MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

Teste de modelos ambientais desenvolvidos via TerraME

RELATÓRIO FINAL DE PROJETO DE INICIAÇÃO CIENTÍFICA - RENOVAÇÃO DE BOLSA
(PIBIC/INPE/CNPq)

Leoni Augusto Romain da Silva
(Centro Universitário Salesiano São Paulo, Bolsista PIBIC/CNPq)
E-mail: augustoromain@gmail.com

Valdivino Alexandre de Santiago Júnior
(LABAC/COCTE/INPE, Orientador)
E-mail: valdivino.santiago@inpe.br

SÃO JOSÉ DOS CAMPOS, SP
junho/2018

RESUMO

A área temática *Modelagem do Sistema Terrestre e Projeção* do Centro de Ciência do Sistema Terrestre (COCST/INPE) objetiva pesquisar a representação do Sistema Terrestre (ST), abrangendo não somente as dimensões físicas e biológicas, como também as dimensões humanas. Existem diversas ações de pesquisa sólidas em relação a essa área temática do COCST/INPE, sendo que uma delas é o TerraME: um ambiente de desenvolvimento para a modelagem dinâmica espacial que apóia o conceito de Autômatos Celulares Aninhados (Nested-CA). Assegurar que os modelos ambientais estejam consistentes/corretos, é uma tarefa bastante desafiadora pois requer o conhecimento no domínio de aplicação, além do conhecimento da linguagem de programação em que o código-fonte do modelo foi escrito. Por outro lado, as metodologias, técnicas e processos, da Engenharia de Software podem contribuir para melhorar a qualidade de um produto de software. A área de Verificação e Validação (V&V) da Engenharia de Software almeja contribuir para essa melhoria da qualidade. Teste de software é, muito provavelmente, o processo mais adotado, na prática, entre todos relacionadas à V&V. Os objetivos específicos desse projeto são: a.) investigar diversas técnicas para geração de casos de teste de software para modelos ambientais desenvolvidos via TerraME; b.) realizar uma comparação estatística rigorosa para identificar quais das técnicas, usadas para geração de casos de teste para os modelos TerraME, obtiveram melhor custo e efetividade. Esse relatório apresenta as atividades desenvolvidas no período de 01 de agosto de 2016 a 30 de junho de 2018.

1.) INTRODUÇÃO

O uso de modelagem ambiental é um esforço que cada vez mais é utilizado por pesquisadores para analisar fenômenos e processos voltados ao meio ambiente. Muitos são os fatores que levam a utilizar processos de modelagem, como a facilidade de simulação via esses modelos, mesmo em casos de sistemas complexos.

A área temática *Modelagem do Sistema Terrestre e Projeção* do Centro de Ciência do Sistema Terrestre (COCST/INPE) objetiva pesquisar a representação do Sistema Terrestre (ST), abrangendo não somente as dimensões físicas e biológicas, como também as dimensões humanas. Existem diversas ações de pesquisa sólidas em relação a essa área temática do COCST/INPE, sendo que uma delas é o TerraME: um ambiente de desenvolvimento para a modelagem dinâmica espacial que apóia o conceito de Autômatos Celulares Aninhados (Nested-CA) [Carneiro 2006].

O TerraME é capaz de apoiar três paradigmas de modelagem: modelagem dinâmica do sistema, autômatos celulares e modelagem baseada em agentes. O ambiente de desenvolvimento TerraME é apresentado na Figura 1. Os principais componentes do TerraME são:

- O interpretador TerraME, que é a parte fundamental de todo o ambiente, pois é responsável pela execução do código-fonte do modelo, escrito na linguagem de modelagem TerraME (uma extensão da linguagem Lua), e também para chamar funções do framework TerraME;

- O framework TerraME, um conjunto de módulos escritos na linguagem C ++ que fornece funções e classes para a modelagem dinâmica espacial. Tal framework também se conecta a bancos de dados espaciais TerraLib;

- TerraView, uma aplicação de Sistema de Informação Geográfica (SIG) desenvolvida sobre a biblioteca C++ TerraLib para o gerenciamento de banco de dados espacial [Câmara et al. 2000];

- Um editor de texto ou um ambiente de desenvolvimento integrado, como o Eclipse, que fornece sintaxe de destaque para a linguagem de programação Lua [Ierusalimschy et al. 1996] e, portanto, para o código-fonte de modelo TerraME.

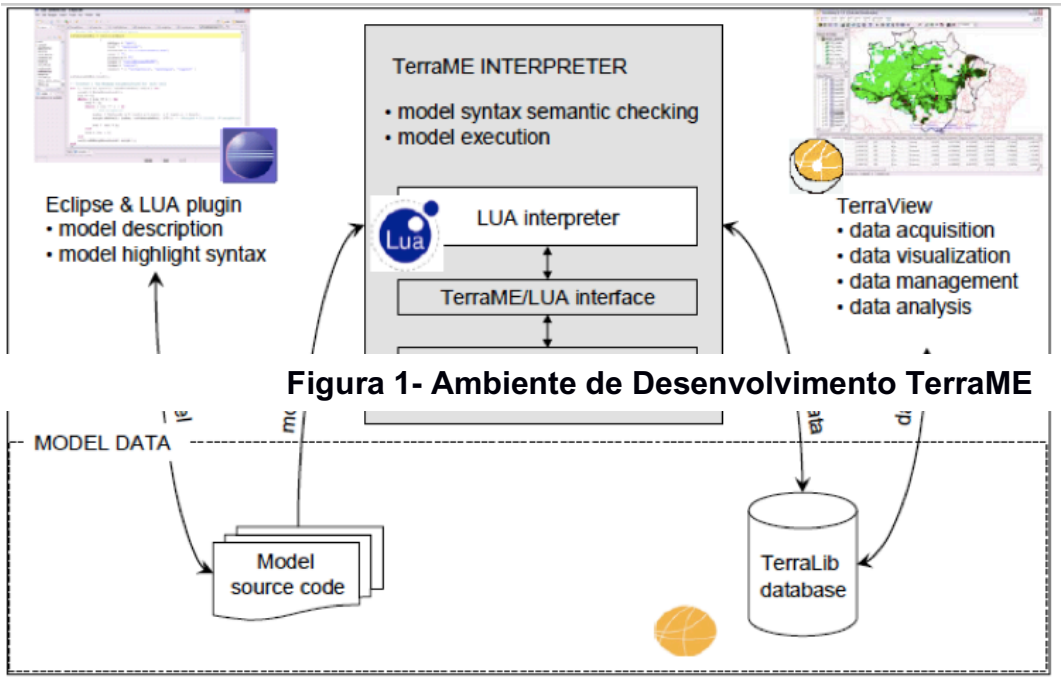


Figura 1- Ambiente de Desenvolvimento TerraME

Assegurar que os modelos ambientais estejam consistentes/corretos, é uma tarefa bastante desafiadora pois requer o conhecimento no domínio de aplicação, além do conhecimento da linguagem de programação em que o código-fonte do modelo foi escrito. Notar que um modelo incorreto pode resultar em uma análise, pelo especialista, inadequada onde, por exemplo, estimativas incoerentes de áreas desmatadas em florestas brasileiras ou de queimadas ocorrendo no nosso território podem ser divulgadas para a sociedade. Portanto, é de fundamental relevância tentar se certificar que os modelos ambientais desenvolvidos via TerraME estejam coerentes.

Por outro lado, as metodologias, técnicas e processos, da Engenharia de Software podem contribuir para melhorar a qualidade de um produto de software. A área de Verificação e Validação (V&V) da Engenharia de Software almeja contribuir para essa melhoria da qualidade. Teste de software é, muito provavelmente, o processo mais adotado, na prática, entre todos relacionadas à V&V. O objetivo de testar um produto de software é encontrar defeitos no código-fonte do mesmo. Inúmeras teorias, metodologias, abordagens têm sido propostas e/ou usadas para as diversas atividades do processo de Teste de software. Uma das atividades do processo de Teste mais estudada, mas que ainda apresenta diversos desafios, é a **geração/seleção de casos de teste**. No fundo, dado que a execução de teste exaustivo não é viável, a ideia é utilizar de formas para selecionar, de infinitas possibilidades, um conjunto de dados de entrada de teste do domínio de entrada de um

programa P, de forma a detectar o maior número possível de defeitos. Existem diversas abordagens para esse propósito, como particionamento por classes de equivalência, teste aleatório [Anand et al. 2013], designs combinatoriais/teste de interação combinatória [Balera e Santiago Júnior 2015][Balera e Santiago Júnior 2016][Balera e Santiago Júnior 2017], Testes Baseados em Modelos [Utting e Legeard 2007][Santiago Júnior 2012], teste de mutação [Delamaro et al. 2007], entre outras.

Portanto, é muito interessante investigar como essas técnicas para geração de casos de teste de software, usualmente aplicadas a produtos desenvolvidos no escopo da Engenharia de Software, podem ajudar na detecção de problemas/defeitos de modelos ambientais. Por exemplo, designs combinatoriais [Mathur 2008] são um conjunto de técnicas de geração de casos de teste de software que buscam a seleção de um pequeno número de casos de teste, uma vez que o domínio de entrada, e o número de subdomínios em suas partições, é largo e complexo. Essa técnica tem-se mostrado eficiente na revelação de defeitos por meio da interação de várias variáveis de entrada. Desse modo, algoritmos para gerar casos de teste de software via designs combinatoriais, por exemplo via a técnica de Matriz de Cobertura com Níveis Variados (MCNV) [Balera e Santiago Júnior 2015], podem ser usados onde as variáveis do modelo ambiental seriam os fatores que são entradas para o algoritmo. Em TBM, pode-se pensar em um meta-modelo, ou seja, um modelo comportamental de um modelo ambiental, para gerar casos de teste de software. Tal meta-modelo pode ser elaborado em diversas linguagens/modelos formais tais como Statecharts [Santiago Júnior 2011] [Santiago Júnior e Vijaykumar 2012][Santiago Júnior et al. 2012], Máquinas de Estados Finitos [Endo e Simão 2013], e Sistemas de Transição que apóiam o método de Verificação Formal chamado Model Checking [Baier e Katoen 2008][Clarke e Emerson 2008][Queille e Sifakis 2008][Fraser et al. 2009]. Portanto, tal investigação pode ser muito frutífera para diminuir os defeitos dos modelos ambientais desenvolvidos via TerraME, e agregar valor ao produto como um todo.

Dado que diversas técnicas para geração de casos de teste serão consideradas nesse projeto, é também muito relevante realizar uma comparação estatística rigorosa, via experimento controlado ou quasiexperimento [Zannier et al. 2006], para identificar quais das técnicas, usadas para geração de casos de teste para os modelos ambientais, obtiveram melhor custo e eficiência. A ausência de estudos de avaliação mais rigorosos na área de Teste de software é um fato comprovado por recentes publicações, não somente na comunidade brasileira, mas na comunidade internacional de Engenharia de Software [Lemos et al. 2013]. Desse modo, um dos resultados importantes do projeto é determinar, entre as técnicas selecionadas para geração de casos de teste, que tiveram melhor custo (geração de um menor conjunto de casos de teste) e melhor eficiência (habilidade de detectar defeitos no código-fonte que representa o modelo ambiental TerraME).

Como uma contribuição secundária desse projeto, também é valioso realizar análise estática do código-fonte dos modelos ambientais desenvolvidos via TerraME (por exemplo, repositórios do projeto Modelos de Autômatos Celulares, Modelos Espaciais baseados em Agentes). Análise estática de código-fonte é um processo automatizado de revisão de código para avaliação do mesmo, ou outra representação do sistema, sem executá-lo [Chess e West 2007]. Portanto, isso difere de Teste onde é necessária a execução do programa. Semelhante a inspeções manuais, análise estática automática é uma técnica em que o código-fonte do produto (modelo TerraME, nesse caso) é verificado a procura de padrões especiais que são automaticamente classificados como potencialmente defeituosos. Como exemplo, partes do código-fonte que não possuem verificação dos valores de dados de entrada, onde é possível que ocorra uma exceção devido a divisão por zero ou overflow numérico, são potenciais causadores de defeitos quando do uso do software.

Os objetivos específicos desse projeto são:

a.) investigar diversas técnicas para geração de casos de teste de software para modelos ambientais desenvolvidos via TerraME;

b.) realizar uma comparação estatística rigorosa para identificar quais das técnicas, usadas para geração de casos de teste para os modelos TerraME, obtiveram melhor custo e efetividade.

Esse relatório apresenta as atividades desenvolvidas no período de **01 de agosto de 2016 a 30 de junho de 2018.**

2.) CRONOGRAMA DE ATIVIDADES E ETAPAS CONCLUÍDAS

Conforme mostrado no “Formulário para Solicitação de Bolsa PIBIC”, a metodologia a ser empregada para atender aos objetivos do projeto está descrita a seguir.

1. Estudar a fundamentação teórica relativa ao projeto. Especificamente, se familiarizar com os conceitos relacionados à Teste de software, linguagem de programação Lua, modelagem ambiental, TerraME, e análise estática de código-fonte;

2. Investigar diversas técnicas para geração de casos de teste de software (teste aleatório, designs combinatoriais, Testes Baseados em Modelos) para modelos ambientais desenvolvidos via TerraME (por exemplo, repositórios do projeto Modelos de Autômatos Celulares, Modelos Espaciais baseados em Agentes);

3. Executar os casos de teste gerados pelas diversas técnicas para geração de casos de teste e analisar os resultados das execuções. Para isso, será ampliado o uso do Jenkins, um servidor de automação para construção, entrega e automação de projetos de software que tem sido adotado para o desenvolvimento do TerraME, para outros

repositórios do projeto tais como Modelos de Autômatos Celulares e Modelos Espaciais baseados em Agentes;

4. Realizar uma comparação estatística rigorosa para identificar quais das técnicas, usadas para geração de casos de teste para os modelos ambientais, obtiveram melhor custo e efetividade. Para isso, será planejado e conduzido um experimento controlado ou quasi-experimento;

5. Realizar análise estática do código-fonte dos modelos ambientais desenvolvidos via TerraME (por exemplo, repositórios do projeto Modelos de Autômatos Celulares, Modelos Espaciais baseados em Agentes);

6. Submeter artigo para conferência e/ou workshop e/ou simpósio na área de Engenharia de Software e/ou Ciência do Sistema Terrestre, e elaborar relatório final de atividades.

