O Teste de Mutação apoiado pelo Algoritmo Genético Coevolucionário com Classificação Genética Controlada

André Assis Lôbo de Oliveira [1]
Celso G. Camilo-Junior [1]
Auri Marcelo Rizzo Vincenzi [1]

Resumo: Este artigo situa-se no campo dos algoritmos genéticos coevolucionários que objetivam a seleção de bons subconjuntos de casos de teste e mutantes, no contexto do Teste de Mutação. Desse campo de estudo, selecionaram-se e avaliaram-se duas abordagens existentes. Tais avaliações, subsidiaram o desenvolvimento de um novo Algoritmo Coevolucionário com Classificação Genética Controlada. Para analisar a abordagem, 164 experimentos foram realizados comparando os resultados do algoritmo proposto com outros três métodos aplicados em quatro benchmarks reais. Os resultados revelam uma melhora significativa do $AGC - CGC$ sobre as outras abordagens quando se considera o aumento do escore de mutação sem aumentar acentuadamente o tempo de execução.

Abstract: This paper is situated in the field of genetic algorithms that aim coevolutionary selection of good subsets of test cases and mutants in the context of Mutation Testing. It was selected and evaluated two approaches from this field of study. Such evaluation supported the development of a new Coevolutionary Genetic Algorithm with Controled Genetic Classification. To analyse the approach, 164 experiments were performed comparing the results of proposed algorithm with three other methods applied to four real benchmarks. The results shows a significant improvement of $AGC - GCC$ compared to other approaches, due to the increase in mutation score without significative increase in runtime.

1 Introdução

Atividades de Validação e Verificação (V&V) consomem cerca de 50% à 60% do custo total no ciclo de vida de um software [1], sendo o teste de software uma das atividades mais utilizadas. Neste contexto, surge a *Search-Based Software Testing* (SBST) [2], uma abordagem de pesquisa que utiliza as metaheurísticas como técnicas de otimização para resolver alguns dos problemas do Teste de Software com o objetivo de minimizar tais custos. Dentre as metaheurísticas da SBST, destacam-se os Algoritmos Genéticos (AGs), uma das

[1]Instituto de Informática (INF), UFG, Câmpus Samambaia, Caixa Postal 131 - CEP 74001-970 - Goiânia - GO
{andreoliveira,celso,auri}@inf.ufg.br

técnicas mais eficientes e com maior uso em problemas de otimização combinatória [3] com alta complexidade de busca.

A atividade de teste pode ser classificada em: Teste de Unidade, Teste de Integração, Teste de Validação e Teste de Sistema [4]. Além destas, há o Teste de Regressão, que objetiva identificar defeitos em versões subsequentes à primeira do Sistema em Teste (*System Under Test* - SUT). Para tal, usa-se os casos de testes definidos inicialmente ou atualizações. No entanto, fatores como tamanho do sistema, recursos disponíveis e tempo disponível para o teste, desfavorecem a aplicação de um conjunto grande de casos de teste definidos inicialmente. Assim, encontrar um bom conjunto de casos de teste tem grande importância para melhor aplicar o teste, especialmente no contexto do Teste de Regressão.

Para avaliar o desempenho dos casos de teste, deve-se usar um critério de teste. Nesse trabalho, adotou-se o Teste de Mutação (TM) por ser uma técnica eficaz em revelar de defeitos [5]. No entanto, o alto custo computacional, proveniente da grande quantidade de programas mutantes gerados, torna o TM pouco utilizado na prática. Outro agravante consiste na existência de mutantes equivalentes dentre os mutantes gerados. Esses são indesejáveis porque não contribuem para evolução de conjunto de teste, além de exigir grande esforço humano para sua classificação como equivalentes. Tal esforço se dá porque a verificação automática da equivalência entre programas é um problema indecidível [5].

Em um de seus trabalhos, Offutt e Hayes [6] afirmam que existe uma classe de mutantes (*hard-to-kill*) que pode conduzir a bons casos de teste, de forma que se forem obtidos o subconjunto de casos de teste que matam tal classe de mutantes, consequentemente, obtem-se um subconjunto que matam todos os mutantes gerados.

Nesse contexto, percebe-se uma forte correlação entre os casos de teste e os programas mutantes, no TM. De um lado, objetiva-se selecionar o subconjunto de casos de teste com maior capacidade em matar mutantes. Do outro, objetiva-se selecionar o subconjunto de mutantes com maior capacidade de evitar serem mortos pelos casos de teste, mutantes *hard-to-kill*. Tal competição percebida no contexto do TM é análoga à coevolução das espécies que ocorre na natureza.

Desta forma, ressaltam-se os desafios gerais que são abordados nesse trabalho: a necessidade de selecionar um conjunto pequeno e bom de casos de teste para melhor executar os testes, e a necessidade de diminuir o custo computacional do TM, por meio da redução do conjunto de mutantes. Além desses, vale destacar o seguinte objetivo: controlar o emprego da *Classificação Genética* para tornar mais eficiente a seleção de bons subconjuntos de casos de teste para ganhar em eficácia (escore de mutação) sem perder em eficiência (tempo de execução) na seleção dos subconjuntos.

Diante disso, este artigo propõe um Algoritmo Genético Coevolucionário com Classificação Genética Controlada ($AGC - CGC$)[2] para encontrar bons subconjuntos de teste (casos de teste e mutantes) com baixo custo computacional.

Foram realizados 164 experimentos com diferentes benchmarks para avaliar: o desempenho de diferentes algoritmos coevolutivos considerando diferentes benchmarks (aplicações reais); a eficácia (escore de mutação) e a eficiência (tempo de execução) das abordagens nas soluções obtidas e; a influência nos escores de mutação e tempo de execução obtidos por meio da variação das Taxas de Classificação Genética (TCG).

Os resultados revelam que o algoritmo $AGC - CGC$ obtém ganhos significativos em termos de escore de mutação e em tempo de execução, em relação às outras abordagens da literatura.

Este artigo está estruturado da seguinte forma: a Seção 2 apresenta uma breve revisão sobre TM; os trabalhos correlatos são apresentados na Seção 3; a caracterização do problema a ser resolvido é descrita na Seção 4. Já a Seção 5 apresenta a abordagem proposta; a Seção 6 descreve a configuração dos experimentos; a análise dos resultados é apresentada na Seção 7 e; por fim as conclusões e os trabalhos futuros são apresentados na Seção 8.

2 Fundamentos

A presente seção se dedica a uma breve fundamentação teórica do presente artigo. Alguns conceitos, problemas de aplicabilidade e pontos importantes sobre o Teste de Mutação são apresentados.

2.1 Teste de Mutação (TM)

A Análise de Mutantes (Mutation Analysis, MA), nome dado ao Teste de Mutação (TM) no nível de unidade, surgiu na década de 1970 na Yale University e Georgia Institute of Technology. Um dos primeiros artigos publicados sobre o TM foi o de Demillo, Lipton e Sayward [8] apresentando 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,

[2] O Algoritmo Genético Coevolucionário com Classificação Genética Controlada ($AGC - CGC$) foi primeiramente proposto no IV Workshop de Engenharia de Software Baseada em Busca (WESB) [7]. O presente artigo trata-se de um extensão do artigo publicado no WESB 2013 merecendo um destaque para as seguintes contribuições: a) a realização de um número maior de experimentos, considerando uma maior variação de parâmetros e benchmarks e; b) um estudo realizado sobre a influência de diferentes Taxas de Classificação Genética (TCGs) nos escores de mutação e tempos de execução alcançados pelo $AGC - CGC$.

pode-se dizer que os defeitos são introduzidos no programa por meio de pequenos desvios sintáticos, que alteram a semântica do programa, muito embora não causem erros sintáticos.

