O Algoritmo Genético Coevolucionário para Redução de Subconjuntos de Casos de Teste da Análise de Mutantes

André Assis Lôbo de Oliveira[1], Beatriz Proto Martins[1], Celso G. Camilo-Junior[1], Auri M. Rizzo Vincenzi[1]

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

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

Abstract. *The Coevolutionary Genetic Algorithm (CGA), used in Mutation Analysis, is responsible for selecting, concurrently, mutants programs and test cases subsets with high mutation score and low cost. However, this algorithm was not evaluated from the perspective of minimizing the selected subsets, a factor that can reduce even more costs in Mutation Analysis. Thus, this research aims to evaluate the AGC by applying test cases subsets minimization. The AGC was compared with another minimization technique in the literature on four benchmarks. The results shows that the AGC selects minimized test cases subsets, or very close to this, on the analyzed scenarios.*

Resumo. *O Algoritmo Genético Coevolucionário (AGC), utilizado na Análise de Mutantes, é responsável por selecionar, concomitantemente, subconjuntos de programas mutantes e casos de teste com alto escore de mutação e baixo custo. Todavia, tal algoritmo não foi avaliado sob a perspectiva de minimização dos subconjuntos selecionados, fator que pode reduzir ainda mais o custo da Análise de Mutantes. Tendo isso em vista, o presente trabalho objetiva avaliar o AGC aplicando-se a minimização de subconjuntos de casos de teste. O AGC foi comparado com outra técnica de minimização da literatura sobre quatro benchmarks. Os resultados revelam que o AGC seleciona subconjuntos de casos de teste mínimos, ou bem próximos, nos cenários analisados.*

1. Introdução

Durante o ciclo de vida de um software, uma grande parcela dos custos é gasta na fase de testes. Com o objetivo de minimizá-los, surge a *Search-Based Software Testing* (*SBST*) [McMinn 2011], uma abordagem que aplica metaheurísticas para otimizar soluções de problemas em Teste de Software. Entre as metaheurísticas, destacam-se os Algoritmos Genéticos (AGs), uma das técnicas mais eficientes em problemas de otimização combinatória [De Jong 2006] com alta complexidade de busca.

Dentre as várias técnicas e critérios do Teste de Software, este trabalho aborda o Teste de Mutação (TM), um critério de teste conhecido por sua grande eficácia em detectar defeitos. No entanto, o alto custo computacional, proveniente da grande quantidade de programas mutantes gerados, torna o TM pouco utilizado na prática.

Técnicas da SBST têm sido aplicadas para diminuir os custos do Teste de Mutação. O Algoritmo Genético é uma das metaheurísticas mais utilizadas por ser facilmente adaptado aos problemas e por gerar bons resultados. Uma de suas adaptações é o Algoritmo Genético Coevolucionário (AGC) [Oliveira, Camilo-Junior e Vincenzi 2013-a], o qual utiliza coevolução para selecionar bons subconjuntos de casos de teste e

mutantes, a fim de reduzir custos sem diminuir a eficácia. Todavia, o AGC não verifica se a seleção empregada possui a capacidade de minimizar os subconjuntos selecionados.

Neste contexto, este trabalho visa avaliar o AGC sob a perspectiva do tamanho dos subconjuntos de casos de teste selecionados. Para isso, o AGC é aplicado em 4 *benchmarks* conhecidos e a qualidade dos subconjuntos selecionados é comparada com as seleções realizadas por outra técnica de minimização. Os resultados revelam que o AGC possui a característica de selecionar subconjuntos reduzidos, o que encoraja a continuação dos estudos sob tal perspectiva.