O cronograma de atividades pode ser visualizado na Figura 1, cada atividade está de acordo com os números acima, e cada uma das colunas de Ano I e II representa um mês.

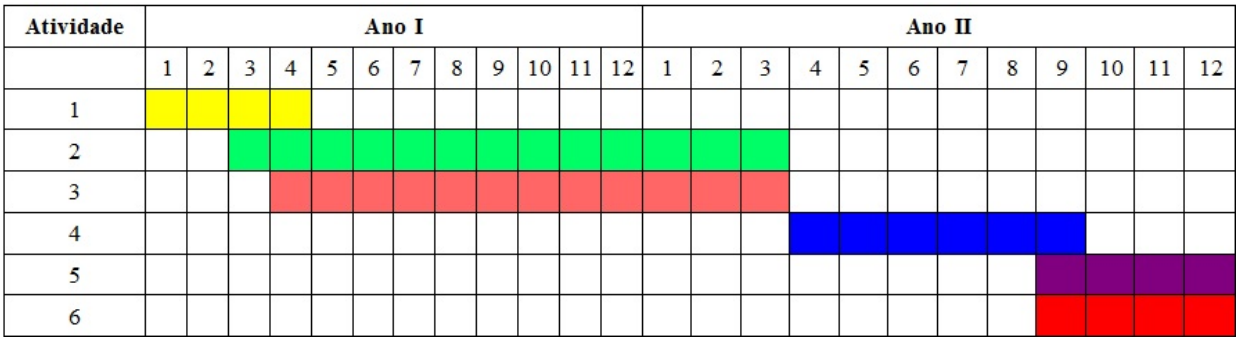


Figura 2 - Cronograma de Atividades

Dessa forma, esse relatório compreende o mês 1 (agosto/2016) do Ano I ao mês 11 (junho/2018) do Ano II do projeto. Considerando as atividades previstas para serem desenvolvidas, conforme a Figura 2, a Tabela 1 exibe as atividades concluídas e a concluir considerando o período referente a este relatório.

Tabela 1 – Etapas Concluídas e a Concluir

	Atividades da Metodologia	Previsão	Realização
1	Estudar a fundamentação teórica relativa ao projeto. Especificamente, se familiarizar com os conceitos relacionados à Teste de software, linguagem de programação Lua, modelagem ambiental, TerraME, e análise estática de código-fonte;	100%	100%

2	Investigar diversas técnicas para geração de casos de teste de software (teste aleatório, designs combinatoriais, Testes Baseados em Modelos) para modelos ambientais desenvolvidos via TerraME (por exemplo, repositórios do projeto Modelos de Autômatos Celulares, Modelos Espaciais baseados em Agentes);	100%	90%
3	Executar os casos de teste gerados pelas diversas técnicas para geração de casos de teste e analisar os resultados das execuções. Para isso, será ampliado o uso do Jenkins, um servidor de automação para construção, entrega e automação de projetos de software que tem sido adotado para o desenvolvimento do TerraME, para outros repositórios do projeto tais como Modelos de Autômatos Celulares e Modelos Espaciais baseados em Agentes;	100%	90%
4	Realizar uma comparação estatística rigorosa para identificar quais das técnicas, usadas para geração de casos de teste para os modelos ambientais, obtiveram melhor custo e efetividade. Para isso, será planejado e conduzido um experimento controlado ou quasi-experimento;	100%	100%
5	Realizar análise estática do código-fonte dos modelos ambientais desenvolvidos via TerraME (por exemplo, repositórios do projeto Modelos de Autômatos Celulares, Modelos Espaciais baseados em Agentes);	75%	100%
6	Submeter artigo para conferência e/ou workshop e/ou simpósio na área de Engenharia de Software e/ou Ciência do Sistema Terrestre, e elaborar relatório final de atividades.	75%	95%

Na Tabela 1 acima, a coluna **Previsão** mostra a porcentagem prevista para a realização da atividade, e a coluna **Realização** mostra a porcentagem realmente realizada da atividade, considerando o período a que se refere essa avaliação/relatório (01 de agosto de 2016 a 31 de junho de 2018). Desse modo, pode-se dizer que todas as atividades previstas para esse período da bolsa foram realizadas de forma satisfatória. As atividades 1 e 4 foram totalmente concluídas (100%), como previsto. A atividade 2 foi bastante evoluída até o momento. A atividade 3 está sendo desenvolvida de forma concomitante à atividade 2, onde os casos/dados de testes estão sendo executados conforme estão sendo gerados. Por outro lado, a atividade 5, que não estava prevista para ser totalmente concluída no período a que se refere esse relatório (a previsão era de 75% de conclusão), foi totalmente (100%) concluída. A atividade 6 está basicamente concluída (95%), pois um artigo relacionado a este projeto de pesquisa foi submetida à conferência de teste de software. Falta apenas o relatório final de conclusão da bolsa. Dado que está sendo feito o pedido de renovação da bolsa, esse relatório final de conclusão será elaborado quando, naturalmente, a bolsa estiver totalmente finalizada.

Na atividade 1, foram realizados estudos sobre a fundamentação teórica relacionada ao projeto, onde ocorreu uma familiarização com conceitos relacionados a teste de software, modelagem ambiental, TerraME, linguagem Lua e análise estática de código-fonte. Foi, inclusive, cursada uma disciplina na Pós-Graduação do INPE, CST-317: Introduction to Earth System Modelling, com o objetivo de se aprofundar em realizar modelagem ambiental usando o TerraME.

Na atividade 2, foi feita uma investigação sobre as técnicas de geração de casos/dados de teste propostas (Teste Aleatório, designs combinatoriais/Teste de Interação Combinatória, Teste Baseados em Modelos), para considerar quais as melhores formas de se aplicar essas técnicas, bem como quais ferramentas e frameworks devem ser utilizados no projeto. Essa atividade é uma das mais

desafiadoras, principalmente para se gerar os resultados esperados dos casos de teste para um determinado conjunto de entradas (parâmetros dos modelos ambientais). A solução adotada foi o desenvolvimento de uma nova metodologia, denominada “Test **Da**ta Generation and **O**racle via Knowledge **B**ase and **M**achine **L**earning” (DaOBML – Geração de Dados e Oráculo de Teste via Base de Conhecimento e Aprendizado de Máquina), para gerar dados de entrada de teste e executar a tarefa do oráculo para modelos ambientais cujas saídas são artefatos complexos, tais como imagens (mapas) ou gráficos. A abordagem sugere várias técnicas de geração de dados de teste (Teste de Interação Combinatória (Combinatorial Interaction Testing - CIT) [Balera e Santiago Júnior 2015][Balera e Santiago Júnior 2016][Balera e Santiago Júnior 2017], Teste Baseados em Modelos (Model-Based Testing - MBT) [Utting e Legeard 2007][Santiago Júnior e Vijaykumar 2012][Santiago Júnior e Silva 2017], Teste Aleatório (Random Testing - RT) [Anand et al. 2013]) e métodos de processamento digital de imagens para criar Bases de Conhecimento. Considerando tais bases e algoritmos de aprendizado de máquina (Machine Learning - ML), um oráculo de teste atribui o veredito de novos dados de entrada de teste. O oráculo de teste propõe seis algoritmos de ML, Rede Neural Artificial (ANN), Árvores de Decisão (DT), Máquina de Vetores de Suporte (SVM), K-Vizinhos Mais Próximos (K-NN), Florestas Aleatórias (RF) e Naive Bayes (NB) [Portugal et al. 2018], para decidir sobre o veredito de novos dados de testes submetidos aos modelos ambientais. Portanto, a atividade 3 (execução dos casos de teste) também foi contemplada na DaOBML. A metodologia é apoiada por uma ferramenta que foi desenvolvida, também chamada DaOBML, e tal metodologia foi aplicada a modelos desenvolvidos via o produto TerraME.

A atividade 4 também foi concluída totalmente. Uma avaliação experimental rigorosa foi realizada, precisamente um experimento controlado, apoiada por procedimentos sistemáticos e testes estatísticos, objetivando responder a diversas questões de pesquisa. Considerando modelos ambientais desenvolvidos via o produto TerraME, esse experimento levou à conclusão de que RT é a abordagem de geração de dados de teste mais viável para o desenvolvimento de Bases de Conhecimento para apoiar o oráculo de teste, as ANNs provaram, novamente, que são uma abordagem interessante da Inteligência Artificial desde que alcançaram o melhor desempenho entre todas as alternativas consideradas, e o tamanho da Base de Conhecimento é importante, onde quanto maior a base, melhor.

A atividade 5, que não estava prevista para ser totalmente concluída no período a que se refere esse relatório, foi totalmente concluída. Assim, foi feita a análise estática de código-fonte de todos os modelos dos pacotes de Autômatos Celulares e modelos Espaciais baseados em Agentes que se encontram dentro do TerraME. A conclusão foi que alguns warnings, descobertos por meio da ferramenta de análise estática LuaCheck, representam conflitos entre a linguagem Lua e o TerraME, que apesar de serem apontados como warnings, não precisam ser modificados pois tal conflito ocorre devido as modificações que o TerraME possui em relação a linguagem Lua. Os resultados também revelaram warnings considerados válidos e que ajudaram a melhorar os modelos dos pacotes de Autômatos Celulares e Baseados em Agentes, reduzindo linhas de código desnecessárias para o funcionamento dos modelos.

A atividade 6 está basicamente concluída. Um artigo foi submetido, e está em processo de avaliação, ao III SIMPÓSIO BRASILEIRO DE TESTE DE SOFTWARE SISTEMÁTICO E AUTOMATIZADO (SAST 2018), que será realizado em setembro de 2018, em São Carlos, SP. O SAST 2018 é um dos simpósios do IX CONGRESSO BRASILEIRO DE SOFTWARE: TEORIA E PRÁTICA (CBSOFT 2018). O SAST 2018 é o principal evento científico em testes de software no Brasil. Seu objetivo é promover um fórum anual para discutir questões sobre a sistematização e automação da atividade de teste de software, promovendo a interação entre pesquisadores e indústria, a fim de fortalecer a cooperação e a inovação nesta importante área de desenvolvimento de software. Conforme mencionado anteriormente, falta apenas o relatório final de conclusão da bolsa. Dado que está sendo feito o pedido de renovação da bolsa, esse relatório final de conclusão será elaborado quando, naturalmente, a bolsa estiver totalmente finalizada.

A seguir, o detalhamento do desenvolvimento dessas atividades será apresentado.

3.) ATIVIDADE 1: FUNDAMENTAÇÃO TEÓRICA

3.1) Teste de Software

As atividades de teste são fundamentais para assegurar a qualidade do produto de software a ser entregue e mantido para os clientes. Testes devem ser planejados, especificados, projetados, construídos e documentados de forma que seja possível repetir ciclos de execução e aumentar sua capacidade em revelar falhas no software. (Lima, 2013)

A Figura 3 apresenta o ciclo de vida da fase de teste:

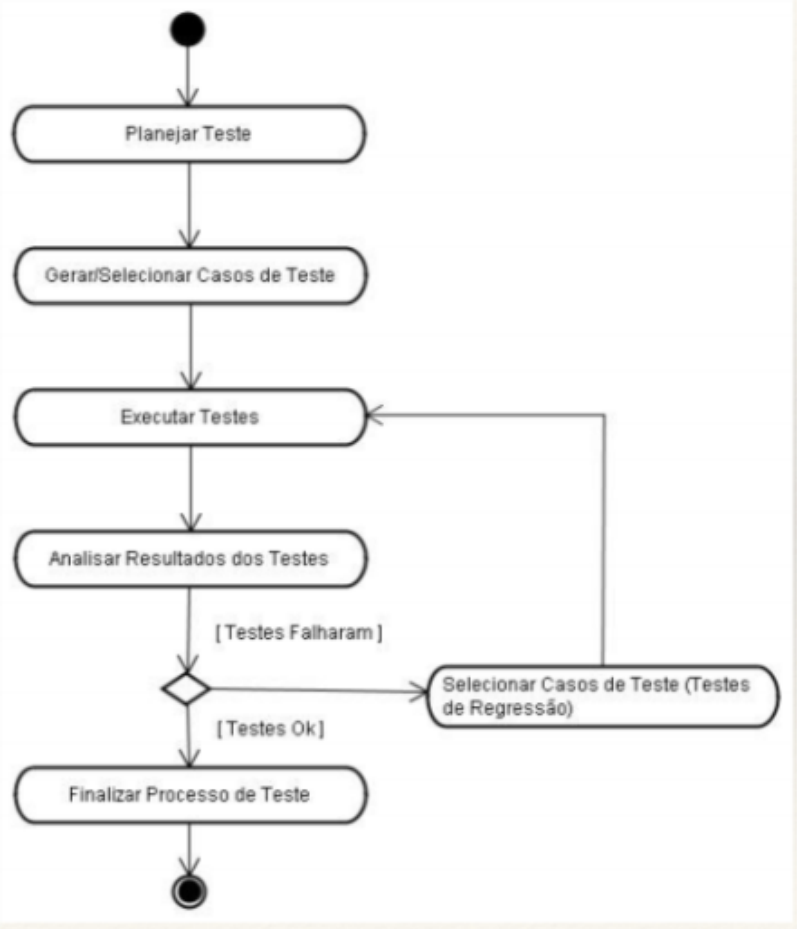


Figura 3 - Ciclo de vida do teste de software. Fonte (Santiago, 2016)

A etapa inicial consiste em planejar os casos de testes, de acordo com objetivos e requisitos presentes na documentação do projeto, cada tipo de aplicação contém características distintas que precisam ser analisadas durante a realização do planejamento de teste.

O passo seguinte é gerar/selecionar os casos de teste e preparar o ambiente de teste que será utilizado para executar os testes posteriormente. Após todos os testes estarem definidos é dado a execução dos testes e seu resultado e armazenado para ser analisado.

A análise consiste em verificar os resultados obtidos dos testes, e compara-los com os resultados esperados e requisitos que foram definidos no planejamento do teste, afim de encontram possíveis falhas e possam ter sido geradas. Caso o teste seja dado como falho é aplicado teste de regressão, que consiste em garantir que não surjam novos defeitos em códigos que já tenham sido analisados.

Após a finalização dos testes e correção das falhas, e feito a finalização do projeto, liberando-o para ser utilizado para seu fim especifico.