O efeito de acoplamento assume que defeitos complexos estão relacionados à defeitos simples. Deste modo, a detecção de um defeito simples pode levar a descoberta de defeitos complexos.

No TM, a partir de um programa original *P*, um conjunto *P'* de programas modificados por operadores de mutação inserem pequenos desvios sintáticos no programa (código fonte ou objeto). O objetivo consiste em avaliar o quanto um conjunto de teste *T* é adequado [8]. O caso de teste executa sobre o programa original e o mutante, caso as saídas sejam diferentes o mutante é dito estar *morto*.

Um problema enfrentado no TM é a geração de uma grande quantidade de programas mutantes. Na prática, isso implica em um alto custo computacional para sua realização. Outro problema, consiste na geração de mutantes equivalentes. Embora, sintaticamente, um mutante equivalente seja diferente do programa original, ambos são semanticamente iguais. Isso significa que um equivalente não pode ser morto por nenhum caso de teste. Por isso, a verificação da equivalência entre dois programas exige esforço humano por ser um problema computacionalmente indecidível [5].

Dá-se o nome de *escore de mutação* a métrica que calcula a adequação de um conjunto de teste. De acordo com Demillo, Lipton e Sayward [8], o *escore de mutação* é um número real que varia entre 0.0 e 1.0, calculado conforme a Equação 1. Portanto, quanto maior o *escore de mutação* de um conjunto de teste, maior a sua capacidade em matar mutantes, ou seja, demonstrar que o programa em teste não contém o defeito representado pelo mutante.

$$MS(P,T) = \frac{DM(P,T)}{M(P) - EM(P)} \tag{1}$$

Onde:

- $DM(P,T)$: número de mutantes mortos pelo conjunto de teste T;
- $M(P)$: número total de mutantes gerados;
- $EM(P)$: número de mutantes gerados equivalentes a P.

Dado um programa P e um conjunto de teste T cuja a qualidade se deseja avaliar, o TM pode ser aplicado pelos seguintes passos [8]:

1. Geração do conjunto de mutantes P'

2. Execução de P com T;

3. Execução do conjunto P' com T;

4. Análise dos Mutantes Vivos.

2.1.1 Aplicabilidade Conforme já descrito, apesar do TM possibilitar uma boa avaliação de um conjunto, ele ainda apresenta problemas acerca de sua aplicabilidade em função do grande número de mutantes gerados. Por isso, muitas pesquisas objetivam a redução dos mutantes gerados ([9], [10], [11], [12], [13], [14] e [5]).

Além disso, muitas vezes é necessário aumentar a quantidade de casos de teste para se obter um melhor escore de mutação, fator que também aumenta o custo computacional. Por isso, a redução destes é também um objeto de pesquisa [5].

Dessa forma, o terceiro passso para aplicação do TM, geralmente, considera alta cardinalidade dos conjuntos de casos de teste (*T*) e mutantes (*P'*), configurando um alto custo computacional para sua realização [5].

Outro problema enfrentado é o grande esforço humano exigido na verificação da equivalência entre programas, pois a verificação automática da equivalência entre programas consiste em geral, numa questão indecidível. Dessa forma, esse processo exige esforço humano [5].

Contudo, pode-se destacar alguns pontos de grande importância para melhoria da aplicabilidade do TM:

1. Evitar mutantes equivalentes;

2. Selecionar um subconjunto reduzido de mutantes que conduzam a bons casos de teste;

3. Selecionar um subconjunto casos de teste que seja eficaz em matar mutantes.

Nesse contexto, o objetivo do presente artigo consiste em selecionar bons subconjuntos de casos de teste e mutantes para a redução de custos do TM. Na Seção 5 a abordagem proposta é apresentada.

3 Trabalhos Correlatos

O conceito de Algoritmo Genético (AG) surgiu com os trabalhos de Holland [3]. Inspirado na teoria de Darwin, AG é um algoritmo que simula o processo evolucionário de uma população de indivíduos, baseado em parâmetros e operadores genéticos em busca de uma

solução ótima ou próxima da ótima. No contexto da Engenharia de Software, a utilização dos AGs recebe um destaque importante para resolução de problemas de Teste de Software.

Já a coevolução, é uma evolução complementar de espécies associadas. Podemos visualizar a coevolução na natureza, por exemplo, entre plantas e insetos: a) para sobreviver a planta necessita evoluir um mecanismo para se defender dos insetos, e b) os insetos precisam das plantas como fonte de alimento [15].

Nesse processo coevolucionário, as plantas e os insetos, em cada geração, melhoram suas regras ofensivas e defensivas. Tal interação entre as espécies, faz com que as próximas gerações recebam informações genéticas das gerações anteriores. Este ambiente coevolucionário pode incorporar-se às metaheurísticas evolucionárias para otimizar problemas reais. Além disso, pode ser uma forma de tratar um problema com mais de um objetivo [16], conforme melhor detalhado na Seção 4.

Com base em uma vasta revisão na literatura sobre abordagens que aplicam coevolução no TM foram encontrados apenas dois trabalhos: a pesquisa de Adamopoulos, Harman e Hierons [17] e de Oliveira, Camilo-Junior e Vincenzi [18]. Pode-se perceber diferenças pontuais entre as mesmas:

- **A interação entre as populações**[3]: na abordagem de Adamopoulos, Harman e Hierons [17] a interação é realizada entre todos os indivíduos de uma população contra todos os indivíduos da outra população. Já na abordagem de Oliveira, Camilo-Junior e Vincenzi [18] tal interação realiza-se entre os melhores indivíduos de uma população contra todos os indivíduos da outra população.

- **A função fitness**: as funções fitness propostas na abordagem de Oliveira, Camilo-Junior e Vincenzi [18] utiliza-se da *Classificação Genética* (CG). Já a abordagem de Adamopoulos, Harman e Hierons [17] não tem essse recurso.

- **Os operadores genéticos**: A abordagem de Oliveira, Camilo-Junior e Vincenzi [18] propõem dois novos operadores genéticos (operador de cruzamento *Effective Son* e de mutação *Muta Gene*) adaptados a representação proposta. Já Adamopoulos, Harman e Hierons [17] utilizam os operadores de cruzamento e mutação uniformes.

Considerando a função de aptidão proposta por Adamopoulos, Harman e Hierons [17] (Equação 2), destacam-se dois aspectos positivos: a) evita mutantes equivalentes e; 2) possui alta capacidade de selecionar mutantes *hard-to-kill* [6].

[3] A interação corresponde à execução dos casos de testes sobre os programas mutantes para o cálculo da aptidão dos indivíduos.

$$M_f = \begin{cases} \frac{\sum_{i=1}^{L} Smt_i}{L} & \text{se } \forall i \in S, Smt_i \neq 1, \\ 0 & de\ outro\ modo. \end{cases} \quad (2)$$

Onde:

- Smt_i: o escore de mutação do mutante i;
- *L*: tamanho do indivíduo da população mutantes;

A função de aptidão que Adamopoulos, Harman e Hierons [17] propuseram (Equação 2), utiliza o conceito de escore de mutação para calcular a aptidão de um mutante. Em outras palavras, *escore de mutação* quando se trata de um mutante, é sua a capacidade em evitar ser morto pelos casos de teste.

Portanto, um indivíduo é constituído de *L* mutantes. Logo, M_f é a razão do somatório dos escores dos mutantes do indivíduo pelo seu tamanho, ou seja, a média dos escores. Acerca da penalização, a Equação 2 verifica se há algum mutante *i* com Smt_i = 1. Se houver, significa que o mutante não morreu com nenhum dos casos de teste que o executou. Para o indivíduo que contém este tipo de mutante no seu código genético, atribui-se zero (0) à aptidão M_f desse indivíduo, uma vez que esse mutante pode ser um possível equivalente.

