Uma abordagem para redução do custo do Teste de Mutação utilizando Redes Neurais

Lucas Lagôa Nogueira[1] **(mestrando), Márcio Eduardo Delamaro**[1] **(orientador)**

[1]Mestrado Acadêmico em Ciência da Computação
Programa de Pós-Graduação em Ciência da Computação
Universidade de São Paulo - Instituto de Ciências Matemáticas e de Computação
CEP: 13566-590 - São Carlos - SP
Ingresso: 03/2019 - Qualificação: 06/2020 - Previsão Defesa: 03/2021

lucaslagoa@usp.br, delamaro@icmc.usp.br

Resumo. *Durante o processo de criação de um software, é essencial conduzir atividades com o objetivo de revelar a existência de possíveis defeitos. Nesse contexto, as atividades de Teste de Software permitem o fornecimento de evidências da confiabilidade do software. Para tanto, o Teste de Software possui várias técnicas que podem ser aplicadas com suas diferentes características. Uma das técnicas existentes é o Teste Baseado em Defeitos em que o principal critério dessa técnica é o Teste de Mutação. O Teste de Mutação ocorre por meio da criação de programas mutantes se baseando no programa original, realizando alterações sintáticas no programa com objetivo de alterar a semântica. Se um caso de teste detectar o mutante do programa original, ele possui maior probabilidade de revelar defeitos. Porém, o Teste de Mutação possui algumas limitações, tais como a geração de mutantes equivalentes, a geração de mutantes redundantes e também a grande geração de mutantes. Recentemente, técnicas de Aprendizado de Máquina têm sido aplicadas para auxiliar o Teste de Software e também o Teste de Mutação. Este trabalho de mestrado tem como principal objetivo de projetar uma abordagem de redução desse custo do teste de mutação. Dessa maneira, pretende-se utilizar técnicas de Aprendizado de Máquina e Redes Neurais para desenvolver a abordagem. Tal abordagem permitirá uma melhora na eficiência da aplicação do critério permitindo uma redução da geração de mutantes.*

Palavras-chave: *teste de software, teste de mutação, aprendizado de máquina, redes neurais.*

Eventos relacionados: *SBES/SAST*

1. Caracterização do problema

Durante o processo de criação de um *software*, atividades relacionadas a garantia de qualidade são essenciais [Bertolino 2007]. Nesse sentido, para a maioria dos sistemas em desenvolvimento, a atividade de teste de software é o meio mais importante pelo qual tais sistemas são submetidos para terem sua conformidade com especificações verificadas [Hunter and Strooper 2001, Ammann and Offutt 2008]. Negligenciar as atividades de teste, muitas vezes, pode remeter à produção de software de má qualidade e prejuízos econômicos. O Teste de Software fornece evidências em relação à confiabilidade do software [Myers 2006].

A literatura dispõe de uma diversidade de técnicas e critérios que podem ser aplicadas de maneira conjunta e complementar, com o objetivo de garantir uma maior completude de verificação do software [Delamaro et al. 2017c]. Uma dessas técnicas é o Teste Baseado em Defeitos que tem como objetivo derivar os requisitos de teste por meio de possíveis defeitos que são cometidos durante o processo de desenvolvimento de software [Delamaro et al. 2017b]. Dentre os critérios do teste baseado em defeitos, destaca-se o Teste de Mutação. O teste de mutação se dá por meio da criação de programas mutantes que são criados com base no programa original realizando alterações sintáticas no programa com o objetivo de alterar sua semântica [Delamaro et al. 2017b]. Consequentemente, os casos de teste que conseguem distinguir o programa original do programa mutante, têm maior probabilidade de revelar defeitos, assim, podem ser considerados mais adequados que outros [Madeyski et al. 2014].

