversa, na qual a execução é na ordem contrária. O método proposto com Algoritmos Genéticos superou as técnicas gulosas e abordagens puramente aleatórias.

3) Relação da priorização com seleção de casos de teste

Yoo e Harman [26] descreveram uma abordagem utilizando o conceito de Pareto para priorização de casos de teste a partir do problema da seleção de casos de teste. Os objetivos utilizados foram a cobertura de código, o custo de execução e o histórico de detecção de falhas. O objetivo especificamente para a priorização foi encontrar um vetor de variáveis de decisão que representam a ordenação dos casos de teste. Neste caso, tal vetor maximiza um vetor de funções de satisfação. Outros algoritmos foram comparados com o algoritmo proposto, obtendo como resultado: o Algoritmo Genético superou o Algoritmo Guloso na ocasião de duas funções de satisfação, e nas demais situações o Algoritmo Guloso teve o melhor desempenho.

4) Comparação de técnicas para a priorização

Li et al. [31] compararam cinco algoritmos para priorização de casos de teste: Algoritmo Guloso, Algoritmo Guloso Adicional, 2-Ótimo Algoritmo, *Hill-Climbing* e Algoritmos Genéticos. Os autores separaram em 1.000 suítes pequenas (8 a 155 casos de teste) e 1.000 suítes maiores (228 a 4.350 casos de teste). Seis programas na linguagem C foram usados nos experimentos, cada um contendo entre 374 a 11.148 linhas de código. A análise foi realizada separadamente para os programas pequenos e os grandes. O objetivo analisado foi a métrica de cobertura percentual média de código. Nas suítes pequenas, para os programas pequenos, o algoritmo genético foi ligeiramente melhor que os demais, enquanto o algoritmo guloso obteve o pior desempenho. Porém, foi mostrado a partir de análise estatística que não há diferença significante entre os algoritmos genéticos e o guloso adicional. Para os programas grandes, não houve diferença significante entre os algoritmos guloso adicional e o 2-ótimo, ao invés de guloso adicional e algoritmo genético. A diferença do algoritmo genético para os dois melhores (guloso adicional e 2-ótimo) foi pouca, mas significante. Para as suítes grandes, os resultados foram semelhantes aos experimentos com suítes de teste pequenas. No caso de programas pequenos, não houve diferença significante entre o adicional guloso e o algoritmo genético, e para os programas grandes não houve diferença significante entre o guloso adicional e o 2-ótimo.

5) Utilização do GRASP reativo na priorização

A utilização da metaheurística GRASP Reativo foi investigada para o problema de priorização de casos de teste [32]. O trabalho realizou experimentos com vários percentuais diferentes das suítes de teste (1%, 2%, 3%, 5%, 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90% e 100%). O GRASP Reativo foi comparado com técnicas já reportadas na literatura, como algoritmo guloso e Algoritmo Genético. A análise dos resultados foi realizada tanto em relação ao desempenho quanto ao tempo de execução para o retorno da solução. Para fins de comparação, os dados utilizados nos experimentos foram os mesmo de trabalho anterior [31]. A função de satisfação utilizada foi a de cobertura de código. Validando o trabalho base, o algoritmo Guloso Adicional obteve o melhor desempenho. A metaheurística GRASP Reativo apresentou o segundo melhor desempenho, porém mostrou-se mais lenta que o algoritmo Adicional Guloso.

D. Testes não funcionais

Os testes não funcionais verificam características do sistema não relacionadas a funcionalidades, como tempo de execução, usabilidade, segurança, performance, *stress*, padrões da empresa (se foi desenvolvido conforme os padrões). Estas características não-funcionais são especificadas pelo cliente.

O teste de performance verifica se a aplicação atende aos requisitos de performance como, por exemplo, tempo para exibição de uma página, tempo de consulta para um relatório, número de acessos simultâneos à aplicação, dentre outros. O teste de *stress* verifica como a aplicação se comporta quando os parâmetros de performance são excedidos, a fim de conhecer o comportamento da aplicação em situações de maior carga ou "stress" e planejar as ações necessárias para que o impacto seja mínimo. Por exemplo, digamos que o cliente especificou que 20 pessoas poderiam acessar a aplicação simultaneamente. O teste de performance vai ser realizado da seguinte maneira: 20 acessos simultâneos serão realizados e, se a aplicação "responder" dentro do tempo esperado, está tudo funcionando corretamente. O teste de *stress* irá utilizar mais de 20 acessos simultâneos com o objetivo de verificar o que ocorre quando a aplicação é "estressada".