3.2) Modelagem Ambiental via TerraME e linguagem Lua

Afim de aperfeiçoar os conhecimentos em linguagem lua, e modelagem ambiental via TerraME, foi realizada a disciplina de introdução à Modelagem do Sistema Terrestre, disponibilizada pelo curso de Pós-graduação de Ciência do Sistema Terrestre (CST-INPE). Dentro do escopo da disciplina, foi desenvolvido como projeto final, um modelo ambiental de autômatos celulares, que aborta o tema sobre vírus influenza e simula sua manifestação dentro de um espaço celular.

Utilizando o ambiente computacional TerraME, foi desenvolvido, em linguagem Lua, uma versão do modelo CA para o vírus Influenza A, onde é apresentado uma nova perspectiva de como o vírus da gripe se manifesta entre as células do corpo humano, e como o organismo reage a essa infecção.

Para realizar a implementação do modelo CA, foram considerados alguns parâmetros, descritos em [Beauchemin et al. 2005], bem como foi definido um diagrama de transição de estados das células. Porém, as mudanças de transição definidas nesse diagrama não são exatamente como descritos no artigo, para que se pudesse chegar a alguns resultados em tempo hábil.

A Tabela 1 mostra os parâmetros usados para o desenvolvimento do modelo CA para o vírus Influenza A, e a Figura 1 mostra o diagrama de transição de estados das células.

Tabela 2 – Parâmetros usados no modelo CA.

Parâmetro	Descrição
qHealthy	Quantidade de células em tempo de execução
qDead	
qImmune	
qInfected	
qInfectious	
healthyLifespan	Tempo maximo de vida útil das células
immLifespan	
infectedLifespan	
infectiousLifespan	
infectiousTime	Tempo maximo para célula *infected* se tornar *infectious*
reviveTime	Tempo maximo para célula *dead* se tornar *healthy*
ageOfHealthy	Idade atual da célula em tempo de execução
ageOfImmune	
ageOfInfected	
ageOfInfectious	
timeToRevive	Mede tempo que uma célula esta no estado *dead*
turnInfectious	Mede quanto tempo falta para célula passar do estado *infected*
infectNeigh	Quantidade de vizinhos no estado *infectious* em tempo de execução

Nesse modelo, as células não foram separadas em grupos e*pithelial* e *immune* contendo cada um seus próprios estados. As células foram tratadas como um único grupo, e podem assumir os seguintes estados: *healthy*, *infectious*, *infected*, *immune* e *dead.* As transições entre os estados ocorrem da seguinte forma:

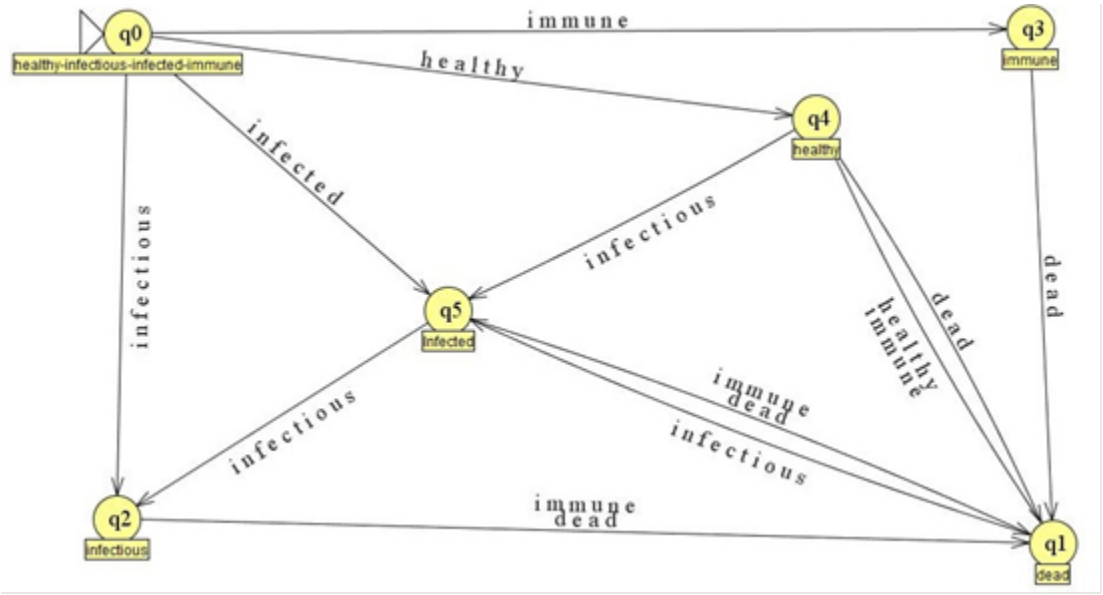


Figura 4 - Diagrama de transição de estados do model CA.

- As células começam aleatoriamente nos estados *healthy*, *infected*, *infectious eimmune*
- As células *healthy* se tornarão *dead q*uando atingirem uma idade maior que a healthyLifespan.
- As células *healthy* se tornarão *infected q*uando sua célula vizinha tiver o estado *infectious* e quando *infectNeigh* tiver valor maior do que 3.
- As células *infected* se tornarão *dead q*uando atingirem uma idade maior que a *infectedLifespan*.
- As células *infected* se tornarão *infectious* quando *turnInfectious* for maior que *infectiousTime.*
- As células *infected* se tornarão *dead* se tiverem uma célula vizinha com o estado *immune.*
- As células *immune* se tornarão *dead* quando atingirem uma idade maior que a *immLifespan*.
- As células *dead* voltarão ao estado *healthy* quando *timeToRevive* for maior que *reviveTime* e seu vizinho tiver o estato *healthy* ou *immune*.
- As células *dead* se tornarão *infectious* quando sua célula vizinha tiver o estado *infected.*

Para analisar a forma como o vírus se comporta, foram criados dois cenários diferentes com uma alternância de alguns atributos como *healthyLifespan*, *immLifespan*,*infectedLifespan ereviveTime.* Esses dois cenários são descritos a seguir.

Caso I

healthyLifespan= 3

immLifespan = 1

infectedLifespan = 3

reviveTime= 5

As Figuras 5 e 6 mostram os resultados para esse cenário. Conforme pode ser visualizado, nesse Caso I, foram considerados valores extremamente baixos para o tempo de vida útil das células e para o tempo que uma célula morta demora para reviver, devido as condições préestabelecidas no modelo. Isso provocou, rapidamente, a morte das células saudáveis e imunes, o que provocou, momentaneamente, o aumento de células mortas.

Porém, as células infetadas (infected) passaram ao estado infeccioso (infectious) na mesma proporção. Devido à condição de o modelo tornar, quando possível, as células mortas em células infecciosas, isso ocasionou, ao final da execução do modelo, a reprodução em massa das células infecciosas. Em outra visão, pode-se deduzir que se um organismo possui imunidade baixa ele se torna mais propenso a adquirir o vírus Influenza A.

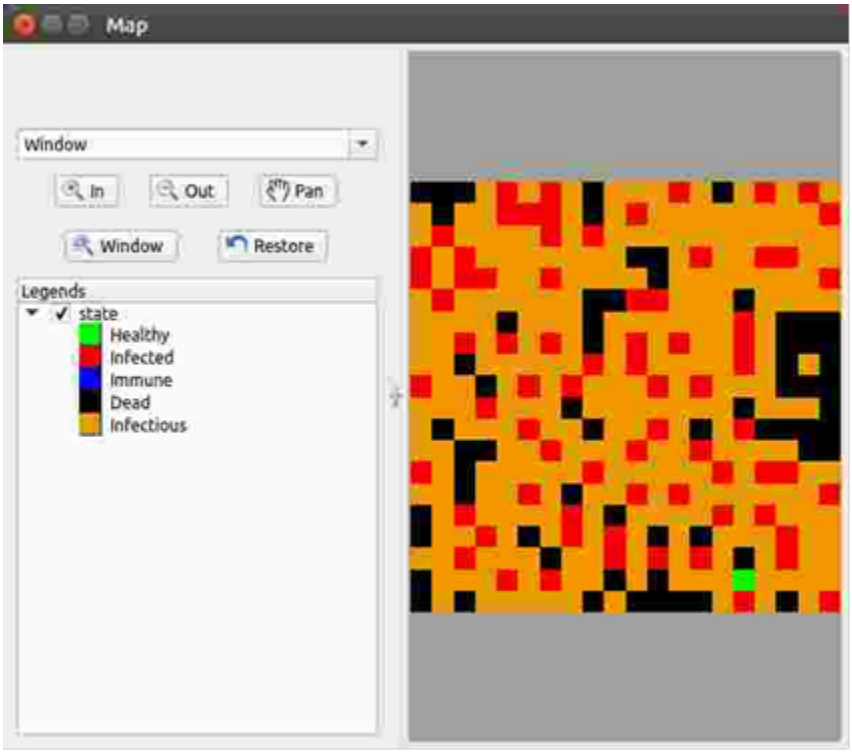


Figura 5 - Mapa do modelo CA utilizado no Caso I

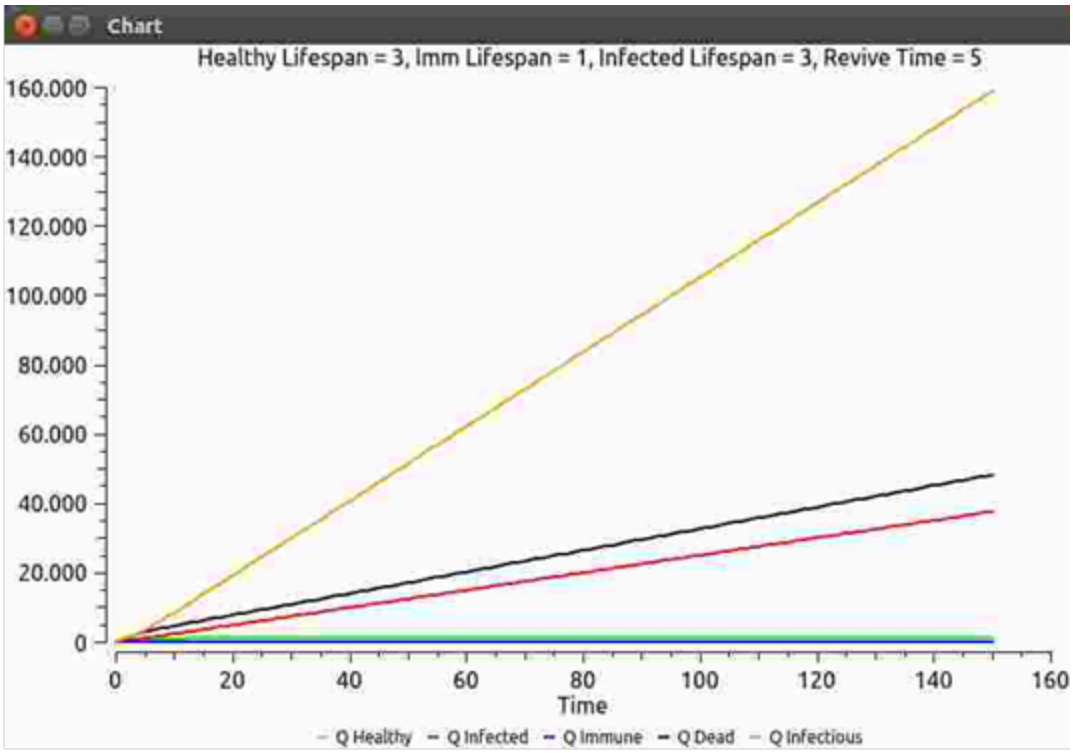


Figura 6 - Resultados do modelo CA utilizado no Caso I

Caso II

healthyLifespan = 70
immLifespan = 95
infectedLifespan = 82
reviveTime = 20

As Figuras 7 e 8 mostram os resultados para esse cenário. O Caso II apresenta resultado divergente ao Caso I. Nesse cenário, o valor das variáveis foi elevado, o que possibilitou as células imunes e saudáveis a se manifestarem por mais tempo, impedindo que as células infecciosas se multiplicarem, já que as células imunes, agora em grande quantidade, conseguem matar essas células infecciosas. Pode-se concluir que o organismo possui uma imunidade maior, e, assim, comprova-se que o vírus da gripe não se manifesta com eficiência no organismo.