2. Revisão Bibliográfica

2.1 Teste de Mutação

A Análise de Mutantes ou Teste de Mutação (TM) no nível de unidade, surgiu na década de 1970. DeMillo, Lipton, e Sayward (1978) apresentam a ideia da técnica que está fundamentada na hipótese do programador competente e no efeito de acoplamento. A hipótese do programador competente assume que programadores experientes escrevem programas muitos próximos de estarem corretos. Assumindo a validade dessa hipótese, pode-se dizer que os defeitos são introduzidos no programa por meio de pequenos desvios sintáticos, que embora não causem erros sintáticos, alteram a semântica do programa. O efeito de acoplamento, por sua vez, assume que defeitos complexos estão relacionados a defeitos simples. Deste modo, a detecção de um defeito simples pode levar a descoberta de defeitos complexos.

A partir de um programa original *P*, é criado um conjunto *P'* de programas modificados (mutantes) com o objetivo de avaliar o quanto um conjunto de teste *T* é adequado [DeMillo, Lipton e Sayward 1978]. Os casos de testes de T são executados sobre o programa original e o mutante, caso as saídas sejam diferentes o mutante é dito estar *morto*, dessa forma é possível que o programa em teste não contenha o defeito representado pelo mutante. Um problema enfrentado em TM é a geração de uma grande quantidade de mutantes traduzindo-se em um alto custo computacional para sua realização. Outro problema, consiste na geração de mutantes equivalentes que são sintaticamente diferentes do programa original, mas que são semanticamente iguais, não sendo mortos por qualquer caso de teste. Verificar a equivalência entre dois programas exige esforço humano por ser um problema indecidível [Jia and Harman 2011].

A métrica que define a adequabilidade de um conjunto de teste chama-se *escore de mutação*. De acordo com DeMillo, Lipton, e Sayward (1978), o *escore de mutação* é um número real que varia entre 0 e 1, calculado conforme a Equação 1. Quanto maior o *escore de mutação* de um caso de teste, maior a sua capacidade em matar mutantes.

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

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

2.2 Minimização de Conjuntos de Casos de Teste

Técnicas de minimização objetivam reduzir o tamanho do conjunto de teste pela

eliminação de casos de teste redudantes. A minimização pode também ser chamada de "redução do conjunto de teste" (*test suite reduction*) [Yoo e Harman 2010].

Técnicas de minimização de subconjuntos de teste são desejáveis no Teste de Regressão em que se objetiva estabelecer um subconjunto de casos de teste que seja capaz de testar as diferentes versões de um programa. A ideia é garantir que as novas características inseridas não influenciam no bom funcionamento das existentes. No contexto do TM, as técnicas de minimização de conjuntos são desejáveis para estabelecer um subconjunto pequeno que diminua o custo da execução dos casos de teste sobre os mutantes, mas com alto escore de mutação.

Um critério de teste é satisfeito quando todos os requisitos de teste são satisfeitos por quaisquer casos de teste. Para maximizar o efeito da minimização, T', um subconjunto de T, deve ser o conjunto mínimo. Todavia, atingir esse conjunto mínimo é um problema NP-completo [Yoo e Harman 2010], uma vez que é similar ao problema de cobertura do conjunto mínimo [Garey e Johnson 1979].

2.3 Trabalhos Correlatos

"*A NP-completude do problema de minimização de subconjuntos encoraja a aplicação de heurísticas*" [Yoo e Harman 2010]. Vale citar as heurísticas GE e GRE [Chen e Lau 1996]:

- *Heurística GE*: a) seleciona todos os casos de teste essenciais do conjunto; b) para os requisitos de teste remanescentes, usa um algoritmo guloso adicional que seleciona o caso de teste que satisfaz o máximo de requisitos não satisfeitos;
- *Heurística GRE*: a) remove todos os casos de teste redudantes no conjunto de teste e partir disso forma o conjunto essencial de casos de teste; b) utiliza a heurística GE para reduzir esse conjunto formado.

Metaheurísticas também são utilizadas nesse contexto de minimização. Nachiyappan, Vimaladevi e SelvaLakshmi (2010), propuseram um algoritmo genético que se baseia em informações de cobertura. Todavia, para construir o subconjunto, a metaheurística considera informações de tempo de execução de aplicações anteriores dos casos de teste, em um cenário de Teste de Regressão.