Outro fator considerado como característica não-funcional é o tempo de execução do sistema. Neste caso, o objetivo é identificar se o sistema responde à entrada com tempo muito curto (antes do desejado) ou muito longo (demora para processamento). Para isso, os testes não-funcionais focam na identificação de dados de entrada que possam chegar ao limite no tempo de resposta do sistema. O tempo é calculado geralmente em unidades de ciclo de procesador, que apesar de diferir entre sistemas, tem maior precisão de cálculo em relação a segundos. A diferença entre o teste de tempo de execução e o de performance é que no primeiro tentamos encontrar dados de entrada que gerem uma resposta no tempo desejado (tempo máximo especificado), e no segundo medimos o tempo de resposta para uma entrada qualquer gerada e o comparamos com o tempo máximo especificado.

1) Teste de performance

Binad et al. [33] propõem uma ferramenta chamada *Real Time Test Tool* (RTTT) desenvolvida utilizando-se o *Algoritmo Genético* para gerar casos de teste para teste de performance. Estes casos de teste possuem entradas que irão gerar *deadlines* nas tarefas que serão executadas. Como os autores salientam, esta não é uma tarefa simples de fazer manualmente. Na abordagem proposta, os testes são realizados a partir de um número de atividades e cada uma delas tem uma prioridade para execução. Estudos de caso foram realizados com o apoio da ferramenta desenvolvida. Para comparação entre os experimentos, tanto o número de tarefas quanto as prioridades destas foram alterados. Os resultados foram satisfatórios por oferecerem uma forma de execução automática, mas não foram comparados com outras abordagens.

2) Teste de tempo de execução

Wegener et al. [34] iniciaram a consideração de ciclos de processador no lugar de segundos como medida de tempo de execução. O artigo realizou testes com cinco sistemas de diferentes domínios para determinação da eficiência da metaheurística *Algoritmos Genéticos* em comparação à busca aleatória. Os resultados indicaram que de fato os AGs encontraram dados com tempos de resposta limites.

Em outro trabalho da área [35] os autores também realizaram experimento com a finalidade de comparar os resultados de *Algoritmos Genéticos* com busca aleatória. Os testes foram realizados com mais parâmetros para determinar a validação da técnica em contextos mais reais. Os resultados mostraram que os Algoritmos Genéticos foram capazes de encontrar dados com valores mais extremos.

E. Testes funcionais

Testes funcionais verificam se a implementação de uma aplicação está de acordo com sua especificação, ou seja, se todas as funcionalidades especificadas pelo cliente estão implementadas conforme a documentação, simulando a interação com o usuário.

1) Teste de interação

Os testes de interação foram considerados em um artigo de 2003 [36]. Os autores recombinaram várias técnicas para encontrar um conjunto maior de vetores de cobertura que podem ser expandidos ou diminuídos quando necessário. Estes vetores de cobertura com força variável são conjuntos de vetores de cobertura dentro de vetores de cobertura maiores e podem ser combinados para garantir uma cobertura maior para o sistema. Os autores utilizaram os algoritmos gulosos TCG (*Test Case Generator*) e AETG (*Automatic Efficient Test Generator*) e as metaheurísticas *Hill Climbing* e *Têmpera Simulada*. As metaheurísticas superaram significantemente os algoritmos TCG e AETG na busca pelo melhor *covering array*, sendo a *Têmpera Simulada* a melhor na maioria dos experimentos, sendo que, em alguns casos esta técnica encontrou a solução ótima.