Observa-se que a função de aptidão M_f (Equação 2) pode guiar a evolução do AG para selecionar os mutantes do tipo *hard-to-kill*, o que é interessante, pois esses tipos de mutantes podem conduzir a casos de testes fortes [6]. Por isso, o algoritmo $AGC - CGC$, proposto no presente artigo, utiliza a aptidão descrita da abordagem de Adamopoulos, Harman e Hierons [17] para a população de mutantes.

Entretanto, a função de aptidão para os indivíduos da população de casos de teste não apresentou o mesmo desempenho da abordagem de Oliveira, Camilo-Junior e Vincenzi [18]. A Equação 3 descreve a função de aptidão para um indivíduo da população de casos de teste.

$$T_f = \begin{cases} \frac{\sum_{i=1}^{L} Stc_i}{L} \end{cases} \quad (3)$$

Onde:

- Stc_i: o escore de mutação do caso de teste i;
- *L*: tamanho do indivíduo da população de casos de teste;

Observa-se que T_f consiste na aptidão de um indivíduo da população de casos de teste. Stc_i é o escore de mutação do caso de teste i e L é o tamanho do subconjunto de casos de teste que compõe o indivíduo. Logo, T_f é igual a razão do somatório dos escores de mutação dos casos de teste indivíduo pelo seu tamanho, em outras palavras, T_f é a média dos escores.

O fato de T_f ser dependente de avaliações individuais dos casos de teste, faz com que a busca por um subconjunto forte seja prejudicada. De fato, isso é um problema, uma vez que a evolução é conduzida na seleção de casos de teste que contenham alto escore individual, mas não necessariamente o conjunto destes casos de teste terão um alto escore. No trabalho de Oliveira, Camilo-Junior e Vincenzi [18], a abordagem de Adamopoulos, Harman e Hierons [17] foi aplicada e a Equação 3 não guiou à seleção para subconjuntos com alto escore de mutação, comparada com outras abordagens.

O trabalho de Oliveira, Camilo-Junior e Vincenzi [18] foi conduzido na mesma direção que o de Adamopoulos, Harman e Hierons [17], coevoluindo duas populações em um AG: uma de casos de teste e outra de mutantes. A diferença está na escolha das melhores soluções baseando-se no conceito chamado *Efetividade Genética*. Tal conceito emprega uma busca por *genes efetivos*, ou seja, aqueles genes que contribuem no aumento da aptidão do indivíduo. Para se aplicar a *Efetividade Genética*, Oliveira, Camilo-Junior e Vincenzi [18] propõem a realização da *Classificação Genética* (CG). A Figura 1 ilustra um indivíduo da população de casos de teste com os seus genes classificados.

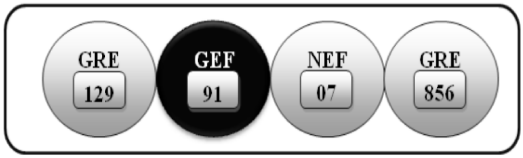

Figura 1. *Classificação Genética* (CG) para um indivíduo da população de casos de teste (GRE - Gene Redundante, GEF - Gene Efetivo e, NEF - Gene não efetivo).

Na abordagem de Oliveira, Camilo-Junior e Vincenzi [18], o cálculo das aptidões dos indivíduos é realizado após a CG dos indivíduos das duas populações. Considerando $IndTC$ um indivíduo da população de casos de teste e $IndMT$ um indivíduo da população de programas mutantes, sendo que Stc e Smt são as medidas de aptidão de $IndTC$ e $IndMT$, respectivamente, o cálculo de Stc e Smt estão descritos nas Equações 4 e 5.

$$Stc = \frac{QMT}{M(P)} \tag{4}$$

Onde:

- QMT: quantidade de mutantes mortos pelos *genes efetivos* de $IndTC$;
- $M(P)$: total de mutantes gerados a partir do programa P.

$$Smt = \begin{cases} \frac{QTC}{T(P)} & se\ p'_i \neq GEQ \quad \forall p'_i \in \text{ IndMT} \\ 0 & de\ outro\ modo. \end{cases} \tag{5}$$

Onde:

- QTC: quantidade de casos de teste evitados pelos *genes efetivos* de $IndMT$;
- $T(P)$: total de casos de teste gerados para testar o programa P.

Em suma, calcular as aptidões dos indivíduos após o emprego da CG, objetiva utilizar somente as informações genéticas que são relevantes para o cálculo. No trabalho de Oliveira, Camilo-Junior e Vincenzi [18] constatou-se que a *Efetividade Genética* foi eficaz no alcance de subconjuntos de casos teste com alto escore de mutação. Por isso, o $AGC-CGC$ também emprega a CG para a população de casos de teste a fim de obter subconjuntos de casos de teste que determinam alto escore de mutação. A diferença é que o $AGC - CGC$ aplica a CG de forma controlada, conforme explicado na Seção 5.1. Além disso, utiliza os operadores de cruzamento *Effective Son* e mutação *Muta Gene* de maneira diferente da abordagem de Oliveira, Camilo-Junior e Vincenzi [18].

A CG é empregada na abordagem de Oliveira, Camilo-Junior e Vincenzi [18] em quatro momentos. Nos dois primeiros momentos, aplica-se a CG antes da realização do cálculo das aptidões das populações de casos de teste e mutantes. Posteriormente, aplica-se a CG no operador de cruzamento ES (*Effective Son*), também nas duas populações. Este operador utiliza dois indivíduos (um pai e uma mãe) e realiza uma CG nos genes desses indivíduos, possibilitando a formação de um filho com maior quantidade de *genes efetivos*, ou seja, com maior aptidão.

Embora aumente o escore, a CG resulta em um tempo de execução maior do que o método proposto por Adamopoulos, Harman e Hierons [17]. Por isso, o $AGC - CGC$, proposto neste artigo, reúne as características positivas das duas abordagens citadas. O objetivo consiste em obter uma maior eficácia do que a abordagem de Adamopoulos, Harman e Hierons [17] e uma maior eficiência do que a abordagem de Oliveira, Camilo-Junior e Vincenzi [18]. A Seção 5 fornece uma explicação mais detalhada sobre como o $AGC - CGC$ integra as evoluções propostas por Adamopoulos, Harman e Hierons [17] e Oliveira, Camilo-Junior e Vincenzi [18] em um único algoritmo.

4 Caracterização do Problema

A alta cardinalidade dos conjuntos de mutantes P' e de um conjunto de casos de teste T, necessários para realização da Teste de Mutação em um programa P, faz com que esse critério apresente alto custo computacional para sua realização, conforme discutido na Seção 2.1.1. Por isso, várias pesquisas são desenvolvidas para aprimoramento desse critério. A seleção de subconjuntos é uma das abordagens de pesquisa [5] para redução dos custos de execução dos mutantes, sendo utilizadas metaheurísticas para seleção de casos de teste e mutantes, a abordagen destacada no presente trabalho.

A importância de uma abordagem de seleção pode ser melhor visualizada na fase do Teste de Regressão, uma vez que o custo da execução dos casos de teste sobre os mutantes pode ser reduzido quando se considera subconjuntos reduzidos, em relação aos conjuntos totais. O problema da *seleção de casos de teste* é definido como segue [19]:

- Dado: o programa P, a versão modificada de P, p' e o conjunto de teste T.
- Objetivo: encontrar um subconjunto $t' \subset T$ que teste P.