No contexto do Teste de Mutação, Polo, Piattini e García-Rodríguez (2008), utilizam mutantes de segunda ordem para reduzir o custo da execução de mutantes. A partir dessa classe de mutantes, forma-se o subconjunto mínimo de casos de teste por meio do seguinte algoritmo guloso: a) coloca-se dentro de T'o caso de teste que mata o maior número de mutantes; b) remove-se todos os mutantes já mortos por T'; c) seleciona-se o caso de teste seguinte que mata o maior número de mutantes; d) remove-se todos os mutantes já mortos por T'. Esses passos são repetidos até que não haja mutantes vivos. Obviamente, tais passos são realizados somente com os mutantes não-equivalentes.

Polo et al. (2008) disponibilizaram todo o material da experimentação utilizado na pequisa (programas e conjuntos de teste)[4]. Tal material pode servir como *benchmark* para técnicas de minimização e seleção, uma vez que se pode comparar o subconjunto selecionado com os subconjuntos ótimos (T'mínimo com maior escore de mutação) encontrados pela abordagem por eles proposta.

4 Disponível em: <http://www.inf-cr.uclm.es/www/mpolo/stvr/>

O AGC, proposto por Oliveira, Camilo-Junior e Vincenzi (2013-a), idem (2013-b), aborda a coevolução de maneira competitiva para selecionar subconjuntos de casos de teste e mutantes afim de reduzir custos sem diminuir a eficácia. Na seleção de casos de teste, o AGC faz uso da Classificação Genética, o que provê alto escore de mutação ao favorecer a seleção de *genes efetivos*, isto é, casos de testes que contribuem para o subconjunto. Percebe-se que a Classificação Genética evita casos de teste redundantes da mesma forma que as heurísticas para minimização de subconjuntos, portanto, pode ser empregado para obter subconjuntos de casos de teste minimizados.

3. Abordagem Proposta

A abordagem proposta centra-se na aplicação do AGC para minimização de subconjuntos de casos de teste. O objetivo consiste em verificar se o AGC consegue selecionar subconjuntos de casos de teste mínimos com escore de mutação máximos, por meio de sua busca coevolucionária entre casos de teste e programas mutantes.

Foi desenvolvida uma metodologia para proporcionar tal verificação, conforme descrita na Figura 1.

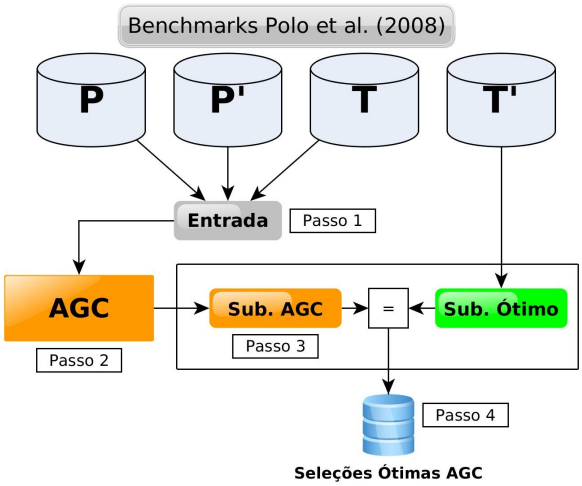

Figura 1. Avaliação dos subconjuntos de casos de teste selecionados pelo AGC

A avaliação utiliza os *benchmarks* fornecidos por Polo et al (2008) pelo fato de disponibilizarem os subconjuntos ótimos (*T'* com escore máximo e tamanho mínimo), os quais servem como base de comparação. Os *benchmarks* são constituídos por quatro elementos: a) programa original *P*; b) conjunto de mutantes *P'*; c) conjunto de casos de teste *T e*; d) subconjunto ótimo de caso de teste *T'*.