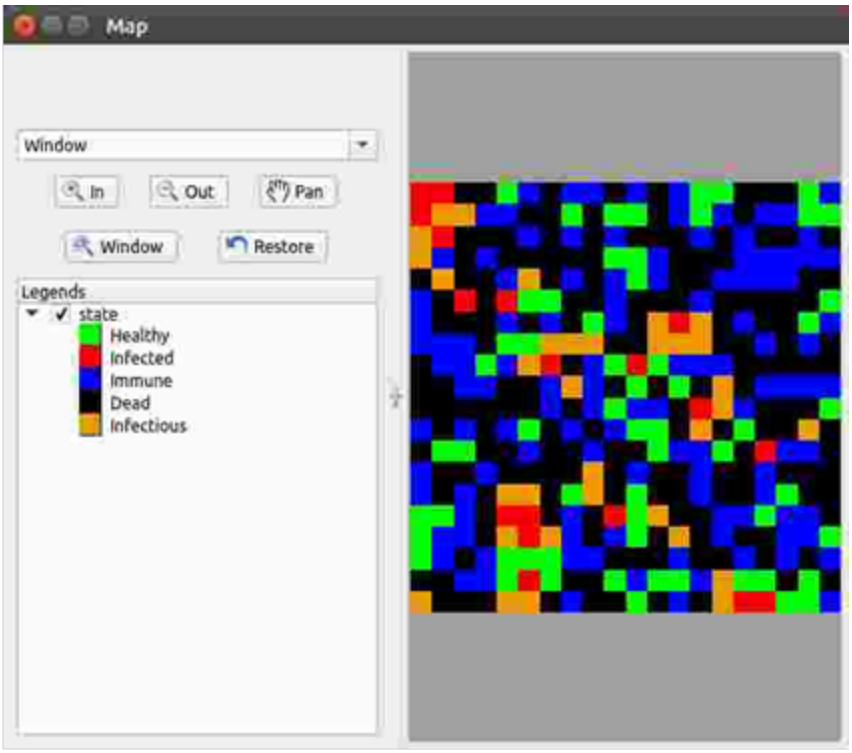


Figura 7 - Mapa do modelo CA utilizado no Caso II

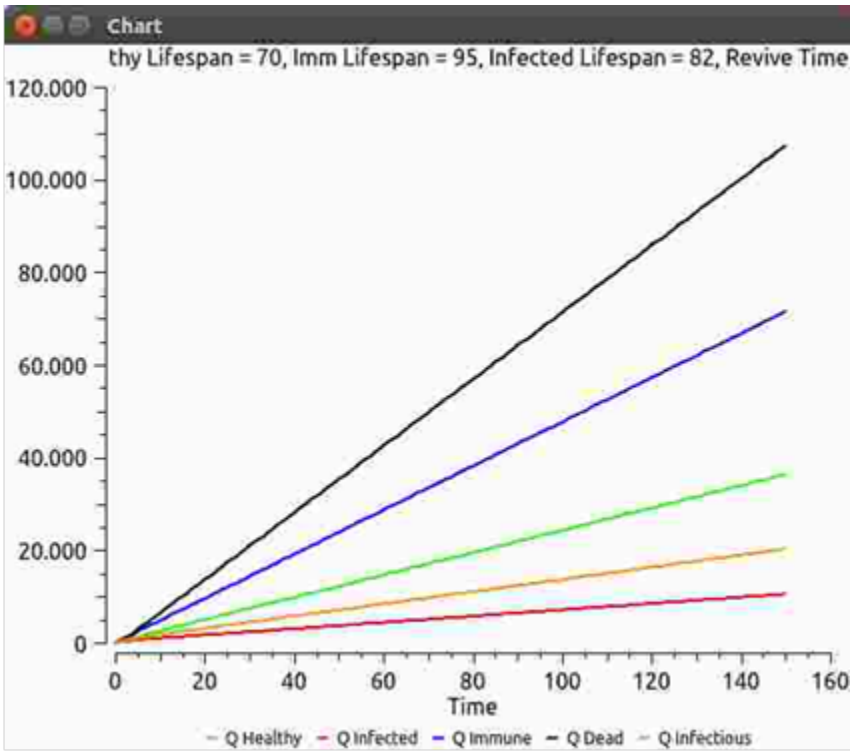


Figura 8 - Gráfico do modelo CA utilizado no Caso II

No decorrer desta atividade, foi possível concluir que o vírus da gripe (Influenza A) se manifesta de forma distinta em cada organismo. No Caso I, o vírus se manifestou em maior escala devido ao organismo possuir uma imunidade muito baixa em relação a quantidade de células que estavam infectadas. No Caso II, ocorreu um efeito contrário, pois o organismo estava com a imunidade consideravelmente alta, o que inibe propagação do vírus.

Para realizar esses casos, foi preciso alterar os valores das variáveis, para ajustar o modelo e encontrar um cenário com valores equilibrados. Sendo assim, o modelo só torna o resultado eficiente quando os valores são manipulados de acordo com o que se espera do fenômeno modelado.

4.) ATIVIDADES 2 e 3: GERAÇÃO DE CASOS DE TESTE E EXECUÇÃO DE CASOS DE TESTE

A metodologia DaOBML é uma abordagem de teste caixa preta, particularmente adequada para os níveis de teste de sistema, aceitação e regressão. A figura 9 apresenta nossa metodologia. A idéia geral por trás do DaOBML é que, se um modelo ambiental (código-fonte) tiver defeitos, as saídas (geralmente algum tipo de imagem) produzidas por esse modelo são incorretas e podemos inferir, com base nesses resultados incorretos, que existem falhas no código fonte.

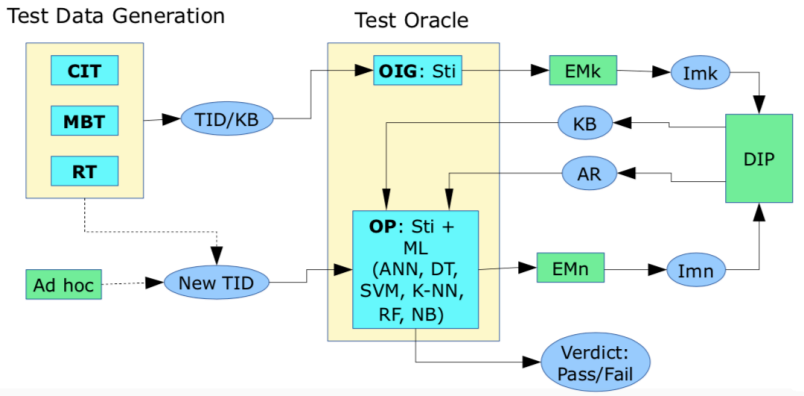


Figura 9 - Metodologia DaOBML, CIT = Combinatorial Interaction Testing; MBT = Model-Based Testing; RT = Random Testing; TID = Test Input Data; KB = Knowledge Base; OIG = Oracle Information Generator; OP = Oracle Procedure; Sti = Stimulation; EM(k/n) = Environmental Models; Im(k/n) = Images; AR = Actual Results; DIP = Digital Image Processing

Vamos explicar nossa metodologia considerando dois fluxos, superior e inferior, onde o fluxo superior precisa ser realizado antes do inferior. No fluxo superior, o primeiro passo é a geração de dados de entrada de teste. Contamos com três técnicas de teste distintas: CIT, MBT e RT. Essas técnicas podem ser usadas isoladamente ou

podem ser combinadas. Mas observe que aqui a idéia de aplicar essas técnicas de geração de dados de teste está relacionada à criação de uma Base de Conhecimento apropriada para que o oráculo possa atribuir um veredicto coerente (veja TID / KB na Figura 9). Assim, inicialmente, a geração de dados de teste ajuda a atividade de avaliação de resultados de teste (problema de oráculo) do processo de teste.

O contexto dessa situação é onde temos um conjunto de modelos ambientais que são confiáveis o suficiente para gerar os resultados esperados. Em DaOBML, a Base de Conhecimento (KB na Figura 9) pode ser realmente considerada como o equivalente dos resultados esperados. Esses modelos confiáveis podem ser indicados como modelos "padrão ouro"(gold standard). Portanto, precisamos testar novas versões de tais modelos com mais recursos ou até mesmo outros modelos ambientais diferentes, mas que possuem características relativamente semelhantes àquelas que criaram a Base de Conhecimento.

Assumimos que não temos imagens suficientes (mapas) na Base de Conhecimento ou não temos nenhuma base. Note-se que esta situação de insuficiente ou mesmo não existência de um conjunto de imagens apropriadas é muito provável de ocorrer quando estamos lidando com novos projetos ambientais de pesquisa ou mesmo com softwares desenvolvidos para novos equipamentos médicos.

Assim, o raciocínio é direcionar a criação, por meio de um procedimento sistemático (estratégia de geração de teste), de uma Base de Conhecimento para suportar a operação do oráculo de teste. Além disso, a ideia de três técnicas de teste distintas é perceber que, eventualmente, para certos tipos de modelos ambientais, por exemplo, uma abordagem baseada em CIT pode gerar uma Base de Conhecimento mais adequada em comparação com MBT e RT.

Depois que os dados de entrada de teste são criados, o componente Oracle Information Generator (OIG) do oracle de teste estimula (Sti) um conjunto de Modelos Ambientais (EMk) com esses dados de teste para, no final, gerar os resultados esperados (Base de Conhecimento). No caso do software de modelagem ambiental, as saídas são, em muitos casos, resultados ou imagens gráficas (Imk), como mapas e gráficos bidimensionais. No entanto, essas imagens precisam ser processadas para que seus atributos possam ser reunidos e, assim, ajudar o oráculo de teste a atribuir os veredictos corretos por meio de abordagens ML.

Observe que as saídas do método Digital Image Processing (DIP) são um conjunto de imagens transformadas. Mas, note que cada entrada da Base de Conhecimento pode ser uma imagem transformada em si, mas também pode ser apenas os recursos extraídos da imagem transformada. Neste último caso, a Base de Conhecimento exige menos espaço de armazenamento. Assim, quando mencionamos uma “entrada” (ou seja, um registro) da Base de Conhecimento, isso significa uma saída da ferramenta de modelagem ambiental (imagem, mapa) que pode ser representada como uma imagem em si ou apenas como suas características.

No fluxo mais baixo, novos dados de entrada de teste (Novo TID) precisam ser submetidos aos modelos ambientais. Observe que DaOBML não sugere nenhuma técnica de geração de dados de teste para esses novos dados de entrada de teste: o profissional pode usar uma estratégia ad hoc ou até mesmo confiar em uma das técnicas (CIT, MBT, RT) usadas para gerar entrada de teste dados para apoiar o desenvolvimento da Base de Conhecimento. As setas tracejadas na Figura 9 significam

esse conjunto de possibilidades que um designer de teste pode ter em sua mão.

O Oracle Procedure (OP) do oracle então estimula os Modelos Ambientais (EMn) com esses novos dados de entrada de teste para produzir outro conjunto de imagens (Imn) que também são processadas pelo método DIP para obter os Resultados Reais (AR). . A mesma observação que acabamos de fazer sobre a Base de Conhecimento, na qual cada registro pode ser uma imagem ou apenas seus recursos, se aplica. Os modelos ambientais aqui considerados (EMn) são geralmente diferentes daqueles usados no fluxo superior (EMk). No entanto, há circunstâncias em que faz sentido que EMn = EMk, quando queremos avaliar a capacidade da metodologia, particularmente relacionada com as forças dos algoritmos ML, para retornar a classe correta (modelo ambiental), desde os novos dados de entrada de teste são completamente diferentes daqueles que foram usados para gerar a Base de Conhecimento.

Com esses dois artefatos, KB e AR, o componente OP usa um dos seis algoritmos ML, conforme mostrado na Figura 2 (ANN, DT, SVM, KNN, RF, NB) para decidir sobre o veredicto dos novos dados de teste. Nós selecionamos tais algoritmos, dada a sua ampla aceitação na comunidade acadêmica como classificadores. O veredicto (Aprovado / Reprovado) é dado pelo componente OP através de uma abordagem ML particular que verifica se uma dada imagem transformada (por exemplo, um mapa), derivada através de novos dados de entrada de teste corresponde a alguma imagem, ik, dentro da Base de Conhecimento. Um modelo ambiental passa por um teste se tal correspondência for encontrada. Caso contrário, dizemos que o teste falha e o modelo ambiental está com defeito. Naturalmente, essa "correspondência" não significa uma igualdade completa entre as duas imagens, in e ik, já que estamos explorando a capacidade dos classificadores (algoritmos ML) de prever a classe correta devido a novos dados de entrada de teste, como foi feito por tanto tempo em aplicativos de reconhecimento de dados / mineração de padrões.

Implementamos uma ferramenta, também chamada DaOBML, para apoiar parcialmente nossa metodologia. A ferramenta executa automaticamente o conjunto de dados de entrada de teste sugeridos por uma técnica específica (CIT, MBT, RT) para orientar a geração das imagens (mapas) que comporão a Base de Conhecimento. Nossa ferramenta implementa algumas etapas do processamento de imagens digitais, como a conversão de imagens coloridas em escala de cinza e o uso do operador Sobel como detector de bordas. Para a conversão em escala de cinza, transformamos uma imagem colorida (mapa) em uma imagem com 16 tons de cinza (escala de cinza de 4 bits). Observe que, embora a escala de cinza de 4 bits seja considerada um antigo sistema de profundidade de cores, ela ainda é adequada para o tipo de imagem produzida pelo software de modelagem ambiental selecionado (TerraME).

O operador Sobel é um método de detecção de bordas baseado em gradiente que encontra bordas usando uma máscara horizontal e uma máscara vertical. Considerando que a convolução é o processo de multiplicar cada valor de intensidade de uma imagem com seus vizinhos locais, ponderados pela máscara, no operador Sobel uma imagem é digitalizada da esquerda para a direita e de cima para baixo da imagem usando as máscaras horizontais e verticais. separadamente. A detecção de bordas é importante porque diminui consideravelmente a quantidade de dados e filtra

informações indesejadas ou insignificantes. Na versão atual da ferramenta DaOBML, todas as imagens inseridas no componente OP do oracle de teste, as que estão dentro de KB e AR, possuem suas bordas detectadas. Para ajudar nas etapas de processamento digital de imagens, levamos em conta a biblioteca JavaCV.

Com relação aos seis algoritmos de ML sugeridos pela nossa metodologia, incorporamos o software de mineração de dados Weka em nossa ferramenta para que o usuário não precise chamá-lo separadamente para conhecer o veredicto dos novos dados de teste. Além disso, observe que cada mapa produzido pela TerraME possui apenas seus recursos, depois de processados digitalmente, armazenados na Base de Conhecimento no formato de entrada principal do software Weka (arff). As imagens (mapas) são criadas e depois processadas (bordas detectadas) são descartadas. A parte inferior da linha é que, na implementação atual do DaOBML, uma Base de Conhecimento é, na verdade, um único arquivo arff no qual concatenamos todas as contribuições parciais dos dados de entrada de teste sugeridas pelas técnicas de geração de teste. O arquivo único arff é criado automaticamente pela nossa ferramenta.

Com a nova abordagem DaOBML, a geração e execução de casos de teste está sendo feito de forma simultânea. As seções 4.1 e 4.2 abordam as atividades de geração de dados de teste. A seção 4.3 aborda a parte de oráculo de teste (que inclui a execução de casos de teste).