V. Conclusões e Trabalhos Futuros

A aplicação de metaheurísticas em problemas da Engenharia de Software faz parte do campo relativamente novo denominado *Search-based Software Engineering* (SBSE). Neste contexto, a fase de Teste de Software apresentou um destaque sobre as outras fases de desenvolvimento de sistemas. Dessa forma foi criado o campo de pesquisa *Search-based Software Testing* (SBST). Os resultados obtidos nessa área indicam o potencial que tal recente campo de pesquisa apresenta. Nesse sentido, essa forma de visualização dos problemas da Engenharia de Software, e de Teste de Software da mesma forma, permite a resolução de problemas antes incapazes de serem resolvidos de forma satisfatória.

Os trabalhos descritos neste artigo apresentam a pesquisa em SBST em problemas de geração de dados de teste, seleção e priorização de casos de teste, e testes funcionais e não funcionais. A partir deste estudo de revisão da literatura pode-se verificar a importância e amplitude desse recente campo de pesquisa. Os trabalhos analisados demonstram a eficiência e relevância da aplicação de metaheurísticas em problemas de Teste de Software em comparação com abordagens aleatórias, por exemplo. Além disso, a validade de tal aplicação é mostrada na capacidade dos métodos na resolução dos problemas sob a forma de problemas de otimização.

Os trabalhos e formulações apresentadas neste estudo servem como motivação para a percepção de modificações nas formulações já apresentadas. Outro aspecto do levantamento realizado é a possibilidade de, a partir do apresentado, a modelagem de novos problemas da área de Teste de Software como problemas de otimização. Como trabalhos futuros indicamos a modelagem e resolução de problemas de diversas áreas da Engenharia de Software, em especial na área de Teste de Software. Especificamente, a resolução dos problemas relacionados à atividade de teste de regressão com o uso de metaheurísticas ainda não utilizadas.

Referências

[1] T. Dyba, “An empirical investigation of the key factors for success in software process improvement”, IEEE Transactions on Software Engineering, IEEE, May 2005, pp. 410-424.

[2] F. G. Freitas, C. L. B. Maia, D. P. Countinho, G. A. L. Campos, J. T. Souza, “Aplicação de Metaheurísticas em Problemas da Engenharia de Software: Revisão de Literatura”, II Congresso Tecnológico Infobrasil (Infobrasil 2009), 2009.

[3] F. Glover, “Future paths for integer programming and links to artificial intelligence”, Computer Operational Research, 13, pp. 533-549.

[4] S. Kirkpatrick, D. C. Gellat, M. P. Vecchi, “Optimizations by simulated annealing” Science v. 220, pp. 671-680, 1983.

[5] J. Holland, *Adaptation in Natural and Artificial Systems*, 1975.

[6] F. Glover, G. Kochenberger, *Handbook of Metaheuristics*, Springer, 1st edition, 2003.

[7] E. Talbi, *Metaheuristics: From Design to Implementation*, Wiley Publishing, 2009.

[8] C. Blum, A. Roli, “Metaheuristics in combinatorial optimization: Overview and conceptual analysis”, ACM Computing Surveys, Vol. 35, N 3, Setembro de 2003, pp. 268-308.

[9] T. A. Feo, M. G. C. Resende. “Greedy randomized adaptive search procedures”, Journal of Global Optimization, 6:109–133, 1995.

[10] M. Prais, C. C. Ribeiro, Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment, INFORMS Journal on Computing 12, 164-176, 2000.

[11] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, “A Fast Elitist Multi-Objective Genetic Algorithm: NSGA-II”, IEEE Transactions on Evolutionary Computation, V. 6, pp. 182-197, 2000.

[12] M. Harman, “Search-Based Software Engineering”, Workshop on Computational Science in Software Engineering, 2006.

[13] J. Clarke, *et al.* “Reformulating software engineering as a search problem”, IEE Proceedings Software, Vol. 150, No. 3, Junho de 2003, pp. 161-175.

[14] M. Harman, B. F. Jone, “The SEMINAL workshop: reformulating software engineering as a metaheuristic search problem”, ACM SIGSOFT Software Engineering Notes, Volume 26, Issue 6, Novembro de 2001, PP. 62-66.

[15] W. Miller, D. L. Spooner, “Automatic generation of floating-point test data”, IEEE Transactions on Software Engineering, IEEE, 1976, pp. 223-226.