O tamanho do subconjunto de casos de teste (tamanho do indivíduo no contexto de algoritmo genético) é um parâmetro de entrada do AGC. A ideia é que ele receba o parâmetro de tamanho de subconjunto e configure os indivíduos da população de casos de teste. Ao final da busca coevolucionária ele deve fornecer um subconjunto de casos de teste (indivíduo) com um tamanho menor do que o dado, ou seja, minimizado.

Dessa forma, o primeiro passo da abordagem de avaliação, consiste em fornecer a entrada ao AGC, constituída pelos conjuntos *P*, *P'* e *T*. O segundo passo consiste na busca evolucionária empregada pelo AGC para selecionar o melhor subconjunto (melhor indivíduo) de casos de teste (Sub. AGC). O terceiro passo é a verificar se a saída do AGC é igual ao subconjunto ótimo (Sub. Ótimo) do respectivo *benchmark* em teste, ou seja, se o escore de mutação é igual e se os tamanhos dos subconjuntos são

iguais. O quarto, e último passo, consiste em armazenar a melhor seleção do AGC caso ela seja ótima para extração de métricas estatísticas.

4. Estudos Experimentais

4.1 Programas Utilizados

São considerados os quatro seguintes *benchmarks* dos utilizados por Polo et al. (2008), descritos, ainda, na Tabela 1:

1. Bubcorrecto (Bub): ordena um vetor através do método *bubble-sort*.
2. Fourballs: calcula o peso relativo de um corpo;
3. Mid: calcula o valor central a partir de três valores;
4. Trytip: realiza a classificação de triângulos, dadas as dimensões dos seus lados.

Tabela 1. Informações dos *benchmarks* utilizados

Nome	Total Mutantes	Total Equivalentes	Total Casos de Teste
Bub	80	21	256
Fourballs	212	44	96
Mid	181	43	125
Trytip	309	70	216

De acordo com as informações da Tabela 2, foram realizados dois experimentos com os tamanhos dos subconjuntos de casos de teste variados para cada *benchmark*, sendo eles maiores do que os tamanhos mínimos possíveis. A intenção é verificar se o melhor subconjunto selecionado pelo AGC é igual em escore e em tamanho em relação ao tamanho ótimo do respectivo *benchmark*.

4.2 Parâmetros dos algoritmos

Para todos os algoritmos foram fixados os seguintes parâmetros: a) operador de seleção: torneio, com 2 competidores; b) taxa de cruzamento: 95%; c) taxa de mutação: 5%; d) elitismo: 1 indivíduo; e) tamanho das populações de casos de teste e de mutantes: 4; f) quantidade de gerações: 50. O tamanho de indivíduo da população de mutantes foi fixado em 2,5% do conjunto total mutantes, configurando um tamanho comum a todos os *benchmarks* e pequeno em relação ao conjunto total de mutantes.

Tabela 2. Informações dos Experimentos

Nome	Nº de casos de teste (Conj. Reduzido)	Experimento 1: Tam. do Indivíduo (sub. de casos de teste)	Experimento 2: Tam. do Indivíduo (sub. de casos de teste)
Bub	1	2	3
Fourballs	5	10	15
Mid	5	10	15
Trytip	17	34	51

4.3 Discussão dos Resultados

Todos os resultados foram obtidos por meio de 30 execuções do AGC. A Tabela 3 apresenta os resultados dos experimentos.

Tabela 3. Resultados experimentais

Experimento 1					
Benchmark	**MAX**	**MED**	**MEDTAM**	**TSMIN**	**IMTAM**
Bub	1	0.9897	1.53	1	99.40%
Fourballs	1	0.9998	4.97	5	94.83%
Mid	1	0.9961	6.07	5	95.15%
Trytip	1	0.9993	16.90	17	92.18%
Experimento 2					
Benchmark	**MAX**	**MED**	**MEDTAM**	**TSMIN**	**IMTAM**
Bub	1	0.9997	1.30	1	99.49%
Fourballs	1	1	5	5	94.79%
Mid	1	0.9976	5.70	5	95.44%
Trytip	1	0.9992	16.87	17	92.19%