4.1) Teste Aleatório

Teste aleatório também conhecido como RT (Random Testing), é uma técnica onde os valores do teste são gerados aleatoriamente, possui uma fácil implementação e é muito usado em testes de segurança para detectar vulnerabilidades que possam existir no sistema, o que torna essa técnica útil também para testes de robustez avaliando como o sistema se comporta com diversas entradas aleatórias inesperadas. (Unicamp, 2014)

Inicialmente, para aplicar o teste aleatório, foi definido utilizar 4 variáveis do modelo Influenza, pois são as que mais influenciam no comportamento do algoritmo, como testar com valores aleatórios apresenta possibilidades infinitas, foi definido como critério de mínimo e máximo valores entre (0,100) , e utilizado a função Random, para alternar entre esses valores, os mesmos são passados por parâmetro dentro de um environment através de uma classe de tese do modelo influenza, que é representado abaixo:

```
require "influenza"

local healthyRandom = math.random(0, 100)
local immLifeRandom = math.random(0, 100)
local InfectRandom = math.random(0, 100)
local reviverandom = math.random(0, 100)

env = Environment{

    teste = Influenza{
        healthyLifespan =  healthyRandom,
        immLifespan = immLifeRandom,
        infectedLifespan = InfectRandom,
        reviveTime = reviverandom}
}

env:run()
```

Figura 10 - classe de teste aleatório do modelo influenza

Posteriormente, o teste aleatório foi aplicado para os modelos do pacote de autômatos celulares do TerraME, assim no modelo Influenza, foram utilizadas as variáveis que mais afetam o comportamento dos modelos, os valores de mínimo

e máximo foram definidos baseando-se nos valores padrões de cada modelo.

```
import "ca"

math.randomseed(os.time())
ft = math.random(10,90)
d = math.random(60,80)

scene = Anneal{

        finalTime = ft,
        dim = d,
}
print(ft)
print(d)
scene:run()
```

Figura 11 - Classe de teste aleatorio do modelo Anneal

```
import "ca"
math.randomseed(os.time())

pc = math.random() + math.random(0, 1)
if(pc > 1) then
    pc = pc/2
end

dc = math.random() + math.random(1, 4)
if (dc > 3.5) then
    dc = dc /2
end

wc = math.random() + math.random(0, 1)
if ( wc > 1.2) then
    wc = wc/2
end

rps = math.random(100,500)
r = math.random(100,500)
ft = math.random(20,90)

scene = BandedVegetation{
    plantCover =pc,
    dryCoeff = dc,
    wetCoeff = wc,
    rainfallPlantSurvival = rps,
    rainfall = r,
    finalTime = ft,
}

scene:run()
```

Figura 12 - classe de teste aleatorio do modelo Banded Vegetation

4.2) Teste Combinatorial

Devido à complexidade dos produtos de software, testar as combinações possíveis de valores de parâmetros de entrada é impraticável. Mathur (Mathur, 2008) define o conceito de designs combinatoriais como um conjunto de técnicas para a geração de casos de teste através da seleção de um pequeno conjunto de casos de teste, mesmo quando o domínio de entrada e os subdomínios de sua partição são grandes e complexos. Ainda, essa técnica tem sido considerada eficaz na detecção de defeitos devido à interação de várias variáveis. (BALERA, 2014)

Inicialmente, para aplicar o teste de designs combinatoriais será utilizado o método de Pairwise, que é muito utilizado por conseguir minimizar a quantidade de casos de testes em situações onde muitas variáveis são consideradas. Para realizar o teste combinatório foi utilizado assim como no teste anterior (teste aleatório) as 4 variáveis mais relevantes, porem dessa vez, cada variável recebeu apenas 4 possíveis valores entre (0,100), que podem ser visualizados abaixo:

Tabela 3 - Tabela com níveis e fatores do modelo

HealthyLifeSpan	immLifespan	infectedLifespan	reviveTime
1	3	2	5
15	25	35	45
7	18	78	90
13	21	99	0

Em seguida foi utilizado a técnica *pairwise test* para gerar menos casos de teste e abranger todos as combinações possíveis, para realizar esse fato, foi utilizado a ferramenta AllPairs, que dado os valores, executa o pairwase test e retorna o resultado, que pode ser visualizado na tabela abaixo:

Tabela 4 - Resultados do teste de Pairwise

Caso	healthyLifespan	immLifespan	infectedLifespan	reviveTime
1	1	3	2	5
2	1	25	35	45
3	1	18	78	90
4	1	21	99	0
5	15	3	35	90
6	15	25	2	0
7	15	18	99	5
8	15	21	78	45
9	7	3	78	0
10	7	25	99	90
11	7	18	2	45
12	7	21	35	5
13	13	3	99	45
14	13	25	78	5
15	13	18	35	0
16	13	21	2	90

É possível aplicar este teste de diferentes formas, uma delas é utilizando uma environment, que possibilita criar vários cenários e executá-los, a grande desvantagem se dá ao fato de que nem todos os modelos possuem a opção de utilizar environments o que torna essa técnica útil apenas para casos específicos.

```lua
require "influenza"
env = Environment{

    Caso1 = Influenza{ healthyLifespan = 1, immLifespan = 3, infectedLifespan = 2, reviveTime= 5},
    Caso2 = Influenza{ healthyLifespan = 1, immLifespan = 25, infectedLifespan = 35, reviveTime= 45},
    Caso3 = Influenza{ healthyLifespan = 1, immLifespan = 18, infectedLifespan = 78, reviveTime= 90},
    Caso4 = Influenza{ healthyLifespan = 1, immLifespan = 21, infectedLifespan = 99, reviveTime= 0},
    Caso5 = Influenza{ healthyLifespan = 15, immLifespan = 3, infectedLifespan = 35, reviveTime= 90},
    Caso6 = Influenza{ healthyLifespan = 15, immLifespan = 25, infectedLifespan = 2, reviveTime= 0},
    Caso7 = Influenza{ healthyLifespan = 15, immLifespan = 18, infectedLifespan = 99, reviveTime= 5},
    Caso8 = Influenza{ healthyLifespan = 15, immLifespan = 21, infectedLifespan = 78, reviveTime= 45},
    Caso9 = Influenza{ healthyLifespan = 7, immLifespan = 3, infectedLifespan = 78, reviveTime= 0},
    Caso10 = Influenza{ healthyLifespan = 7, immLifespan = 25, infectedLifespan = 99, reviveTime= 90},
    Caso11 = Influenza{ healthyLifespan = 7, immLifespan = 18, infectedLifespan = 2, reviveTime= 45},
    Caso12 = Influenza{ healthyLifespan = 7, immLifespan = 21, infectedLifespan = 35, reviveTime= 5},
    Caso13 = Influenza{ healthyLifespan = 13, immLifespan = 3, infectedLifespan = 99, reviveTime= 45},
    Caso14 = Influenza{ healthyLifespan = 13, immLifespan = 25, infectedLifespan = 78, reviveTime= 5},
    Caso15 = Influenza{ healthyLifespan = 13, immLifespan = 18, infectedLifespan = 35, reviveTime= 0},
    Caso16 = Influenza{ healthyLifespan = 13, immLifespan = 21, infectedLifespan = 2, reviveTime= 90}
}

env:run()
```

Figura 13 - Modelo de teste utilizando Environment

Outra forma de aplicar o teste e possuir acesso a todos os parâmetros do modelo é utilizar um cenário que armazenara todos os parâmetros do modelo de forma direta sem utilizar uma environment, essa técnica pode ser utilizada em qualquer modelo, o que a torna mais eficiente para ser aplicar de forma geral.

```lua
require "influenza"

scene1 = Influenza{
    qHealthy = 0,
    qDead = 0,
    qImmune = 0,
    qInfected = 0,
    qInfectious = 0,
    dim = 20,
    finalTime = 50,
    healthyLifespan = 55,
    immLifespan = 55,
    infectedLifespan = 55,
    infectiousLifespan = 64,
    infectiousTime = 20,
    reviveTime = 3,

}
scene1:run()
```

Figura 14 - Modelo de teste utilizando atribuição direta

Posteriormente, para aplicar o teste de design combinatoriais, decidiu-se utilizar o algoritmo T-Tuple Reallocation (TTR) nos modelos de autômato celulares do TerraMe, assim como no teste anterior (teste aleatório), foram utilizadas as variáveis que mais influenciam no comportamento do modelo.

4.2.1) TTR

O T-Tuple Reallocation (TTR) é um algoritmo para a geração de designs combinatoriais através da técnica t-way testing. A construção de cada linha do MCV é feita através da realocação de tuplas afim de que cada caso de teste cubra a maior quantidade de tuplas ainda não cobertas possíveis, através do comprimento de “metas”, que são calculadas através da combinação simples entre o valor do strenght e a quantidade de fatores cobertos pelo caso de teste naquele momento (BALERA,2014).

```
Digite o valor do Strenght:
2
Digite em cada linha o número de níveis de cada fator:

4
4
4
0
1       |         1   1   1   meta: 2
2       |         1   2   2   meta: 2
3       |         1   3   3   meta: 2
4       |         1   4   4   meta: 2
5       |         2   1   2   meta: 2
6       |         2   2   1   meta: 2
7       |         2   3   4   meta: 2
8       |         2   4   3   meta: 2
9       |         3   1   3   meta: 2
10      |         3   2   4   meta: 2
11      |         3   3   1   meta: 2
12      |         3   4   2   meta: 2
13      |         4   1   4   meta: 2
14      |         4   2   3   meta: 2
15      |         4   3   2   meta: 2
16      |         4   4   1   meta: 2
Combinação: [1, 2]
Combinação: [1, 3]
Combinação: [2, 3]
Quantidade de casos de teste final: 16
```

Figura 15 - Saída do algoritmo TTR

4.3) Oráculo de Teste

Após o entendimento das técnicas de teste aleatório e combinatorial para gerar os dados de entrada de teste, o mesmo processo utilizado no modelo influenza foi aplicado para os modelos do pacote de autômatos celulares (CA) do TerraME, afim de gerar uma base de conhecimento com os maps e charts obtidos nos testes, para serem tomados como referência afim de se obter um veredito do caso de teste sem necessitar explicitar os resultados esperados.

Para realizar a extração de características dos modelos optou-se por desenvolver um algoritmo em Java que permite extrair os valores RGB de cada pixel presente em um map do Terrame, através do auxilio de um framework chamado javaCV (utilizado como interface para a biblioteca OpenCV). Cada pixel é verificado e caso possua uma tonalidade de cor semelhante com um dos modelos da base de dados, esse pixel é considerado pelo algoritmo como válido.

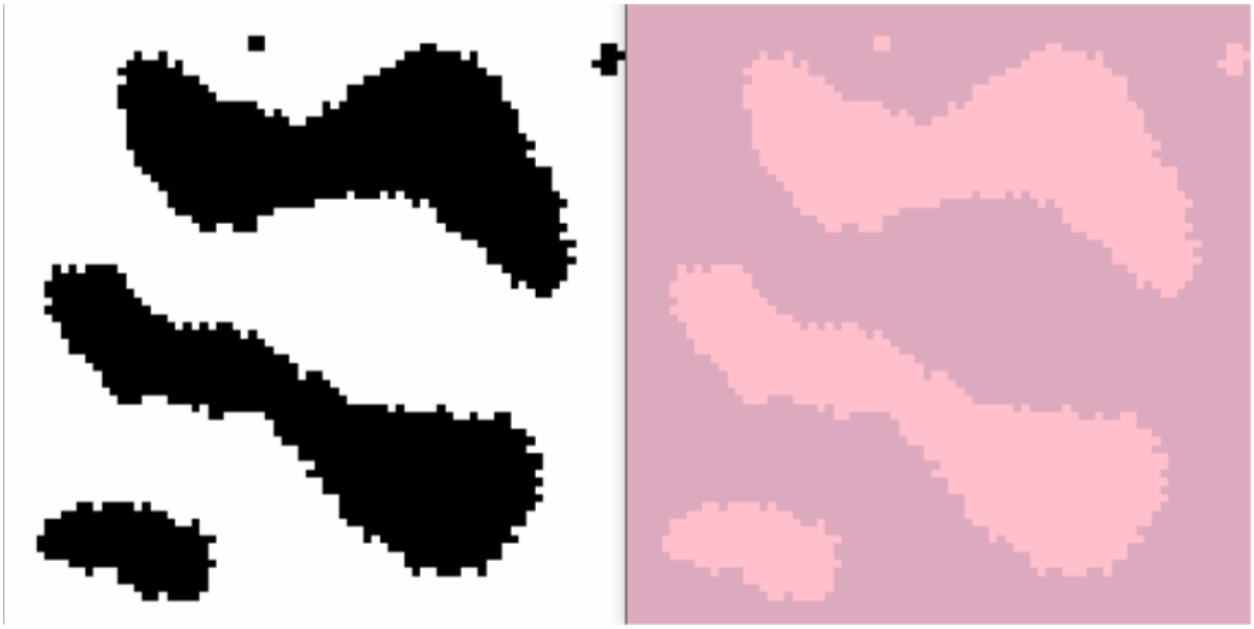


Figura 16 - Modelo Anneal original / modelo extraído

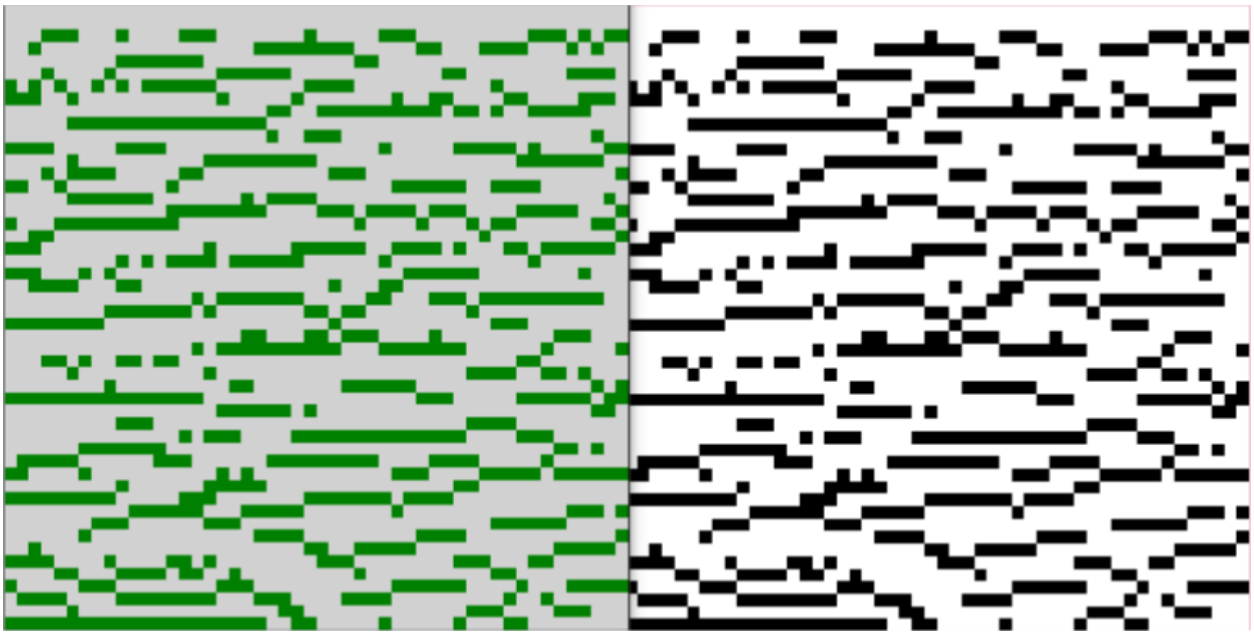


Figura 17 - Modelo Banded Vegetation original / Modelo extraído

As Figuras 16 e 17 mostram respectivamente a esquerda o map do modelo original, e a direita o modelo após a extração dos valores RGB onde cada pixel valido foi pintado de outra cor para que fosse visivel o processo que o algoritmo realiza

Ao realizar o procedimento em todos os maps, os valores RGBs que foram extraidos são enviados para um arquivo .arff afim de gerar um nova base de dados já com valores numericos, para realizar a execução dos casos de testes na atividade 3.