Além disso, é desejável reduzir também o conjunto de mutantes para se obter menor custo. Por isso, o problema pode ser formulado com dois objetivos:

1. *Seleção de subconjuntos de casos de teste (Análise de Mutantes):*
   - Dado: o programa, P, o conjunto de mutantes P' e o conjunto de teste T.
   - Objetivo 1: encontrar um subconjunto $t' \subset T$ com maior escore de mutação sobre o conjunto P'.
2. *Seleção de subconjuntos de mutantes (Análise de Mutantes):*
   - Dado: o programa, P, o conjunto de mutantes P' e o conjunto de teste T.
   - Objetivo 2: encontrar um subconjunto $p' \in P'$ com maior escore de mutação[4] sobre o conjunto T'.

As Equações 4 e 5 são exemplos de funções objetivos para esse problema. Assim, para solucionar o problema é necessário o uso de uma abordagem que considere os diferentes e conflitantes objetivos relatados [20].

[4] O escore de mutação no caso dos mutantes [17, 18], é a sua capacidade em evitar ser morto pelos casos de teste, conforme descrito na Seção 3.

Diversos trabalhos aplicam e propõem metaheurísticas, dentre técnicas de buscas locais (*Tabu Search*) [21] e populacionais (NSGA-II) [22] para otimizar problemas multiobjetivo, inclusive para o contexto do Teste de Mutação [23, 24]. Algoritmos coevolucionários também podem ser utilizados para otimizar problemas multiobjetivos [16, 25]. Nos Algoritmos Genéticos (AGs), um indivíduo de uma população é considerado um solução completa do problema. Assim, a coevolução pode ser incorporada ao AGs para que mais de uma espécie coevolua, podendo cada espécie corresponder a um objetivo do problema.

Conforme Seção 3, dois trabalhos (Oliveira, Camilo-Junior e Vincenzi [18] e Adamopoulos, Harman e Hieron [17]) propõem as Equações 4 e 5, 2 e 3, respectivamente, para selecionar de casos de teste e mutantes. Ao final dos algoritmos boas seleções de casos de teste e mutantes são dados como saída e o *trade-off* entre os diferentes objetivos do problema está implícito na interação competitiva [15] entre as duas populações durante a coevolução. Contudo, o problema de seleção de casos de teste e mutantes é caracterizada com dois objetivos conflitantes. A presente pesquisa situa-se nesse contexto e tem um objetivo de avaliar a eficiência do algoritmo $AGC - CGC$ em relação às outras abordagens coevolucionária propostas, a saber, Oliveira, Camilo-Junior e Vincenzi [18] e Adamopoulos, Harman e Hieron [17].

5 O Algoritmo Genético Coevolucionário com Classificação Genética Controlada ($AGC - CGC$)

O $AGC - CGC$ emprega a coevolução considerando: uma população de casos de teste e uma população de programas mutantes. A Figura 2 ilustra o ambiente coevolucionário empregado pelo $AGC - CGC$ que pode ser descrito pelos seguintes passos:

1. **Inicialização das populações**: as populações de casos de teste e mutantes são iniciadas aleatoriamente. Cada população tem um tamanho fixo para formar os indivíduos, conforme estabelecido nos parâmetros;

2. **Calculo das aptidões**:

   - **População de mutantes**: os indivíduos da população de casos de testes executam o teste sobre os indivíduos da população de programas mutantes. A partir dessa interação, calcula-se a aptidão dos indivíduos da população de mutantes, conforme a Equação 2;
   - **População de casos de teste**: após os indivíduos da população de casos de testes terem executados os testes sobre os indivíduos da população de programas mutantes, ocorre a escolha da função de aptidão com base na quantidade de gerações.

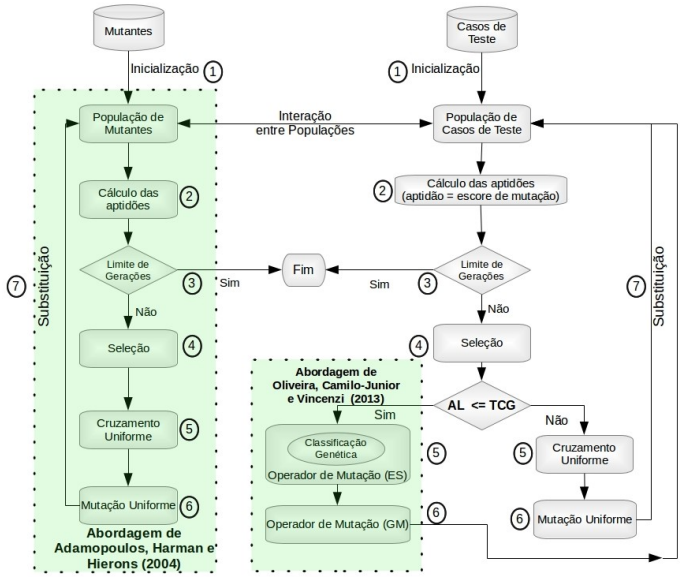

Figura 2. Ambiente Coevolucionário do $AGC - CGC$.

Seja G_c o número atual de gerações. Seja G_t a quantidade total de gerações (definida nos parâmetros do AG). O cálculo da aptidão dos indivíduos da população de casos de teste, obedece a seguinte regra:

- caso $G_c <= G_t/2$, calcula-se a aptidão conforme Equação 3. A intenção consiste favorecer a aptidão dos indivíduos com casos de teste fortes individualmente.
- caso $G_c > G_t/2$, calcula-se a aptidão conforme Equação 4. A intenção consiste em favorecer a aptidão dos indivíduos fortes em conjunto.

3. **Condição de parada do algoritmo**: pode ser pela quantidade de gerações ou pelo alcance da qualidade (escore de mutação) desejada. Na experimentação realizada a condição de parada utilizada foi unicamente a quantidade de gerações.

4. **Operador de seleção**: seleciona os pais para o cruzamento utilizando o operador de seleção *Torneio*;

5. **Operador de cruzamento**:

- **População de mutantes**: o operador de cruzamento utilizado para os indivíduos dessa população é a máscara uniforme [5];
- **População de casos de teste**: População de casos de teste: aplica-se a *Classificação Genética Controlada* (CGC) (Seção 5.1);

6. **Operador de mutação**: aplica-se o operador de mutação conforme *Classificação Genética Controlada* (CGC) (Seção 5.1);

7. **Substituição**: os filhos substituem a população corrente, com elitismo de um indivíduo para cada população.

5.1 A Classificação Genética Controlada (CGC)

A CGC é aplicada somente na população de casos de teste utilizando os operadores genéticos de cruzamento e mutação (ES e GM, respectivamente) propostos por Oliveira, Camilo-Junior e Vincenzi [18]. Tais operadores são utilizados pelo $AGC - CGC$ diferentemente da proposta original.

O controle para aplicação da Classificação Genética baseia-se na Taxa de Classificação Genética (*TCG*), considerando o Percentual Máximo de Classificação (*PMC*) que, por sua vez, é estabelecido como parâmetro do $AGC - CGC$. A *TCG* é calculada conforme a Equação 6.

$$TCG = \frac{G_c * PMC}{G_t} \tag{6}$$

Como exemplo, considere G_t = 500 e *PMC* = 0.30. Na primeira geração (G_c = 1) então *GCR* = 0.0006. Na geração corrente de número 200 (G_c = 200) tem-se *GCR* = 0.12. Por fim, na última geração (G_c = 500) tem-se a *TCG* = 0.30. Logo, a medida que a G_c aumenta, o valor de *TCG* também aumenta, sendo sempre *TCG* = *PMC* na última geração.

O valor TCG define qual o operador de cruzamento será utilizado. Portanto, o controle da CG funciona da seguinte forma:

1. Gera-se, aleatoriamente, o número real AL com valor entre 0 e 1;
2. Calcula-se o valor atual para *TCG*;
3. Caso *TCG* <= AL, aplica-se o operador de cruzamento ES. Caso contrário, aplica-se o operador de máscara uniforme.