Mesmo que o teste de mutação apresente muitas vantagens, o mesmo pode se tornar impraticável devido ao alto custo da aplicação da técnica, tais como: (i) a grande geração de mutantes equivalentes; (ii) a grande geração de mutantes redundantes; e (iii) a grande geração de mutantes afetam a eficiência da aplicação da técnica. [Wong and Mathur 1995]. Uma das possíveis alternativas para mitigar tais problemas é a aplicação de Aprendizado de Máquina (AM) para complementar o Teste de Mutação. [Márki 2019] e [Chekam et al. 2018] propuseram possíveis soluções utilizando algoritmos de AM para melhorar o Teste de Mutação. No entanto, ainda existem poucos estudos que aplicam algoritmos de AM para impulsionar o teste de mutação [Durelli et al. 2019].

Motivado por esse cenário, este trabalho de mestrado visa aplicar técnicas de AM para reduzir o custo da aplicação do teste da mutação. Para tanto, os seguintes objetivos específicos foram traçados: (i) redução do número de mutantes gerados; (ii) melhora na eficiência da aplicação do critério; e (iii) uma nova abordagem para aplicação do teste de mutação com aplicação de AM.

2. Fundamentação Teórica

Nesta seção são apresentados conceitos relevantes para o entendimento do presente projeto de mestrado em andamento. Com isso, são apresentados conceitos relacionados à Teste de Software como o Teste de Mutação e conceitos relacionados à AM como as Redes Neurais.

2.1. Fundamentação Teórica de Teste de Software

Existem técnicas que permitem organizar e melhorar a aplicação do teste de software. Uma dessas técnicas é o teste baseado em defeitos. O teste de mutação ou análise de

mutantes é um critério da técnica de teste baseado em defeitos. Os defeitos típicos do processo de implementação de um software são utilizados nessa técnica para a formação dos requisitos de teste [Delamaro et al. 2017a]. Para realizar essa estratégia de teste deve-se realizar alterações no programa que está sob teste; criando programas alternativos, simulando a inserção de defeitos diferentes em cada versão. Cada versão alternativa criada com a inserção de um defeito é considerada um mutante e cada mutante criado representa um subdomínio do domínio de entrada.

O teste de mutação refere-se ao processo de usar a análise dos mutantes para oferecer suporte ao teste, quantificando os pontos fortes do conjunto de testes [Papadakis et al. 2019]. Quando um teste realizado consegue detectar o comportamento diferenciado do mutante comparando com o software original é dito que o mutante foi 'detectado' ou 'morto', e, quando não há detecção, é dito que ele permaneceu 'vivo'. Para realizar a geração dos mutantes é realizada a mudança sintática do software. A mudança da sintaxe do software segue um conjunto de regras mais conhecido como 'operadores de mutação'. Os operadores de mutação são responsáveis por definir quais serão as alterações na sintaxe do programa. O objetivo do teste é 'matar os mutantes', isto é, gerar casos de testes que detectam as mudanças realizadas no código mutante.

Os mutantes vivos devem ser analisados atentamente pois eles podem apresentar comportamentos interessantes. Um desses comportamentos é que ele pode ser considerado um mutante equivalente. Mutantes equivalentes são mutantes que possuem o mesmo comportamento do programa, fazendo que se tornem indistinguíveis, tornando impossível a morte do mutante. O problema gerado pelos mutantes equivalentes é geralmente indecidível, pois necessita da intervenção humana para que se realize uma análise e decidir se um mutante é equivalente ou não evitando que se crie novos casos de testes que falharão em matar esse mutante. Logo após todos os mutantes se encontrarem mortos é possível realizar o cálculo do nível de adequação desse conjunto de teste por meio de um escore de mutação. O escore de mutação é um número no intervalo de 0 e 1; consequentemente, quanto mais próximo do 1 for o escore de mutação, maior é o nível de adequação do conjunto de casos de teste.