Para obter um possível veredito do caso de teste, foi utilizada a ferramenta WEKA de mineração de dados, para auxiliar no processo de classificação das imagens, utilizando o arquivo .arff que foi criado. Para a manipulação do WEKA foi desenvolvido um algoritmo em java que recebe um nova imagem e realiza a extração RGB para comparar com os dados que já estão na base de dados, através de tecnicas como Naive Bayes, arvore de decisão, entre outras.

4.3.1) Naive Bayes

O algoritmo Naive Bayes é um classificador probabilístico, que estima a probabilidade de um dado, atraves de sua frequencia em que aparece em um determinado conjunto de caracteristicas dentro de um ambiente de treino, quando um novo dado é inserido, o classificador calcula qual a probabilidade dele pertencer a uma classe na base de dados, segundo as caracteristicas de cada classe.

4.3.2) Arvore de decisão

O algoritmo J48 é utilizado para criar arvores de decisão, onde a raiz da arvore ou seja o topo, é onde se localiza o atributo que possui maior significancia em relação aos outros, em seguida os outros nós da arvore recebem os proximos atributos que contem a maior significancia sucessivamente.

4.3.3) SVM

SVM (support vector machine), ou máquina de vetores de suporte em português, é uma técnica de aprendizado supervisionado que é usada para avaliar padrões de um conjunto de dados que são classificados em uma dentre duas categorias.

4.3.4) IBK

O IBK e um método de aprendizagem baseado em instâncias. Esse tipo de algoritmo e derivado do método de classificação k-vizinhos mais próximos (k-nearest neighbor). Porem, este último é um algoritmo não-incremental e tem como objetivo principal manter uma consistência perfeita com o conjunto inicial de treinamento. [Aha et al. 1991].

4.3.5) Random Forest

As árvores de classificação são muito utilizadas em problemas que necessitem classificar instâncias desconhecidas. Como vantagens desse algoritmo estão sua velocidade em relação a outros métodos e sua compreensível representação gráca. Além disto, seu procedimento recursivo top-down permite que o modelo gerado seja facilmente verificado [Inza et al., 2010].

4.3.6) Multilayer Perceptron

As redes neurais artificiais tiveram como inspiração o funcionamento do sistema nervoso dos seres humanos. Os neurônios (células nervosas) transmitem, processam e guardam informações que permitem o funcionamento de sentidos como paladar, olfato, tato, etc. Para realizar estas atividades, o neurônio transmite atividade elétrica através do axônio. Então, por meio dos terminais dos axônios, podem ser realizadas sinapses para os dendritos de outros neurônios. Além disto, o cérebro humano contém bilhões de neurônios trabalhando em paralelo e distribuídos em rede [Dougherty, 2012].

Após os testes realizados com essas tecnicas verificou-se que somente a extração de cores não demonstra um resultado robusto, visto que a cor não é um fator determinante de um modelo, já que um mesmo modelo pode possuir varias cores para seus atributos. Assim, foi proposta investigar uma nova tecnica utilizando extração de bordas (edge detection) das imagens, e assim verificar se os resultados serão superiores ao da extração de cores.

4.4) Sobel

O Sobel utiliza uma técnica de gradiente usando máscaras para detectar bordas tanto horizontais como verticais em uma imagem, que previamente for convertida para escala de cinza, essa tecnica varia em cada imagem, pois depende de fatores como a luminosidade e a quantidade de cores.

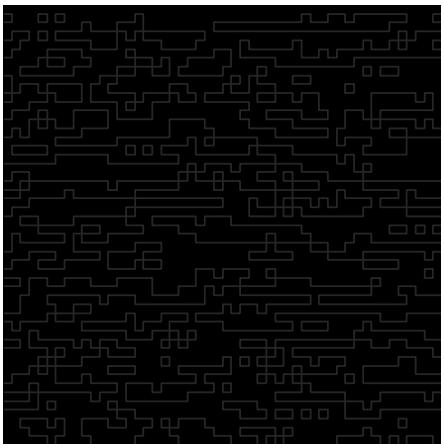


Figura 18 - Resutado SOBEL do modelo BandedVegetation

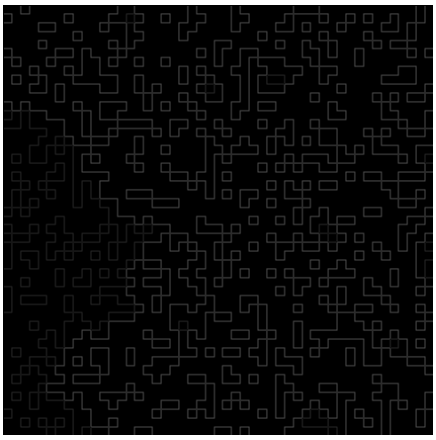


Figura 19 - Resultado SOBEL do modelo Fire

5) ATIVIDADE 4: COMPARAÇÃO ESTATÍSTICA RIGOROSA

O objetivo desta avaliação é avaliar vários aspectos relacionados ao DaOBML. Primeiramente, queremos perceber quão adequadas são as Bases de Conhecimento criadas pelas abordagens de geração de dados de teste (ALLT, CIT, MBT e RT) para que o oráculo de teste possa atribuir veredictos de Positivos Verdadeiros. Para o CIT, selecionamos o algoritmo guloso de realocação T-Tuple (TTR) e a ferramenta Advanced Combinatorial Testing System (ACTS) configurada com o algoritmo guloso IPOG (In-Parameter-Order General). Para o MBT, contamos com a tradução baseada em hierarquia do Statecharts para o método de verificação de modelo e propriedades de padrões de especificação para teste (HiMoST) que gera casos de teste de software por meio da verificação de modelo. Mas, neste caso, começamos diretamente do modelo do NuSMV Model Checker e não do modelo Statechart.

Outra questão é o desempenho dos algoritmos de ML, isto é, que das seis soluções (ANN, DT, KNN, NB, RF e SVM) apresenta veredictos mais corretos. Este é um objetivo muito importante desta avaliação, uma vez que fornecer recomendações de abordagens para os praticantes é bastante aconselhável.

Qualquer estratégia apoiada por uma Base de Conhecimento é naturalmente influenciada por essa base. Por isso, nos perguntamos se o tamanho da Base de Conhecimento influencia o desempenho do oráculo de teste. Criamos dois tipos de bases: uma menor e outra maior. Para cada tipo de base, combinamos essa possibilidade com as técnicas de geração de dados de teste, resultando em 9 Bases de Conhecimento diferentes, conforme mostrado na Tabela 1. Observe que a técnica MBT criou uma única base menor enquanto ALLT significa que a base foi criada combinando os mapas. de todas as abordagens anteriores: CIT / TTR + CIT / ACTS + MBT + RT. Neste caso, a base única do MBT foi considerada para criar as bases maiores ALLT menores e ALLT maiores. Verificamos duplicatas de mapas ao combinar as bases para formar as bases de conhecimento do ALLT. O tamanho da base menor é de cerca de 25 entradas e a maior é de cerca de 250 entradas. Esses números não são iguais para todas as abordagens, devido às características das técnicas de geração de testes e aos modelos ambientais que selecionamos.

TDG	Smaller	Larger
ALLT	✓	✓
CIT/TTR	✓	✓
CIT/ACTS	✓	✓
MBT	✓	
RT	✓	✓

Figura 20 - Bases de conhecimento. TDG = Test Data Generation

Com relação às métricas para avaliar a precisão de nossa estratégia, consideramos os True Positives, ou seja, as instâncias corretamente classificadas. Nosso conjunto de amostras é composto pelos seguintes modelos (programas) do produto TerraME 2.0-RC5. :
(1) 13 modelos do pacote CA: Anneal, Banded Vegetation, Fire, Growth, Life, Oscillator, Parasit, Snow, Parity, Interspecific Competition, Excitable, Wolfram, and Solid Diffusion;

(2) 9 modelos do pacote ABM: Growing Society, Heat Bugs, Labyrinth, Life Cycle, Overpopulation, Single Agent, Sugarscape, Schelling, and Predator Prey.

Como exemplo, a Figura 21 mostra uma saída (mapa) do modelo de Competição Interspecífica do pacote de CA armazenado na base menor de RT. Este modelo mostra como a justaposição de espécies no espaço pode levar a diferentes dinâmicas populacionais entre as espécies competidoras. Este mapa é o resultado final da simulação, onde podemos ver a interação competitiva de cinco espécies de gramíneas: Agrostis stolonifera (células laranja), Holcus lanatus (células verde-escuras), Cynosurus cristatus (células azul-escuras), Poa trivialis (células azuis claras).) e Lolium perenne (glóbulos vermelhos escuros).

Figura 21 - Um mapa de saída do modelo ambiental Interspecific Competition armazenado na base menor do RT

5.1) Perguntas e Variáveis de Pesquisa

Definimos as seguintes perguntas de pesquisa (RQs) para responder:

(1) RQ_1 - Quão adequadas são as Bases de Conhecimento criadas por cada abordagem de geração de dados de teste sugerida por DaOBML para apoiar o oráculo de teste: ALLT, CIT, MBT e RT?
(2) RQ_2 - Qual dos algoritmos ML (ANN, DT, K-NN, NB, RF, SVM) apresenta o melhor desempenho?
(3) RQ_3 - O tamanho da Base de Conhecimento influencia o desempenho do oráculo de teste?

As variáveis independentes são a tecnologia de geração de dados de teste. niques, os algoritmos ML e o tamanho das Bases de Conhecimento. A variável dependente é o número de True Positives obtidos após a aplicação da metodologia DaOBML. Nesta avaliação, um Positivo Verdadeiro significa um veredicto de Passagem emitido pelo oráculo de teste.

5.2) Descrição do experimento

Começamos gerando os dados de entrada de teste para criar as 9 Bases de Conhecimento (consulte a imagem 20). Para ambas as abordagens, CIT / TTR e CIT / ACTS, foram considerados 2 valores por parâmetro para a base menor e 4 valores por parâmetro para a base maior. Em ambos os casos, temos a força (t) igual a 2. Por exemplo, vamos considerar o modelo Banded Vegetation do pacote CA. Um parâmetro deste modelo é o dryCoeff, um coeficiente entre 1,2 e 3,5 para alterar o estado de uma célula para secar. Assim, selecionamos dois valores desse parâmetro para criar as bases menores devido a TTR e ACTS, e 4 valores para as bases maiores. Este modelo possui outros parâmetros (plantCover, wetCoeff, etc.) nos quais seguimos o mesmo

raciocínio. Cada combinação de valores produzidos por um algoritmo CIT, onde cada parâmetro contribui com um valor, é inserida em um modelo ambiental e, portanto, essa combinação resulta em um mapa que será armazenado na Base de Conhecimento, após detectar as arestas. Como as saídas de ambos os algoritmos não são determinísticas, o tamanho preciso das bases (menores, maiores) não é conhecido a priori.

Como mencionamos na Seção 5, o método HiMoST foi o usado como estratégia de MBT. Para cada amostra (modelo ambiental), desenvolvemos um sistema de transição (modelo) diretamente na notação do Verificador de modelo NuSMV, em vez de começar com um modelo Statechart. Seguimos as diretrizes para formalização de propriedades em Computation Tree Logic (CTL) e geração de casos de teste proposta no HiMoST. Selecionamos o Padrão de Ausência e o Escopo Global do Sistema de Padrões de Especificação proposto por Dwyer et al. Com relação ao TR, a escolha dos valores dos parâmetros foi completamente aleatória. Além disso, selecionamos RT puro e não RT adaptativo.

É importante afirmar que nem todos os modelos dos pacotes CA e ABM são adequados para que possamos gerar dados de entrada de teste para eles. Por exemplo, para gerar dados de entrada de teste via algoritmos do CIT com t = 2, o modelo ambiental deve ter pelo menos 3 parâmetros livres para que possamos atribuir diferentes valores (2 ou 4) a eles. Se um modelo não atender a esse requisito, não poderíamos gerar dados de teste por meio dessas abordagens.

Depois de desenvolver todas as 9 Bases de Conhecimento, a próxima atividade foi selecionar novos dados de teste nos quais consideramos 25 novos dados de entrada de teste de acordo com uma abordagem ad hoc (consulte a Figura 9) para enviar a cada modelo ambiental. No entanto, esses 25 conjuntos de valores foram determinados para cada modelo em que foi possível gerar dados de entrada de teste de acordo com uma determinada técnica de teste. Esses novos dados de entrada de teste são completamente diferentes dos sugeridos por cada abordagem de teste para orientar a criação das Bases de Conhecimento. Por exemplo, poderíamos gerar dados de teste para o modelo Banded Vegetation, considerando todas as 9 configurações de técnicas de geração de dados de teste e o tamanho das Bases de Conhecimento. Assim, foram determinados 25 novos conjuntos de valores, que geraram 25 mapas, para o modelo Banded Vegetation e esses valores não são iguais a nenhum dos conjuntos de valores sugeridos pelo CIT, MBT e RT.

Em seguida, os algoritmos ML (ANN, DT, KNN, NB, RF, SVM) entraram em jogo para dar os veredictos dos novos dados de teste. Note que aqui DaOBML sugere a adoção de um único algoritmo ML e não uma combinação de algoritmos ML. O principal raciocínio da atribuição de veredicto é baseado na correspondência "adequada" do mapa devido a novos dados com algum mapa da Base de Conhecimento.