[5] Uma máscara binária é gerada de maneira aleatória e dois filhos são construídos pela troca de genes do pai e da mãe.

A característica da *TCG* aumentar conforme se aumenta o número de gerações minimiza o número de aplicações do operador de cruzamento ES. Além disso, objetiva aumentar a probabilidade de aplicação desse operador nas últimas gerações a fim de que a CG seja empregada em informações evoluídas.

6 Configuração dos Experimentos

A presente Seção apresenta os algoritmos (Seção 6.1), os benchmarks (Seção 6.2) e os parâmetros (Seção 6.3) considerados na realização da presente experimentação. Portanto, o algoritmo de Adamopoulos, Harman e Hierons [17], Oliveira, Camilo-Junior e Vincenzi [18] e também o algoritmo aleatório AL (descrito na Seção 6.1) são considerados nessa pesquisa. A compreensão da eficácia e da eficiência dos algoritmos é subsidiada pelas seguintes análises:

1. **Perspectiva dos Benchmarks - Eficácia** (Seção 7.1): avalia os escores de mutação dos algoritmos em cada benchmark, sem levar em consideração quais parâmetros foram utilizados. O objetivo consiste em comparar os algoritmos em diferentes cenários;

2. **Tempo de Execução das Abordagens - Eficiência** (Seção 7.2): avalia os tempos de execução dos algoritmos propostos por Adamopoulos, Harman e Hierons [17]; Oliveira, Camilo-Junior e Vincenzi [18] e; da abordagem aleatória, já falada anteriormente.

3. **Eficácia *versus* Eficiência** (Seção 7.3): aborda os resultados em termos de escore de mutação com o tempo de execução gasto. Dessa forma, o algoritmo proposto pelo presente artigo ($AGC - CGC$) é comparado aos outros algorimos sob a perspectiva da *Eficácia e Eficiência* na obtenção das melhores soluções.

6.1 Algoritmos

Além do algoritmo proposto $AGC - CGC$ (Seção 5), são considerados os seguintes algoritmos na experimentação:

1. **Algoritmo A1**: consiste no algoritmo que representa a abordagem de Adamopoulos, Harman e Hierons [17];

2. **Algoritmo AGC**: consiste no algoritmo que representa a abordagem de Oliveira, Camilo-Junior e Vincenzi [18];

3. **Algoritmo AL**: consiste no algoritmo que representa a abordagem aleatória.

O funcionamento do algoritmo AL, pode ser descrito em três passos:

1. Geração aleatória dos subconjuntos: um subconjunto de casos de teste e outro de mutantes são formados;
2. Cálculo do escore de mutação dos subconjuntos: o escore de mutação do subconjunto é calculado conforme Equação 1.
3. Retorna o melhore subconjunto gerado: gera como saída o melhor subconjunto de mutantes e o melhor subconjunto de casos de teste de todos os que foram gerados.

6.2 Benchmarks

Foram utilizados quatro programas escritos em linguagem C para realização dos experimentos. Tais programas são utilitários UNIX que implementam as seguintes funções:

1. **Cal**: apresenta um calendário para o ano ou mês especificado;
2. **Comm**: seleciona ou rejeita linhas comuns entre dois arquivos;
3. **Look**: procura palavras em um dicionário ou linhas em uma lista ordenada; e
4. **Uniq**: informa ou remove linhas adjacentes duplicadas.

Esses utilitários, além de possuírem uma baixa probabilidade de defeitos em seu código, devido ao seu uso intenso, foram amplamente utilizados em estuerros natudos anteriores. Por exemplo, as pesquisas de Wong [26], Wong et al. [27] e de Vincenzi [28]. Por isso, tais programas são utilizados na pesquisa.

Para geração dos programas mutantes foi utilizada a ferramenta Proteum [29]. A Tabela 1 apresenta informações sobre os conjuntos de programas mutantes e dos conjuntos de casos de teste considerados nos experimentos.

Tabela 1. Informações sobre os utilitários UNIX (benchmarks) utilizados nos experimentos

	Cal	**Comm**	**Look**	**Uniq**
No. de Mutantes	4622	1869	1980	1618
No. de Equivalentes	344	222	233	224
% Equivalentes	7.44%	11.88%	11.77%	13.84%
No. de Casos de Teste	2000	801	255	490
Escore Máximo	0.9981	1.0	1.0	1.0

Observa-se que para todos os programas, o conjunto de teste pode ser considerado um superconjunto com, possívelmente, mais de um subconjunto adequado ao TM.

6.3 Parâmetros

Conforme constatado por Oliveira, Camilo-Junior e Vincenzi [18], o tamanho do indivíduo e o tamanho da população são os parâmetros que exerceram maior influência nos escores de mutação dos algoritmos.

Com isso, optou-se por conduzir uma experimentação comparando os algoritmos em diferentes variações desses parâmetros. A Tabela 2 apresenta as variações nos tamanhos de indivíduos e nos tamanhos das populações.

Tabela 2. Variações nos tamanhos de indivíduos e populações. Considere, *IndTC* como um indivíduo da população de casos de teste (PopTC) e *IndMT* como um indivíduo da população de mutantes (PopMT).

Experimento	**Tamanhos**			
	IndTC	***IndMT***	**PopTC**	**PopMT**
1	10	10	50	50
2	14	14	50	50
3	20	20	50	50
4	30	30	50	50
5	10	10	100	100
6	14	14	100	100
7	20	20	100	100
8	30	30	100	100
9	10	30	50	50
10	10	30	100	100
11	14	30	50	50
12	14	30	100	100

A fim de medir a influência da variação desses tamanhos, foram fixados os seguintes parâmetros: elitismo com 1 indivíduo, 200 gerações, operador de seleção torneio com 2 competidores, taxa de cruzamento em 95%, operador de cruzamento uniforme, taxa de mutação em 5% e operador de mutação uniforme.

Informa-se que todos os experimentos foram feitos em um computador *desktop* com as seguintes configurações de *hardware*: a) 3.39 Gb de memória; b) processador Intel Xeon(R) CPU E5504, 2.00GHz x 4; c) modelo Dell Precision T7500 de 64 bits.

Assim sendo, com base nos parâmetros definidos nessa seção e considerando as variações apresentadas na Tabela 2 os experimentos foram realizados. Nsa seção a seguir são feitas as análises dos resultados.

7 Análise dos Resultados

As análises das Seções 7.1 e 7.2 são correspondentes ao número de 144 experimentos (3 algoritmos - Seção 6.1, 4 benchmarks - Seção 6.2 e 12 variações parâmetros - Tabela 2). A análise da Seção 7.3 incorpora mais 20 experimentos. Uma vez que cada experimento foi executado 30 vezes para cada algoritmo, a experimentação do presente artigo envolve 4.920 execuções.

7.1 Perspectiva dos Benchmarks

A Tabela 3 apresenta os escores de mutação alcançados pelos algoritmos sobre cada benchmark.

Legendas para a Tabela 3:

- **AGs**: nome dos algoritmos utilizados;
- **MAXTC**: escore máximo alcançado pelo melhor *IndTC*[6] encontrado.
- **MEDTC**: média dos escores de mutação dos melhores *IndTCs*;
- **DESVPTC**: desvio padrão dos escores de mutação dos melhores *IndTCs*;
- **MAXMT**: escore máximo alcançado pelo melhor *IndMT* [7] encontrado.
- **MEDMT**: média dos escores de mutação dos melhores *IndMTs*;
- **DESVPMT**: desvio padrão dos escores de mutação dos melhores *IndMTs*;

Considerando a Tabela 3 e os indivíduos selecionados da população de mutantes (*IndMTs* - subconjuntos de mutantes), percebe-se que todos os algoritmos selecionaram *IndMTs* com escore igual a 1.0 em todas as execuções. Em relação ao benchmark Cal e considerando os indivíduos selecionados da população de casos de teste (*IndTCs* - subconjuntos de casos de teste), observa-se maiores variações nos escores de mutação entre os algoritmos.