Legendas da Tabela 3: a) MAX: escore máximo alcançado; b) MED: média dos escores de mutação; c) MEDTAM: média dos tamanhos dos subconjuntos de casos de teste selecionados; d) TSMIN: tamanho do subconjunto mínimo do *benchmark* e) IMTAM: percentual de impacto da minimização no conjunto (percentual de redução em relação ao conjunto total) [Yoo e Harman 2010], calculado conforme Equação 2.

$$IMTAM = \left(1 - \frac{tamanhosubconjuntodecasosdeteste}{tamanhodoconjuntototaldecasosdeteste}\right) * 100 \qquad (2)$$

Conforme Tabela 3, o AGC obtém bons escore de mutação. Além disso, percebe-se que, em média, o AGC seleciona indivíduos com tamanhos muito próximos ao do subconjunto ótimo do benchmark. O experimento 1 forma inicialmente indivíduos menores do que no experimento 2. Todavia, para os *benchmarks* Bub e Trytip, o AGC seleciona indivíduos maiores no experimento 1 do que no experimento 2, evidenciando que esse aumento de tamanho não prejudicou a busca do AGC. Já para o *benchmark* Fourballs obteve melhores resultados, em termos de escore de mutação e tamanho, no experimento 2, alcançando a solução ótima nas 30 execuções. Da mesma forma, no *benchmark* Mid, esse aumento também melhorou os resultados do AGC.

As Figuras 2 e 3, colocam em paralelo a quantidade de soluções ótimas encontradas pelo AGC por *benchmark*, com a quantidade atingida de escores máximos e de subconjuntos de tamanhos mínimos encontrados, para os experimentos 1 e 2, respectivamente.

Conforme Figuras 1 e 2, encontrar soluções com escore de mutação máximo, não significa encontrar subconjuntos com tamanhos reduzidos. Percebe-se também que o aumento do tamanho do subconjunto de casos de teste não implicou em um aumento do tamanho do subconjunto selecionado pelo AGC. Isso traz indícios de que, mesmo com subconjuntos de tamanhos maiores, o AGC pode ser capaz de encontrar subconjuntos de tamanho mínimo.