Como mencionamos anteriormente, um Positivo Verdadeiro é, de fato, um veredicto de Passagem de dados de entrada de um novo teste em particular. A atribuição de True Positives é a seguinte. Vamos considerar a base maior do RT, onde todas as 22 amostras criaram entradas nessa base. Vamos considerar um modelo ambiental, e. g. Fogo, e um novo conjunto de dados de entrada (valores) para este modelo que irá gerar um novo mapa, mnf. Assim, esperávamos que todo algoritmo ML

classificasse este mnf como Fire e não como, por exemplo, Banded Vegetation. Para afirmar que um algoritmo ML classificou mnf como Fire, nós olhamos a porcentagem mais alta de instâncias classificadas corretamente, como mostrado na saída de Weka. Assim, esperávamos que o Fire fosse a classe com o maior percentual de instâncias corretamente classificadas entre todos os 22 modelos ambientais. Neste caso, temos um Positivo Verdadeiro. Caso contrário, temos um Falso Negativo que representa inversamente um veredito de Falha. A título de ilustração, na base RT maior, o algoritmo de RF possuía apenas 11 True Positives dos 25 novos mapas para o modelo Fire, apresentando assim 14 False Negatives. Por outro lado, o algoritmo de RNA apresentou 21 Positivos Verdadeiros de 25 e 4 Falso Negativos.

Outra observação sobre a atribuição do veredito é que algumas abordagens que utilizam recursos para auxiliar na análise de similaridade entre as imagens adotam um parâmetro de limiar, que indica a distância máxima aceita para considerar duas imagens semelhantes. Não temos esse parâmetro de limite para ajustar porque, como acabamos de dizer acima, damos o veredicto dos novos dados de teste por uma comparação relativa entre a porcentagem de instâncias classificadas corretamente dos algoritmos ML. Decidimos fazer isso porque uma das principais questões que gostaríamos de responder é sobre o desempenho dos classificadores (algoritmos ML), medidos como dando os veredictos corretos (Pass) dentro do DaOBML, para nos dar uma indicação de qual dos seis As abordagens de ML são melhores para futuros desenvolvimentos da nossa metodologia.

Focamos nossa análise de dados considerando apenas os Positivos Verdadeiros, porque, em nosso contexto, fazer análises com Falso Negativo, como outros fizeram, não nos forneceria informações significativas. Como estamos usando os mesmos modelos ambientais para receber os novos dados de entrada de teste como aqueles para criar as Bases de Conhecimento, esperamos que cada algoritmo ML dentro do oráculo de teste produziria um veredicto Pass (True Positive) para todos os novos dados de entrada de teste. . Além disso, decidimos fazer isso, os mesmos modelos ambientais para criar as Bases de Conhecimento e receber dados de entrada de teste novos e diferentes (EMk = EMn na Figura 9), novamente porque conhecer o desempenho dos algoritmos ML é um dos principais problemas gostaria de resolver. Como temos 25 novos conjuntos de valores, sempre temos essa situação:

$$FN = 25 - TP,$$

onde FN = número de falsos negativos e TP = número de positivos verdadeiros.
Estatisticamente falando, é o suficiente para realizar a análise com True Positives onde quanto maior o seu valor (mediana, média), melhor.

Em relação ao RQ_1, para obter o número de Positivos Verdadeiros para cada modelo ambiental, usamos a seguinte fórmula:

$$em_f = (\sum_{p=1}^{b} \sum_{q=1}^{m} TP_{pq})/b,$$

Figura 22 - formula

Onde emf é um modelo ambiental identificado por f (no nosso caso, temos 1 ≤ f ≤ 15), p identifica uma base relacionada a uma técnica de geração de dados de teste em que o número máximo de bases é b (aqui temos b = 2,4,1,2 para ALLT, CIT, MBT e RT, respectivamente), q é um algoritmo ML e o número máximo de algoritmos é m (no nosso caso, m = 6), e T Ppq é o número de True Positivos devido a um determinado ML q. Observe que o valor máximo de True Positives é 150 (25 × 6) para cada amostra.

Com relação a RQ_2, o valor de True Positives para um modelo ambiental, emf, é simplesmente a soma de True Positives devido a cada algoritmo ML. Neste caso, fomos autorizados a considerar todos os 22 modelos e, como temos 9 Bases de Conhecimento, o valor máximo de True Positives é 225 (25 × 9) para cada amostra. Para responder ao RQ_3, realizamos uma comparação par a par entre as bases menores e maiores de cada estratégia de teste onde aqui o ACTS e o TTR foram considerados isoladamente e levamos em conta 15 modelos (CIT / ACTS, CIT / TTR) e todos os 22 modelos (RT , ALLT). Lembre-se de que, dependendo da estratégia de geração de dados de teste e do modelo, não podemos gerar dados de entrada de teste. Aqui, o valor máximo de True Positives é 150 (25 × 6) para cada modelo ambiental.

Para responder adequadamente a todos os RQs, fizemos uso de avaliação estatística apropriada. Para RQ_1 e RQ_2, temos mais de duas populações e, portanto, usamos o teste de Friedman mais o teste post-hoc de comparações exatas de todos os pares com os métodos de ajuste de valor p de Bonferroni e Holm. Para RQ_3, realizamos uma comparação par a par entre bases menores e maiores. Assim, contamos com o teste dos dois lados do sinal de Wilcoxon ou com sua variação assintótica de dois lados, no caso de empate. Em todos os casos, definimos o nível de significância α = 0,01.

5.3) Validação

Uma das ameaças à validade de conclusão é a confiabilidade das medidas. Nossos resultados estão associados a seis ferramentas / bibliotecas: DaOBML, Weka (incorporado em DaOBML), JavaCV (incorporado em DaOBML), TTR, ACTS e NuSMV. Após todos os passos de processamento, as medidas foram obtidas automaticamente de acordo com os algoritmos ML da ferramenta Weka. Acreditamos que a replicação deste estudo por outros pesquisadores produzirá resultados semelhantes e nosso estudo tem uma alta validade de conclusão.

As amostras do nosso experimento foram modelos ambientais, codificados na linguagem de modelagem do TerraME e, portanto, não tivemos nenhum fator humano / natureza / social nem eventos imprevistos para interromper a coleta das medidas, uma vez que começaram a apresentar uma validade interna. Portanto, nosso experimento

controlado tem uma alta validade interna.

Temos uma ameaça à validade externa relacionada à população, ou seja, quão significativo é o conjunto de amostras usadas no experimento. No geral, investigamos 22 modelos ambientais de dois pacotes do produto TerraME. Esses modelos servem como exemplos de como usar os diversos recursos da linguagem do TerraME e, portanto, é necessário avaliar mais modelos ambientais para generalizar nossos resultados. Mas acreditamos que os resultados desse experimento controlado são interessantes, conforme discutido na Seção 5.4.

5.4) Resultados e Análises

Apresentamos agora e discutimos os resultados da avaliação empírica que conduzimos. Lembre-se de que consideramos os Verdadeiros Positivos (Veredictos de Passe), conforme explicado na Seção 5.2.

5.4.1) RQ_1

Em relação ao primeiro RQ que tem como objetivo responder quão adequadas são as Bases de Conhecimento criadas por cada abordagem de geração de dados de teste para auxiliar o oráculo de teste, tivemos quatro classes de bases: ALLT, CIT, MBT e RT. O resultado do teste de Friedman apresentou um valor de p = 0,0007067, refutando as hipóteses nulas de distribuições populacionais iguais. Os resultados (p-valores) do teste post-hoc de comparações exatas de todos os pares com os métodos de ajuste p-valor de Bonferroni e Holm são mostrados na figura 23 As Figuras 24a e 24b apresentam o boxplot e o meio gráfico, respectivamente.

Comparison	Bonferroni	Holm
CIT × ALLT	0.83025	0.41513
MBT × ALLT	0.00051	0.00051
RT × ALLT	1.00000	0.72257
MBT × CIT	0.16312	0.10875
RT × CIT	1.00000	0.72257
RT × MBT	0.00628	0.00524

Figura 23 - RQ_1 - Comparações exatas de todos os pares, teste post-hoc com os métodos de ajuste p-valor de Bonferroni e Holm

Conforme apresentado em vermelho na figura 23, em dois casos a hipótese nula é rejeitada e há diferença entre as populações: MBT × ALLT e RT × MBT. Isso acontece com os dois métodos de ajuste, Bonferroni e Holm. Como podemos ver nas Figuras 24a e 24b, a técnica de MBT é a pior técnica (menor mediana e média) na comparação pareada com ALLT e RT. Como não há diferença entre quaisquer outros

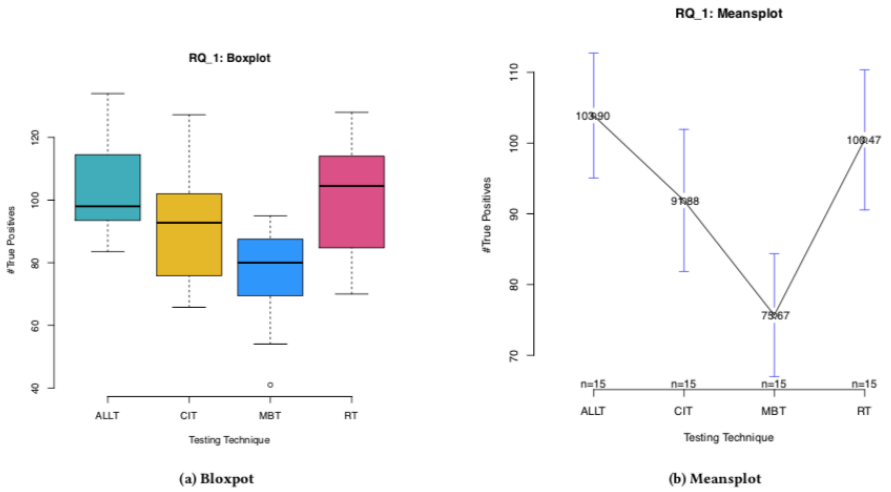
pares de técnicas (incluindo RT × ALLT) e para criar a base ALLT é necessário aplicar todas as três técnicas de teste que exigem mais esforço, concluímos que RT é a abordagem mais adequada para criar um Base de Conhecimento para ajudar no oráculo de teste. De acordo com essa conclusão, o profissional pode simplesmente selecionar um conjunto de valores aleatórios para os parâmetros, o que é uma abordagem mais direta.

Apesar das críticas de RT puro como usamos, como os próximos casos de teste/dados a serem selecionados podem ser menos uniformemente distribuídos sobre o domínio de entrada, porque não faz uso de conhecimento de casos de teste/dados previamente executados, o RT puro foi a melhor solução em nossa avaliação. A simplicidade do RT puro é uma das suas vantagens para fins práticos.

5.4.2) RQ_2

Para responder à questão de qual dos algoritmos ML apresenta o melhor desempenho, primeiro precisamos perceber que temos seis classes, correspondentes aos seis algoritmos ML que o DaOBML propõe: ANN, DT, K-NN, NB, RF e SVM. Repetindo o mesmo procedimento descrito na Seção 5.2, o teste de Friedman resultou em um valor de p = 2.442e-12, novamente rejeitando a hipótese nula. Os resultados (p-valores) do teste post-hoc de comparações exatas de todos os pares com os métodos de ajuste de valor p de Bonferroni e Holm são mostrados na figura 25. As Figuras 27a e 27b apresentam o boxplot e o meio, respectivamente.

Na figura 25, vemos que há diferença estatisticamente significante quando comparamos a RNA com todas as outras abordagens de ML com o método de ajuste de Holm. Considerando o método de Bonferroni, apenas a comparação K-NN × ANN aceitou a hipótese nula (não há diferença) enquanto todas as demais avaliações pareadas envolvendo o algoritmo das RNAs apresentaram diferenças. Mesmo assim, neste caso, o valor p ajustado é um pouco superior (0,0136) do que o nível de significância (0,01). Em relação às demais comparações pareadas, apenas o K-NN × DT apresentou diferença estatisticamente significante.



(a) Bloxpot (b) Meansplot

Figura 14 – RQ_1 - Bloxpot e Meansplot

Comparison	Bonferroni	Holm
DT × ANN	6.8e-13	6.8e-13
K-NN × ANN	0.0136	0.0091
NB × ANN	2.1e-09	1.8e-09
RF × ANN	1.0e-09	9.6e-10
SVM × ANN	0.0013	0.0010
K-NN × DT	0.0052	0.0038
NB × DT	1.0000	1.0000
RF × DT	1.0000	1.0000
SVM × DT	0.0574	0.0344
NB × K-NN	0.1242	0.0580
RF × K-NN	0.0747	0.0399
SVM × K-NN	1.0000	1.0000
RF × NB	1.0000	1.0000
SVM × NB	0.5903	0.1968
SVM × RF	0.4820	0.1928

Figura 25 - RQ_2 - Comparações exatas de todos os pares post-hoc com os métodos de ajuste p-valor de Bonferroni e Holm

A mediana e a média da RNA são as maiores e o K-NN está em segundo lugar, como mostrado nas Figuras 27a e 27b. Com base nesses fatos, podemos afirmar que a RNA é o algoritmo ML que apresentou o melhor desempenho global. Em nosso experimento, utilizamos como RNA o Perceptron Multicamadas clássico com o algoritmo backpropagation para treinamento. Esta RNA clássica vem se mostrando bem sucedida em vários domínios de aplicação devido à sua aprendizagem adaptativa, capacidade de lidar com dados complicados ou imprecisos, entre outros pontos. É um dos classificadores mais conhecidos e, para este experimento, provou ser a melhor solução.

5.4.3) RQ_3

Como afirmamos na Seção 5.2, a fim de responder a essa questão, realizamos uma comparação pareada entre as bases menores e maiores de cada estratégia de teste, onde aqui o ACTS e o TTR foram considerados isoladamente. Portanto, realizamos quatro comparações com os dois tipos de bases: CIT / ACTS, CIT / TTR, RT e ALLT. Detectamos vínculos em todas as quatro avaliações em pares e, portanto, o teste de classificação dos dois lados assintótico Wilcoxon foi selecionado. A figura 26 apresenta os valores p.