Outro problema que pode afetar o teste de mutação são os mutantes redundantes. Os mutantes redundantes são mutantes que afetam a qualidade de um conjunto de testes. Por exemplo, alguns mutantes são mortos por quase qualquer teste, com isso, eliminar esses mutantes não afeta na escolha dos testes, mas resulta em um escore de mutação diferente. Em outras palavras, pode-se dizer que os escores de mutação podem ser inflacionados por mutantes redundantes [Ammann et al. 2014b]. Para explicitar o que são os mutantes redundantes é necessário a definição de alguns termos como:

- **Conjunto de Testes Minimais**: O conjunto de teste só é minimal se caso cada um dos casos de teste forem removidos individualmente, o escore de mutação será alterado em cada remoção. Com isso, cada caso de teste é considerado importante para que não ocorra alterações no escore de mutação.
- **Mutantes Redundantes**: Para que um mutante seja considerado redundante, sua remoção não deve alterar o conjunto de mutantes minimais.
- **Conjunto de Mutantes Minimais**: Para que um conjunto de mutantes seja minimal o conjunto não pode conter nenhum mutante redundante [Ammann et al. 2014a].

A identificação do conjunto de mutantes minimais é importante pois esse conjunto não possui nenhum mutante redundante, o que faz com que esse conjunto seja necessário ser identificado. Como apresentado, o teste de mutação é considerado custoso devido à grande geração de mutantes que está relacionado aos mutantes equivalentes e aos redundantes.

2.2. Fundamentos de Inteligência Artificial e Aprendizado de Máquina

A área de IA era vista como uma área teórica, porém a partir da década de 1970, houve um avanço em desenvolvimento de técnicas da área da computação para a resolução de problemas reais de IA [Faceli et al. 2011]. Com a grande geração de dados gerados e com o crescimento da complexidade de problemas que são tratados computacionalmente, viu-se uma necessidade de desenvolvimento de ferramentas computacionais que possam ser mais autônomas. Para que essas ferramentas consigam ser desenvolvidas, elas deveriam possuir a capacidade de criar uma hipótese ou função por si próprias, por via de experiência passada. O processo de criação dessa ferramenta é denominado Aprendizado de Máquina [Faceli et al. 2011]. Uma das áreas do AM que será abordado neste trabalho são as Redes Neurais.

As Redes Neurais ou Redes Neurais Artificiais (RNAs) podem ser definidas como sistemas computacionais distribuídos compostos de processamento simples densamente interconectadas. As RNAs procuram simular os neurônios presentes no cérebro através dos neurônios artificiais, e com isso, fazer com que sejam capazes de computar funções matemáticas [Faceli et al. 2011]. Estudos como o de [Chekam et al. 2018] e o de [Márki 2019] nos demonstraram que uma possível solução para a redução do custo do teste de mutação pode vir dos algoritmos de AM, com isso, a técnica de redes neurais foi estudada como sendo uma possível solução para o problema do trabalho em questão ao identificar os mutantes equivalentes e mutantes redundantes.

3. Metodologia

Este trabalho pretende realizar a criação e verificar a eficácia de um algoritmo de rede neural ao identificar mutantes que pertencem ao conjunto minimal de mutantes (mutantes redundantes) e mutantes equivalentes. Para tanto, esta pesquisa de mestrado objetiva responder as seguintes Questões de Pesquisa (QPs):

- **QP1**: *Quão preciso é a rede neural na identificação de mutantes minimais?*
- **QP2**: *Quão preciso é a rede neural na identificação de mutantes equivalentes?*

Com isso, pode-se dizer que o método de pesquisa consiste nas etapas definidas abaixo:

1. Análise das atividades envolvidas no teste de software, teste de mutação e inteligência artificial com foco para as redes neurais;
2. Estudo das técnicas para redução do custo de mutação;
3. Estudo e definição dos atributos de programas, mutantes e casos de teste, que servirão como entrada para a criação de modelos preditivos;
4. Definição da ferramenta para geração dos programas mutantes;
5. Estudo para definição de ferramentas utilizadas na construção do modelo preditivo;

6. Desenvolvimento do modelo preditivo que auxilie na redução do custo da aplicação do Teste de Mutação;
7. Definição e condução de estudos experimentais;
8. Escrita de artigos científicos e o documento de dissertação.