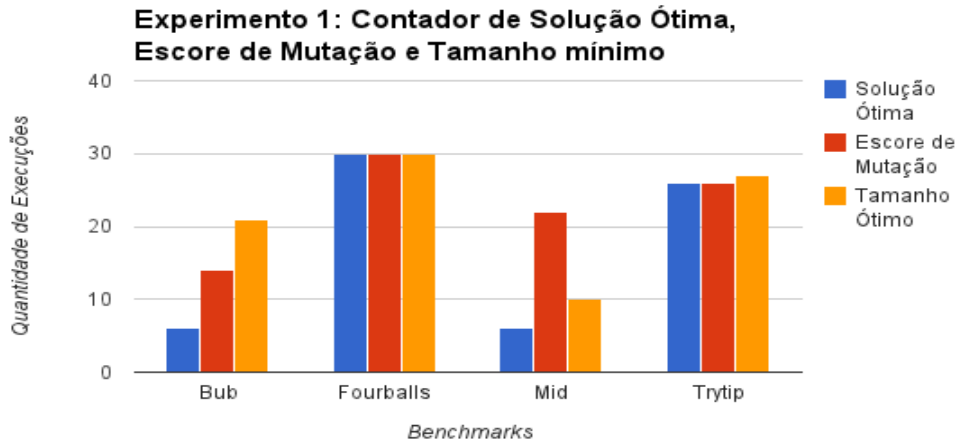

Figura 2. Experimento 1: Quantitativo de soluções ótimas encontradas

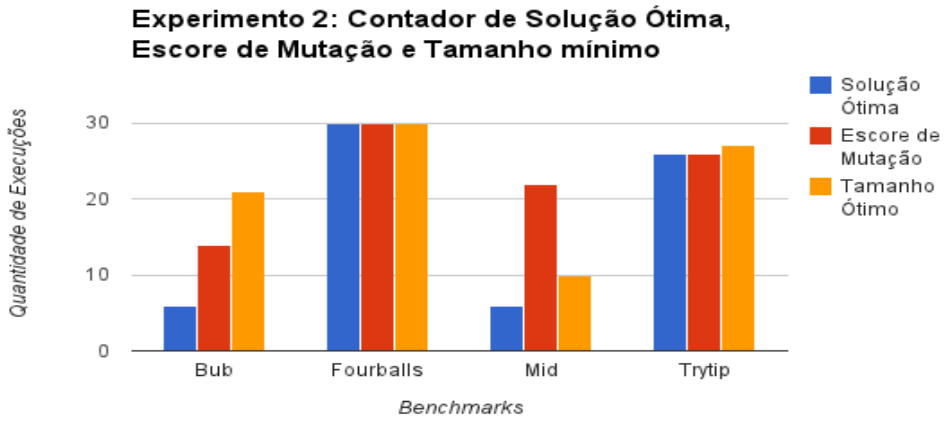

Figura 3. Experimento 2: Quantitativo de soluções ótimas encontradas

5. Conclusões e Trabalhos Futuros

Nesta pesquisa o AGC foi avaliado sob a perspectiva de minimização de subconjuntos de casos de teste aplicado a benchmarks reais. Os resultados revelam um bom desempenho do AGC na obtenção dos subconjuntos de tamanhos reduzidos ao se utilizar a Classificação Genética. Diante disso, acredita-se que o AGC constitui-se numa abordagem promissora também no campo da minimização de subconjuntos de casos de teste como contribuição à redução de custos do Teste de Mutação.

Como trabalhos futuros, pretende-se: i) comparar com outras metaheurísticas e técnicas de minimização para extrair maiores conclusões dessa abordagem; ii) aplicar o AGC em número maior de *benchmarks* e iii) aprimorar o AGC para que ele seja capaz de se auto-parametrizar para descobrir o melhor tamanho de subconjunto de casos de teste para o programa que se está sendo testado.

Referências

P. McMinn (2011) “Search-Based Software Testing: past, present and future. In: International Conference on Software Testing, Verication and Validation Workshops (ICSTW) p. 153-163.

K. A. De Jong (2006) “Evolutionary Computation: a unified approach”, MIT Press.

A. Oliveira, C. Camilo-Junior and Vincenzi (2013-a) “A Coevolutionary Algorithm to

Automatic Test Case Selection and Mutant in Mutation Testing". In: IEEE Congress on Evolutionary Computation (CEC) .

A. Oliveira, C. Camilo-Junior e A. Vincenzi (2013-b) "Um Algoritmo Genético Coevolucionário com Classificação Genética Controlada aplicado ao Teste de Mutação". IV Workshop de Engenharia de Software Baseada em Busca (WESB), 2013, Brasília.

Chen TY, Lau MF. Dividing strategies for the optimization of a test suite. Information Processing Letters, 60(3):135–141, 1996.

Garey MR, Johnson DS. Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Company: New York, NY, 1979.

M. Polo, M. Piattini and I. García-Rodríguez. Decreasing the Cost of Mutation Testing with Second-Order Mutants,&rdquo, *Software Testing, Verification, and Reliability*, vol. 19, no. 2, pp. 111-131, 2008.

S. Yoo an M. Harman. Regression testing minimization, selection and priorization: a survey. Software Testing, Verification and Reability, 1(1): 121-141, March 2010.

S. Nachiyappan, A. Vimaladevi, C.B. SelvaLakshmi. An Evolutionary Algorithm for Regression Test Suite Reduction, in proceedings of the International Conference on Communication and Computational Intelligence, India.27-29 D ecember, 2010, pp. 503-508.