[6] *IndTC* é indivíduo da população de casos de teste, ou seja, um subconjunto de casos de teste.
[7] *IndMT* é indivíduo da população de mutantes, ou seja, um subconjunto de mutantes.

Tabela 3. Desempenho dos algoritmos - Benchmarks: Cal, Comm, Look e Uniq

	CAL			COMM		
AGs	**A1**	**AL**	**AGC**	**A1**	**AL**	**AGC**
MAXTC	0,9357	0,9135	0,9974	0,8591	0,9411	0,9994
MEDTC	0,7257	0,8363	0,9901	0,8403	0,8774	0,9851
DESVPTC	6,72E-002	6,41E-002	7,70E-003	9,88E-003	2,90E-002	1,40E-002
MAXMT	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000
MEDMT	1,0000	1,0000	1,0000	1,0000	0,9997	1,0000
DESVPMT	7,88E-005	2,63E-005	0,00E+000	0,00E+000	3,56E-003	0,00E+000
	LOOK			**UNIQ**		
AGs	**A1**	**AL**	**AGC**	**A1**	**AL**	**AGC**
MAXTC	0,9353	0,9399	0,9994	0,9900	1,0000	1,0000
MEDTC	0,8912	0,8180	0,9853	0,9672	0,9890	1,0000
DESVPTC	1,71E-002	7,55E-002	1,48E-002	7,50E-003	1,71E-002	8,40E-005
MAXMT	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000
MEDMT	1,0000	0,9999	1,0000	1,0000	0,9983	1,0000
DESVPMT	0,00E+000	7,97E-004	0,00E+000	0,00E+000	1,05E-002	0,00E+000

O algoritmo *AGC* obteve os melhores escores de mutação: a média de 0.9901 (MEDTC) e o máximo alcançado de 0.9974 (MAXTC). Já os algoritmos *A1* e *AL* alcançaram escores inferiores. *A1* obteve a média de 0.7257 (MEDTC) e o máximo de 0.9357 (MAXTC). *AL* obteve a média de 0.8363 (MEDTC) e o máximo de 0.9135 (MAXTC). Observa-se que o algoritmo *AL* (abordagem aleatória) obteve um escore médio maior do que o alcançado pelo algoritmo *A1*.

Nos outros benchmarks (Comm, Look e Uniq), os resultados continuam semelhantes aos percebidos no benchmark Cal, o algoritmo *AGC* é superior às outras abordagens na seleção de subconjuntos de casos de teste com escores de mutação mais elevados. Por outro lado, vale observar que as abordagens coevolucionárias evitaram mutantes equivalentes, diferentemente, da abordagem aleatória (AL) que não possui a penalização dos possíveis equivalentes.

7.2 Tempo de Execução das Abordagens

A análise anterior (Seções 7.1, subsidia a avaliação dos algoritmos propostos por: Adamopoulos, Harman e Hierons [17] e Oliveira, Camilo-Junior e Vincenzi [18], em termos de eficácia (escore de mutação). A Tabela 4 fornece os tempos de execução dos algoritmos com relação à cada benchmark.

Tabela 4. Análise do tempo de execução por benchmark (Algoritmos A1, AL e AGC)

Cal		Comm	
AG	**TEMPO (min)**	**AG**	**TEMPO (min)**
A1	0,82	A1	0,37
AL	0,10	AL	0,03
AGC	36,55	AGC	9,99
Look		**Uniq**	
AG	**TEMPO (min)**	A1	**TEMPO (min)**
A1	0,12	A1	0,22
AL	0,02	AL	0,02
AGC	9,24	AGC	4,91

Conforme Tabela 4 percebe-se que *AL* é o algoritmo com menor tempo de execução. O segundo algoritmo mais eficiente é o algoritmo *A1*. Por fim, o algoritmo com o tempo de execução maior é *AGC*.

Contrastando, os resultados de tempo de execução da Tabela 4 com os escores de mutação apresentados nas Tabelas 3, percebe-se que o algoritmo *AGC* é o mais eficaz em encontrar melhores soluções, porém o mais ineficiente em termos de tempo de execução.

De fato, a *Classificação Genética* (GC) é responsável pela alta qualidade dos resultados alcançados pelo *AGC*. Entretanto, também pode ser considerada como um mecanismo que aumenta substancialmente o custo computacional, observando o tempo de execução do AGC.

Por outro lado, percebe-se que o algoritmo *A1* opera em um tempo eficiente e seleciona bons subconjuntos de mutantes. Entretanto, é ineficaz em selecionar bons subconjuntos de casos de teste.

7.3 Eficácia *versus* Eficiência

As análises das Seções anteriores demonstram:

- Abordagem de Oliveira, Camilo-Junior e Vincenzi [18]:
  - Aspectos positivos: a) eficaz em obter subconjuntos de casos de teste com alto escore de mutação e; b) eficaz em obter subconjuntos de mutantes com alto escore de mutação
  - Aspecto negativo: a) ineficiente em tempo de execução, com relação às outras abordagens.

- Abordagem de Adamopoulos, Harman e Hierons [17]:
  - Aspectos positivos: a) eficaz em reunir mutantes difíceis de matar nos subconjuntos de mutantes selecionados e; b) eficiente em tempo de execução;
  - Aspecto negativo: a) ineficaz na seleção de casos de teste com alto escore de mutação;

Diante dessas conclusões, surge o algoritmo $AGC-CGC$, projetado com a união dos aspectos positivos dos algoritmos *A1* e *AGC*. Dessa forma, espera-se que o $AGC - CGC$:

- Seja mais eficaz que o algoritmo *A1* na seleção de casos de teste com alto escore. Para isso, aplica a *Classificação Genética* na população de casos de teste (Seção 5.1).
- Opere de forma mais eficiente que o *AGC*. Para isso, emprega da *Classificação Genética Controlada* (CGC) (Seção 5.1);

A Tabela 5 apresenta os escores de mutação alcançados pelos algoritmos *A1*, *AL*, *AGC* e $AGC - CGC$, bem como os tempos de execução de cada algoritmo para o alcance da melhor solução. Os resultados apresentados são correspodentes às execuções dos algoritmos sobre o benchmark **Cal** e **Comm** considerando o experimento 6 (Tabela 2). Tais *benchmarks* foram escolhidos dentre os quatro por serem os cenários mais difíceis, considerando os critérios de escore de mutação e tempo de execução, apresentados nas Tabelas 3 e 4. Além disso, vale observar que o $AGC - CGC$ considera a TCG = 30%.

Tabela 5. Benchmark Cal e Comm - escores de mutação e tempos de execução dos algoritmos para análise de *Eficácia* e *Eficiência* (Experimento 6 da Tabela 2).

	Cal		
	Escore Máximo	**Escore Médio**	**Tempo (min)**
AGC	0,9943	0,9907	8,3550
AGC-CGC	0,9862	0,9577	0,5983
A1	0,6905	0,6827	0,3017
	Comm		
	Escore Máximo	**Escore Médio**	**Tempo (min)**
AGC	0,9890	0,9803	3,1833
AGC-CGC	0,9914	0,9837	0,3833
A1	0,8476	0,8392	0,1500

Conforme a Tabela 5, em termos de escore de mutação (Eficácia), o algoritmo A1 teve o pior desempenho. O $AGC - CGC$ obteve um escore de mutação médio de 0.9803 e máximo de 0.9862 contra 0.9890 e 0.9803 do algotitmo AGC, respectivamente. Ou seja,

o melhor desempenho na seleção de um subconjunto de casos de teste (*IndTC*) para o benchmark **Comm**. Já considerando o programa **Cal**, o AGC teve o melhor desempenho, nesse quesito. Portanto, percebe-se que o $AGC - CGC$ foi bastante competitivo com o algoritmo AGC.