4. Estado atual do trabalho

A partir da metodologia proposta, as seguintes atividades foram conduzidas: primeiramente, uma revisão da literatura sobre Teste de Software e AM foi realizado. A partir dos resultados da revisão, definiram-se os atributos (*features*) seriam utilizados para criar o modelo preditivo, dado que os atributos possuem um papel importante para garantir a qualidade do modelo. Esses atributos nos ajudariam a determinar o comportamentos dos mutantes: equivalentes ou não e pertencentes ao conjunto minimal ou não. Alguns atributos que serão levados em consideração são: *Operator*, *Complexity* e *Type Statement*, entre outros.

Após a decisão dos atributos, foi pesquisado ferramentas que poderiam nos auxiliar a gerar os programas mutantes. A ferramenta escolhida foi a *Proteum (PROgram Testing Using Mutants)* [Delamaro et al. 1996] que é uma ferramenta que constrói, compila, executa cada mutante comparando seu comportamento com o do programa em teste. Os programas e conjuntos de testes que irão compor nossa amostra foram fornecidos pelo segundo autor desse artigo. Os programas mutantes que serão gerados serão baseados 27 programas na linguagem C e ao total esses programas possuem 47 funções. Alguns exemplos de programas que serão utilizados são: *CheckPalindrome* e o algoritmo de ordenação *MergeSort*.

Com a definição dos atributos e da ferramenta para geração dos programas mutantes, o estado atual que o trabalho se encontra é o de identificar ferramentas que possam auxiliar na construção do modelo preditivo.

5. Método para avaliar resultados

Com a criação do modelo preditivo da rede neural de acordo com os atributos selecionados, é necessário descobrir um método para avaliar os resultados que serão obtidos. A avaliação de um modelo preditivo pode ser realizada através de várias métricas, porém as métricas mais adotadas na literatura são: Acurácia, Precisão, Revocação e F1-Score. Um conceito importante que precede a aplicação de métricas é a matriz de confusão, que separa as previsões, como mostrado abaixo:

- True Positives (TP): Previsões feitas como positivas e realmente positivas;
- False Negatives (FN): Previsões feitas como negativas, mas na realidade são positivas;
- False Positives (FP): Previsões feitas como positivas, mas na realidade são negativas;
- True Negatives (TN): Previsões feitas como negativas e realmente negativas.

A fórmula para cada uma das métricas é mostrada abaixo.

$$Acuracia = \frac{(TP + TN)}{(FP + FN + TP + TN)} \quad (1)$$

$$Precisao = \frac{TP}{TP + FP} \tag{2}$$

$$Revocacao = \frac{TP}{TP + FN} \tag{3}$$

$$F1 - Score = 2X\frac{(PrecisaoXRevocacao)}{(Precisao + Revocacao)} \tag{4}$$

Como a F1-Score, abrange duas importantes métricas, a F1-Score será a métrica principal de avaliação. Essa métrica poderá nos auxiliar a descobrir a eficiência da rede neural desenvolvida em identificar os mutantes equivalentes e os mutantes redundantes.

6. Contribuições esperadas

Ao final do projeto de mestrado, espera-se avançar o estado da arte na área de Teste de Software, propondo uma abordagem mais eficiente para aplicação do critério Teste de Mutação utilizando aplicações de AM, mais específico, o uso de Redes Neurais.

Dentre as principais contribuições que esse trabalho pode desencadear, é possível destacar um modelo preditivo que auxilie na redução do custo da aplicação do Teste de Mutação juntamente com estudos científicos que corroboram para a eficiência da abordagem. Tal contribuição pode derivar uma série de desdobramentos para o contexto da aplicação do teste de mutação, como na redução do número de mutantes gerados, melhora na eficiência da aplicação do critério e uma nova abordagem para aplicação do teste de mutação com aplicação de AM.

7. Comparação com trabalhos relacionados

Existem alguns trabalhos na literatura que podem se relacionar com o tema desenvolvido por este trabalho. Um trabalho que pode ser destacado é de [Durelli et al. 2019] que traz uma revisão sistemática sobre o tema de aplicar algoritmos de AM para teste de software. É dito que na área de teste de software está ocorrendo uma grande crescente em aplicar algoritmos de AM com objetivo de automatizar e otimizar a área, porém ainda existem poucos estudos que aplicam AM para acelerar o teste de mutação.