[16] M. Harman, B. F. Jones, “Search-based software engineering”, Information and Software Technology, 2001, pp. 833-839.

[17] C. C. Michael, G. McGraw, M. Schatz, “Generating software test data by evolution”, Proceedings of IEEE Transactions on Software Engineering, Vol. 27, Number 12, 2001, pp. 1085-1110.

[18] B. Korel, “Automated software test data generation”, Proceedings of IEEE Transactions on Software Engineering, Vol. 16, Number 8, 1990, pp. 870-879.

[19] I. Hermadi, M. A. Ahmed, "Genetic algorithm based test data generator", The 2003 Congress on Evolutionary Computation, 2003.
[20] S. Khor, P. Grogono, "Using a genetic algorithm and formal concept analysis to generate branch coverage test data automatically", 19th International Conference on Automated Software Engineering, 2004, pp. 346-349.
[21] N. Tracey, J. Clark, K. Mander, J. McDermid, "An automated framework for structural test-data generation", Proceedings of the International Conference on Automated Software Engineering, pp. 285-288.
[22] N. Mandour, R. Bahsoon, G. Baradhi, "Empirical comparison of regression test selection algorithms", The Journal of Systems and Software 57, 2001, pp. 79-90.
[23] M. J. Harrold, R. Gupta, M. L Soffa, "A methodology for controlling the size of a test suite", ACM Transactions on Software Engineering and Methodology, Vol. 2, Issue 3, 1993, pp. 270-285.
[24] H. Agrawal, J. R. Horgan, E. W. Krauser, S. London, "Icremental regression testing", Conference on Software Maintenance, 1993, pp. 348-357.
[25] R. Gupta, M. J. Harrold, M. L. Soffa, "Program Slicing-Based regression testing techniques", Software Testing, Verification and Reliability, Vol 6, Number 2, 1996, pp. 83-111.
[26] S. Yoo, M. Harman, "Pareto efficient Multi-Objective test case selection", Proceedings of the International Symposium on Software Testing and Analysis, 2007, pp. 140-150.
[27] C. L. B. Maia, R. A. F. Carmo, F. G. Freitas, G. A. L. Campos, J. T. Souza, "A Multi-Objective approach for the regression test case selection problem", XLI Simpósio Brasileiro de Pesquisa Operacional, 2009.
[28] G. Rothermel, M. J. Harrold, "Empirical studies of a safe regression test selection technique", IEEE Transactions on Software Engineering, vol. 24, no. 6, Agosto de 1998, pp. 401-419.
[29] G. Rothermel, R. J. Untch, C. Chu, "Prioritizing test cases for regression rest", IEEE Transactions on Software Engineering, vol. 7, no. 10, 2001, pp. 929-948.
[30] K. Walcott, M. L. Soffa, G. M. Kapfhammer, R. S. Roos, "Time-Aware test suite prioritization", Proceedings of the International Symposium on Software Testing and Analysis, 2006, pp. 1-12.
[31] Z. Li, M. Harman, R. M. Hierons, "Search algorithms for regression test case prioritization", IEEE Transactions on Software Engineering, vol. 33, no. 4, Abril de 2007, pp. 225-237.
[32] C. L. B. Maia, R. A. F. Carmo, F. G. Freitas, G. A. L. Campos, J. T. Souza, "Automated test case prioritization with reactive GRASP", Advances in Software Engineering, vol. 2010, 2010, pp. 29-46.
[33] L. C. Brinad, Y. Labiche, M. Shousha, "Performance stress testing of Real-Time systems using genetic algorithms", Technical Report, Carleton University, 2004.
[34] J. Wegener, H. Sthamer, B. F. Jones, D. E. Eyres, "Testing real-time systems using genetic algorithms", Software Quality Control 6 (2) 127–135, 1997.
[35] J. Wegener, M. Grochtmann, "Verifying timing constraints of real-time systems by means of evolutionary testing", Real-Time Systems 15 (3) (1998) 275–298.
[36] M. B. Cohen, P. B. Gibbons, W. B. Mugridge, C. J. Colbourn, "Constructing test suites for interaction testing", Proceedings of the 25th International Conference on Software Engineering, IEEE Computer Society, 2003, pp. 38–48.