Por outro lado, considerando o quesito tempo de execução (Eficiência), percebe-se que o algoritmo AGC teve o pior desempenho. Nesse quesito, A1 obteve melhor resultado. Entretanto, o algoritmo $AGC-CGC$ foi o mais competitivo comparado ao A1. Por exemplo, o $AGC - CGC$ obteve um tempo de execução de 0.59 minutos contra 0.30 minutos do algoritmo A1, para o benchmark **Cal**. Já com o benchmark **Comm** o $AGC - CGC$ obteve 0.38 minutos contra 0.15 minutos do algoritmo A1.

Considerando os dois benchmarks (Cal e Comm), o $AGC - CGC$ obteve um ganho de 20.77% em relação ao A1 na média dos escores de mutação alcançados, bem como como reduziu o tempo de execução em 91.49% na média dos tempos de execução gastos em relação ao AGC, para seleção de melhores subconjuntos de casos de teste com tamanho 14, conforme Tabela 5. Dessa forma, a Classificação Genética Controlada (CGC - Seção 5.1) inerente ao $AGC - CGC$ contribuiu para o ganho em termos de escore de mutação e na redução do tempo de execução, frente as outras abordagens.

7.4 Influência da Variação na Taxa de Classificação Genética (TCG)

O controle da CG (Seção 5.1) é realizado por meio da Taxa de Classificação Genética (TCG). Conforme seção anterior, a TCG possibilitou o alcance de bons resultados em tempos mais reduzidos. Uma possibilidade de análise, consiste em aplicar o algoritmo $AGC - CGC$ sob diferentes TCGs objetivando um melhor entendimento da influência da TCG nos resultados alcançados.

A Figura 3 apresenta a influência nos escores de mutação alcançados pelo $AGC - CGC$ considerando a TCG variando de 5% até 100%. Além disso, considerou-se o Experimento 8 da Tabela 2 que define o maior tamanho para os subconjuntos, exigindo portanto, um maior custo computacional para realização da CG.

Já a Figura 4 apresenta a influência nos tempos de execução alcançados pelo $AGC - CGC$ considerando a TCG variando de 5% até 100%, considerando também o Experimento 8 da Tabela 2 para definir o tamanho dos subconjuntos selecionados.

Conforme Figura 3, a TCG = 100% obteve o maior escore de mutação alcançado. Entretanto, conforme Figura 4 a TCG = 100% remeteu a um tempo de execução mais elevado. Esses resultados explicam a grande eficácia do AGC e ao mesmo tempo sua grande ineficiência, uma vez que o AGC não controla a CG.

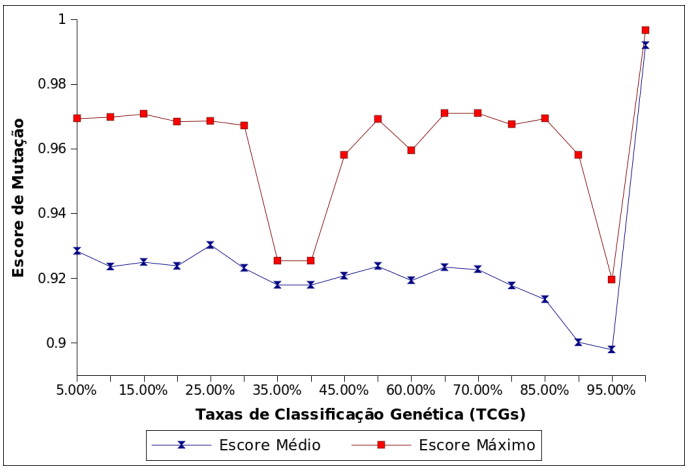

Figura 3. Variação das Taxas de Classificação Genética (TCGs) - influência nos escores de mutação do $AGC - CGC$.

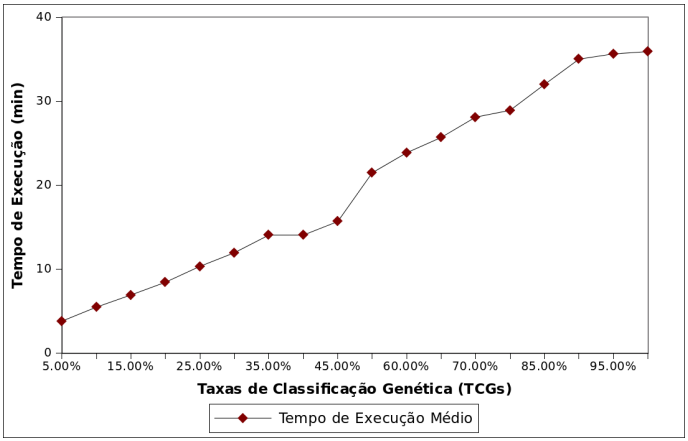

Figura 4. Variação das Taxas de Classificação Genética (TCGs) - influência nos tempos de execução do $AGC - CGC$.

Na Figura 3, observa-se que o fato de aumentar o valor da TCG não significa em um aumento do escore de mutação. Por exemplo, a TCG = 30% obteve um escore de mutação médio superior a 0.92. Entretanto, a TCG entre 35% e 40% provocou uma redução no escore de mutação médio. Todavia, retirada a taxa de TCG = 100%, as demais taxas não tiveram um acréscimo ou redução significativa na eficácia do algoritmo. No caso da média, por exemplo, a diferença entre a melhor e a pior foi de apenas 3,47%. O que se pode concluir com base na experimentação realizada, é que nem sempre o aumento da TCG implicou em um aumento do escore de mutação. Por outro lado, a relação confirmada é que o tempo de execução do $AGC - CGC$ é diretamente proporcional ao aumento da TCG, conforme se observou na Figura 4.

Por outro lado, o aumento do valor da TCG implica em um maior tempo de execução para o $AGC - CGC$ encontrar a melhor solução.

8 Conclusões e Trabalhos Futuros

Esse artigo apresentou três abordagens coevolucionárias como propostas de solução para um problema com dois objetivos conflitantes presentes no TM: a seleção automática de bons subconjuntos de casos de teste e programas mutantes.

Inicialmente, as abordagens que aplicaram a coevolução no TM [17, 18] foram avaliadas com a realização de diversos experimentos. Esse estudo comparou os resultados das metaheurísticas propostas com os alcançados pelo método de seleção aleatório. Como resultado, obteve-se a compreensão dos aspectos positivos e negativos de tais abordagem.

Nesse contexto, foi apresentado o algoritmo $AGC - CGC$ como uma proposta para seleção automática de casos de teste e mutantes no contexto da Análise de Mutantes. Tal algoritmo foi concebido pela hibridização de duas abordagens existentes. As três abordagens foram comparadas em termos de *Eficácia* e *Eficiência*.

O algoritmo $AGC-CGC$ obteve escores de mutação com um ganho de 20.77% sobre o algoritmo *A1*, bem como operou com uma redução de 91.49% sobre os tempos empregados pelo algoritmo *AGC*.

Diante disso, acredita-se que o $AGC - CGC$ constitui-se numa abordagem promissora para seleção de casos de teste e programas mutantes para o TM, uma vez que conseguiu balancear atributos de *Eficácia* e *Eficiência* frente às abordagens existentes na literatura.