Outros trabalhos que também buscam reduzir o custo do teste de mutação com a aplicação de aprendizado de máquina são os trabalhos de [Márki 2019] e [Chekam et al. 2018]. O trabalho de [Márki 2019] é um trabalho em desenvolvimento que busca a redução do custo minimizando o conjunto de mutantes sem remover nenhum mutante dominante, o que irá evitar a criação de testes apenas para matar os mutantes redundantes. Já o trabalho de [Chekam et al. 2018] busca a redução em identificar os mutantes 'reveladores de defeitos' que são os mutantes que são mais prováveis de serem mortos.

Referências

Ammann, P., Delamaro, M., and Offutt, J. (2014a). Establishing theoretical minimal sets of mutants. In *ICST*, pages 21–30. cited By 72.

Ammann, P., Delamaro, M. E., and Offutt, J. (2014b). Establishing theoretical minimal sets of mutants. In *Proceedings of the 2014 IEEE International Conference on Software Testing, Verification, and Validation*, ICST '14, page 21–30, USA. IEEE Computer Society.

Ammann, P. and Offutt, J. (2008). *Introduction to software testing*. Cambridge University Press.

Bertolino, A. (2007). Software testing research: Achievements, challenges, dreams. In *Future of Software Engineering*, pages 85–103.

Chekam, T. T., Papadakis, M., Bissyandé, T. F., Le Traon, Y., and Sen, K. (2018). Selecting fault revealing mutants. *Empirical Software Engineering*, pages 1–54.

Delamaro, M., Jino, M., and Maldonado, J. (2017a). *Introdução ao Teste de Software*. Elsevier Editora Ltda.

Delamaro, M. E., Barbosa, E. F., Vincenzi, A. M. R., and Maldonado, J. C. (2017b). Introdução ao Teste de Software – Capítulo 5 - Teste de Mutação. In *Introdução ao Teste de Software*, pages 77–116. Elsevier Editora Ltda.

Delamaro, M. E., Maldonado, J. C., and Jino, M. (2017c). Introdução ao Teste de Software – Capítulo 1 - Conceitos Básicos. In *Introdução ao Teste de Software*, pages 1–8. Elsevier Editora Ltda.

Delamaro, M. E., Maldonado, J. C., and Mathur, A. (1996). Proteum-a tool for the assessment of test adequacy for c programs user's guide. In *PCS*, volume 96, pages 79–95.

Durelli, V. H., Durelli, R. S., Borges, S. S., Endo, A. T., Eler, M. M., Dias, D. R., and Guimaraes, M. P. (2019). Machine learning applied to software testing: A systematic mapping study. *IEEE Transactions on Reliability*, 68(3):1189–1212.

Faceli, K., Lorena, A. C., Gama, J., Carvalho, A. C. P. d. L., et al. (2011). Inteligência artificial: Uma abordagem de aprendizado de máquina.

Hunter, C. and Strooper, P. (2001). Systematically deriving partial oracles for testing concurrent programs. In *Australian Computer Science Communications*, pages 83–91.

Madeyski, L., Orzeszyna, W., Torkar, R., and Jozala, M. (2014). Overcoming the equivalent mutant problem: A systematic literature review and a comparative experiment of second order mutation. *IEEE Transactions on Software Engineering*, 40(1):23–42.

Márki, A. (2019). Towards minimal mutation analysis: Using the approximated dominator set of mutants.

Myers, G. J. (2006). *The art of software testing*. John Wiley & Sons.

Papadakis, M., Kintis, M., Zhang, J., Jia, Y., Traon, Y. L., and Harman, M. (2019). Chapter six - mutation testing advances: An analysis and survey. In Memon, A. M., editor, *Advances in Computers*, volume 112 of *Advances in Computers*, pages 275 – 378. Elsevier.

Wong, W. and Mathur, A. P. (1995). Reducing the cost of mutation testing: An empirical study. *Journal of Systems and Software*, 31(3):185 – 196.