Algumas das contribuições desse trabalho podem ser listadas abaixo:

1. A avaliação das abordagens que aplicam a coevolução no TM;

2. A *Classificação Genética Controlada* (CGC) para redução de custo computacional na seleção de bons subconjuntos de casos de teste; e

3. Um estudo sobre a influêcia de diferentes Taxas de Classificação Genética (TCGs) nos escores de mutação e nos tempos de execução empregados pelo $AGC - CGC$.

Dentre os possíveis trabalhos futuros a serem realizados, pode-se estudar outros meios para realização do controle da *Classificação Genética* a fim de se obter maiores escores de mutação com um custo computacional ainda menor.

Outra pesquisa que pode ser desenvolvida, consiste na aplicação do $AGC-CGC$ para redução de custos em outros contextos. Por exemplo: a) no Teste de Mutação no nível de interface [30]; b) na Análise de Mutantes SQL [31] e; c) na Análise de Mutantes considerando mutantes de alta ordem (Higher Order Mutation [32]). Além disso, pode-se também estudar o $AGC - CGC$ como uma metaheurística de seleção de operadores essenciais [23, 27]. Levando em consideração a possibilidade de haver relação entre os mutantes difíceis de morrer (*hard-to-kill*) e os operadores essenciais.

Referências

[1] Mike Papadakis, Nicos Malevris, and Maria Kallia. Towards Automating the Generation of Mutation Tests. *Proceedings of the 5th Workshop on Automation of Software Test - AST '10*, pages 111–118, 2010.

[2] Phil McMinn. Search-Based Software Testing: Past, Present and Future. *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*, pages 153–163, March 2011.

[3] Kenneth A. De Jong. *Evolutionary Computation: A Unified Approach*. MIT Press, 2006.

[4] Márcio Eduardo Delamaro, Mario Jino, and José Carlos Maldonado. *Introduction to Software Testing*. Elsevier Ltd (in portuguese), 2007.

[5] Yue' Jia and Mark Harman. An analysis and survey of the development of mutation testing. *Software Engineering, IEEE Transactions on*, 37:649 – 678, 2011.

[6] AJ Offutt and JH Hayes. A Semantic Model of Program Faults. In *International Symposium on Software Testing, Verification, and Analysis*, pages 195–200. ACM Press, 1996.

[7] A. A. L. Oliveira, C. G. Camilo, and A.M.R. Vincenzi. Um algoritmo genético coevolucionário com classificação genética controlada aplicado ao teste de mutação. In *IV Workshop de Engenharia de Software Baseada em Busca (WESB)*.

[8] R.a. DeMillo, R.J. Lipton, and F.G. Sayward. Hints on Test Data Selection: Help for the Practicing Programmer. *Computer*, 11(4):34–41, April 1978.

[9] E William. Weak Mutation Testing and Completeness of Test Sets. (4):78–85, 1982.

[10] AP Mathur. Performance, effectiveness, and reliability issues in software testing. *Computer Software and Applications Conference*, pages 0–1, 1991.

[11] W Krauser, , Aditya P Mathur, and Vernon J Rego. High Performance Software Testing on SIMD Machines. 17(5), 1991.

[12] A. Jefferson Offutt, Jie Pan, Kanupriya Tewary, and Tong Zhang. An experimental evaluation of data flow and mutation testing. *Softw. Pract. Exper.*, 26(2):165–176, February 1996.

[13] Yu-Seung Ma, Jeff Offutt, and Yong Rae Kwon. MuJava: an automated class mutation system. *Software Testing, Verification and Reliability*, 15(2):97–133, June 2005.

[14] Shamaila Hussain. *Mutation Clustering*. PhD thesis, KingâĂŹs College London, 2008.

[15] AP Engelbrecht. *Computational Intelligence: An introduction*. John Wiley & Sons, Ltd, South Afrca, 2 edition, 2007.

[16] SG Ficici. *Solution concepts in coevolutionary algorithms*. PhD thesis, University Brandeis, 2004.

[17] Konstantinos Adamopoulos, Mark Harman, and R Hierons. How to Overcome the Equivalent Mutant Problem and Achieve Tailored Selective Mutation Using Co-Evolution. In *Genetic and Evolutionary Computation*, 2004.

[18] A. A. L. Oliveira, C. G. Camilo-Junior, and A.M.R. Vincenzi. A coevolutionary algorithm to automatic test case selection and mutant in mutation testing. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 829–836, 2013.

[19] S. Yoo and M. Harman. Regression testing minimization, selection and prioritization: A survey. *Softw. Test. Verif. Reliab.*, 22(2):67–120, March 2012.

[20] FG de Freitas and CLB Maia. Otimização em Teste de Software com Aplicação de Metaheurísticas. *Revista de Sistemas de ...*, 5:3–13, 2010.

[21] K. C. Tan, E. F. Khor, T. H. Lee, and Y. J. Yang. A tabu-based exploratory evolutionary algorithmfor multiobjective optimization. *Artif. Intell. Rev.*, 19(3):231–260, May 2003.

[22] Shin Yoo and Mark Harman. Pareto Efficient Multi-objective Test Case Selection. *Proceedings of the 2007 international symposium on Software testing and analysis - ISSTA '07*, pages 140–50, 2007.

[23] Adam S. Banzi, Tiago Nobre, Gabriel B. Pinheiro, JoãO Carlos G. ÁRias, Aurora Pozo, and Silvia Regina Vergilio. Selecting mutation operators with a multiobjective approach. *Expert Syst. Appl.*, 39(15):12131–12142, November 2012.

[24] Tiago Nobre. *Uma Abordagem Baseada em Algoritmos de Otimização Multiobjetivos para Reduzir o Custo do Critério de Teste Análise de Mutantes*. PhD thesis, Universidade Federal do Paraná, 2011.

[25] Richard A. Watson and Jordan B. Pollack. Symbiotic combination as an alternative to sexual recombination in genetic algorithms. In *Proceedings of the 6th International Conference on Parallel Problem Solving from Nature*, PPSN VI, pages 425–434, London, UK, UK, 2000. Springer-Verlag.

[26] W.E. Wong. On mutation and data flow. Technical report, Purdue University, Dec 1993.

[27] W.E. Wong, J.C. Maldonado, M.E. Delamaro, and S.R.S. Souza. A comparison of selective mutation in C and fortran. In *Workshop do Projeto Validação e Teste de Sistemas de Operação*, pages 71–80, Águas de Lindóia, SP, January 1997.

[28] Auri Marcelo Rizzo Vincenzi. *Subsídios para o Estabelecimento de Estratégias de Teste Baseadas na Técnica de Mutação*. PhD thesis, Universidade de São Paulo, 1998.

[29] Márcio Eduardo Delamaro and José Carlos Maldonado. Proteum-A tool for the assessment of test adequacy for C programs. In *Proceedings of the Conference on Performability in Computing Systems (PCS 96)*, pages 79–95, New Brunswick, NJ, July 1996.

[30] M. E. Delamaro, J. C. Maldonado, and A. M. R. Vincenzi. Proteum/IM 2.0: An integrated mutation testing environment. In *Mutation 2000 Symposium*, pages 91–101, San Jose, CA, October 2000. Kluwer Academic Publishers.

[31] J. Tuya, M.J. Suarez-Cabal, and C. de la Riva. Sqlmutation: A tool to generate mutants of sql database queries. In *Mutation Analysis, 2006. Second Workshop on*, pages 1–1, Nov 2006.

[32] Yue Jia and Mark Harman. Constructing Subtle Faults Using Higher Order Mutation Testing. *2008 Eighth IEEE International Working Conference on Source Code Analysis and Manipulation*, pages 249–258, September 2008.