Comparison	p-value
CIT/ACTS: smaller × larger	0.07007
CIT/TTR: smaller × larger	6.104e-05
RT: smaller × larger	5.484e-05
ALLT: smaller × larger	9.537e-07

Figura 26 - RQ_3 - Teste de postos sinalizados de Wilcoxon assintótico

Percebemos que apenas no caso do ACTS não há diferença entre as populações. Em todas as outras três comparações, o teste de Wilcoxon determinou que existem diferenças. Em todos os casos, a base maior devido a uma determinada técnica de teste tem sempre a média e mediana mais altas comparadas com a respectiva base menor. Assim, concluímos que quanto maior a Base de Conhecimento, em termos de tamanho, melhor. Embora essa conclusão possa ser considerada óbvia, é importante ressaltar que, em outras comparações rigorosas, trabalhando com sistemas de saída complexos, o tamanho das bases não era tão importante. Assim, decidimos investigar em nosso contexto se o tamanho das bases era realmente relevante e acabou acontecendo.

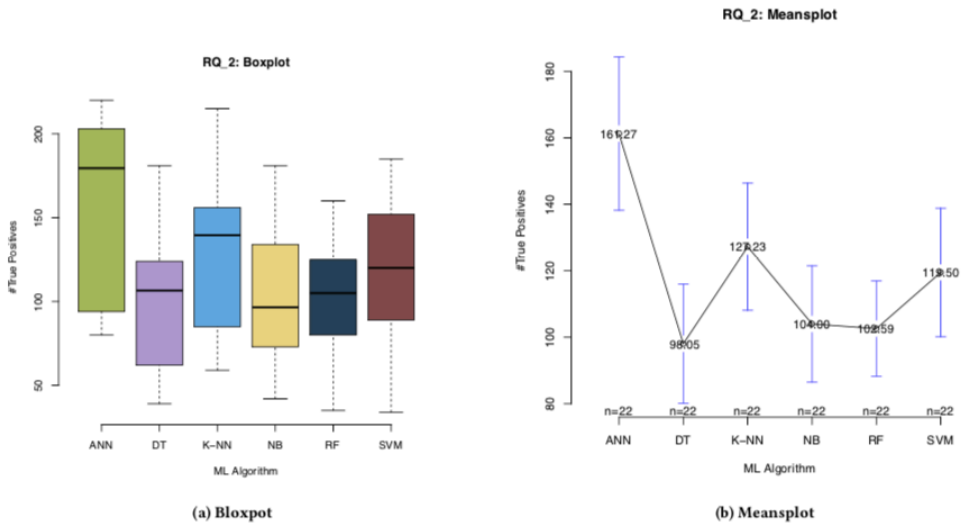


Figura 27 - RQ_2 Bloxpot a Meansplot

6.) ATIVIDADE 5: ANÁLISE ESTÁTICA DE CÓDIGO-FONTE

Para realizar a análise estática foi utilizado a ferramenta LuaCheck (versão 0.18.0), onde todos os modelos dos pacotes de autômatos celulares (CA) e modelos baseados em agentes, foram executados individualmente baseados na versão 2.0-BETA-5 no TerraMe. Optou-se por utilizar uma versão separada do luacheck, pois a versão que se encontra integrada ao TerraMe, não conseguiu executar a análise dos modelos.

Ao realizar uma análise estática utilizando o luacheck em um modelo do TerraME é provável que seja gerado warnings que possivelmente não devem ser considerados validos, isso ocorre devido ao luacheck analisar o código do modelo baseando-se nos padrões de um algoritmo de linguagem lua normal, e não utilizando as nomenclaturas e padrões do TerraME que são desconhecidas pelo analisador, porém ao executar os modelos utilizando o compilador do TerraMe, os mesmos funcionaram corretamente.

- **setting non-standard global variable**

Por padrão uma variável que não é explicitamente declarada como uma variável local, é considerada uma variável global, o luacheck guarda todas essas variáveis globais em uma lista e ao encontrar uma variável fora da listagem produz um warning.

Em modelos do TerraMe, este warning ocorre principalmente quando é utilizado a nomenclatura Model para se inicializar um modelo TerraMe, onde o luacheck considera que o nome do modelo não foi declarado anteriormente, portanto, não é definida como local ou global, o mesmo também ocorre em funções como *countNeighbors, forEachNeighbor,* Porém sua utilização se encontra correta dentro dos padrões de um modelo do TerraME.

Ex: Influenza = Model {}

luacheck considera 'influenza' uma variável global não inicializada por padrão.

- **accessing undefined variable**

  Ao utilizar os diversos componentes do TerraMe como Model, Map, Timer, CellularSpace, Chart, Choice, etc, o luacheck ocasiona um warning pois considera que essas nomenclaturas são variáveis que estão sendo utilizadas sem terem sido definidas e inicializadas, espera-se que as mesmas sejam inicializas no escopo do modelo, porém esse warning pode ser considerado invalido, já que as mesmas não são variáveis comuns, mas tipos de funções do TerraME, e sua utilização esta correta.

  Ex: Influenza = Model {}

  Luacheck considera 'Model' como uma variável que não foi definida anteriormente e nem inicializada com um valor, por padrão variáveis são criadas com o valor nil.

Warings corretos:

Alguns warnings são considerados validos e podem ser corrigidos, como problemas com espaçamento de código e de tabulação desnecessária, mas considerasse muitas vezes que estes espaços são utilizados para tornar o código mais visível e organizado aos olhos do programador.

- **line contains trailing whitespace**

  Este warning pode ser tratado, removendo espaços desnecessário que forma criados em um linha de código utilizando a tecla 'espaço'.

- **line contains only whitespace**

Este waring pode ser tratado, removendo as quebras de linhas desnecessárias usadas dentro do código do modelo.

- **inconsistent indentation (SPACE followed by TAB)**

  Este warning ocorre quando é pressionado a tecla TAB no final de uma linha, e pode ser tratado removendo essa tabulação desnecessária.

A analise estática mostrou-se eficiente para detectar erros nos modelos de Autômatos celulares e baseados em agentes do TerraME, melhorando as sintaxe dos modelos, bem como reduzindo linhas de código com espaçamento e tabulação desnecessárias, alem de revelar supostos warnings que na verdade são trechos de códigos que utilizam a sintaxe do TerraMe, e que podem ser desconsiderados, não são realmente warnings considerados validos.

7.) ATIVIDADE 6: SUBMISSÃO DE ARTIGO

Conforme mencionado previamente, um artigo foi submetido, e está em processo de avaliação, ao III SIMPÓSIO BRASILEIRO DE TESTE DE SOFTWARE SISTEMÁTICO E AUTOMATIZADO (SAST 2018), que será realizado em setembro de 2018, em São Carlos, SP. O SAST 2018 é um dos simpósio do IX CONGRESSO BRASILEIRO DE SOFTWARE: TEORIA E PRÁTICA (CBSOFT 2018). O SAST 2018 é o principal evento científico em testes de software no Brasil. Seu objetivo é promover um fórum anual para discutir questões sobre a sistematização e automação da atividade de teste de software, promovendo a interação entre pesquisadores e indústria, a fim de fortalecer a cooperação e a inovação nesta importante área de desenvolvimento de software. Conforme mencionado anteriormente, falta apenas o relatório final de conclusão da bolsa. Dado que está sendo feito o pedido de renovação da bolsa, esse relatório final de conclusão será elaborado quando, naturalmente, a bolsa estiver totalmente finalizada.

8.) CONCLUSÃO

Esse relatório apresentou as atividades desenvolvidas no período de 01 de agosto de 2016 a 30 de junho de 2018 relacionadas ao projeto “Teste de modelos ambientais desenvolvidos via TerraME”. Os objetivos previstos para o período a que se refere esse relatório foram alcançados satisfatoriamente. Uma nova metodologia foi proposta, DaOBML, para identificar defeitos em modelos ambientais desenvolvidos via TerraME. A metodologia está sendo implementada com apoio do framework javaCV e do ambiente de mineração de dados WEKA.

Foi utilizado três técnicas para a criação das bases de conhecimento (Teste aleatório, teste combinatorial, teste baseado em modelos), para analisar as bases foram usados seis classificadores de aprendizado de máquina (Naive Bayers, J48, IBK, SVM, Random Forest, MLP).

Considerando modelos ambientais desenvolvidos através do produto TerraME, um experimento controlado nos faz concluir que o RT é a abordagem de geração de dados de teste mais viável para desenvolver as Bases de Conhecimento para apoiar o oráculo de teste, as RNAs provam novamente que é uma abordagem interessante de Inteligência Artificial o melhor desempenho dentre todas as alternativas consideradas, e o tamanho da Base de Conhecimento é importante para que quanto maior, melhor.

Mesmo que, inicialmente, variações de escala e iluminação não sejam um problema para nós, precisamos aprimorar as etapas de processamento digital de imagens de nossa metodologia e ferramenta. Técnicas de filtragem podem ser usadas, assim como podemos considerar outros detectores de borda, como a própria RNA e métodos de efeito de área parcial. Também precisamos realizar outros experimentos controlados, aumentando o número de modelos ambientais, bem como avaliando modelos que não são muito parecidos com os que criaram as Bases de Conhecimento, para que possamos generalizar nossos resultados. Como qualquer outra abordagem de caixa preta, a localização de falhas é um problema dentro do DaOBML, e nosso objetivo é trabalhar nessa direção, onde podemos confiar em algoritmos ML supervisionados ou não supervisionados para essa finalidade. Além disso, tentaremos desenvolver essa estratégia de localização de falhas de maneira genérica, para que possa ser adequada a várias linguagens de programação usadas para modelagem ambiental.

8.) REFERÊNCIAS

[Aha et al. 1991] Aha, D. W., Kibler, D., e Albert, M. K. (1991). Instance-based learning algorithms. Machine Learning, 6(1):37–66

[Carneiro 2006] CARNEIRO, T. G. S. Nested-ca: a foundation for multiscale modelling of land use and land cover change. 2006. 114 p. (INPE-14702-TDI/1227). Tese (Doutorado em Computação Aplicada) - Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2006. Available from: <http://urlib.net/sid.inpe.br/mtc-m17@80/2007/01.03.11.57>. Access in: 2016, June 21.

[Câmara et al. 2000] CÂMARA, G.; SOUZA, R. C. M.; PEDROSA, B. M.; VINHAS, L.; MONTEIRO, A. M. V.; PAIVA, J. A.; CARVALHO, M. T.; GATTASS, M. TerraLib: Technology in support of GIS innovation. In: WORKSHOP BRASILEIRO DE GEO-INFORMÁTICA, 2., 2000, São Paulo. Anais... São José dos Campos: INPE, 2000. p. 126-133.

[Dougherty, G. (2012)] Pattern recognition and classication: an introduction. Springer Science & Business Media.

[Ierusalimschy et al. 1996] IERUSALIMSCHY, R.;FIGUEIREDO, L. H.; FILHO, W. C. Lua — An Extensible Extension Language. Software: Practice and Experience, v. 26, n. 6, p. 635-652, 1996.

[Balera e Santiago Júnior 2015] BALERA, J. M.; SANTIAGO JÚNIOR, V. A. T-tuple reallocation: An algorithm to create mixed-level covering arrays to support software test case generation. In O. Gervasi, B. Murgante, S. Misra, M. L. Gavrilova, A. M. A. Rocha, C. Torre, D. Taniar, and B. O. Apduhan, editors, Computational Science and Its Applications - ICCSA 2015, volume 9158 of Lecture Notes in Computer Science (LNCS), p. 503-517. Springer International Publishing, 2015.

[Inza, I. 2010] Calvo, B.; Armañanzas, R.; Bengoetxea, E.; Larrañaga, P. & Lozano, J. A.. Machine learning: an indispensable tool in bioinformatics. Em Bioinformatics methods in clinical research, pp. 25--48. Springer.

[Mathur 2008] MATHUR, A. P. Foundations of software testing. Dorling Kindersley (India), Pearson
Education in South Asia, Delhi, India, 2008. 689 p.

[Utting e Legeard 2007] UTTING, M.; LEGEARD, B. Practical Model-Based Testing: A tools Approach. Waltham, MA, USA: Morgan Kaufmann Publishers, 2007.

[Santiago Júnior 2011] SANTIAGO JÚNIOR, V. A. SOLIMVA: A methodology for generating model-based test cases from natural language requirements and detecting incompleteness in software specifications. 2011. 264 p. Thesis (Doctorate at Post Graduation Course in Applied Computing) - Instituto Nacional de Pesquisas Espaciais, São José dos Campos, SP, Brazil, 2011. Avaliable from: <http://urlib.net/8JMKD3MGP7W/3AP764B>. Access in: 2016, June 21.

[Santiago Júnior e Vijaykumar 2012] SANTIAGO JÚNIOR, V. A.; VIJAYKUMAR, N. L. Generating model-based test cases from natural language requirements for space application software. Software Quality Journal, v. 20, n. 1, p. 77-143, 2012. DOI: 10.1007/s11219-011-9155-6.

[Santiago Júnior et al. 2012] SANTIAGO JÚNIOR, V. A.; VIJAYKUMAR, N. L.; FERREIRA, E.; GUIMARÃES, D.; COSTA, R. C. GTSC: Automated Model-Based Test Case Generation from Statecharts and Finite State Machines. In: Sessão de Ferramentas do III Congresso Brasileiro de Software: Teoria e Prática (CBSoft), 2012, Natal-RN. Anais do III Congresso Brasileiro de Software: Teoria e Prática (CBSoft), 2012. p. 25-30.

[Anand et al. 2013] ANAND, S.; BURKE, E. K.; CHEN, T. Y.; CLARK, J.; COHEN, M. B.; GRIESKAMP, W.; HARMAN, M.; HARROLD, M. J.; McMINN, P.; BERTOLINO, A.; LI, J. J.; ZHU, H. An orchestrated survey of methodologies for automated software test case generation, The Journal of Systems and Software, Volume 86, Issue 8, 2013, p. 1978-2001.

[Delamaro et al. 2007] DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. Introdução ao teste de software. Campus-Elsevier, 2007. 408 p.

[Endo e Simão 2013] ENDO, A. T.; SIMAO, A. S. . Evaluating Test Suite Characteristics, Cost, and Effectiveness of FSM-based Testing Methods. Information and Software Technology, v. 55, p. 1045-1062, 2013.

[Baier e Katoen 2008] BAIER, C.; KATOEN, J.-P. Principles of model checking. Cambridge, MA,
USA: The MIT Press, 2008.

[Clarke e Emerson 2008] CLARKE, E. M.; EMERSON, E. A. Design and synthesis of synchronization
skeletons using branching time temporal logic. In: GRUMBERG, O.; VEITH, H. (Ed.). 25 years of model checking. Berlin/Heidelberg, Germany: Springer Berlin/Heidelberg, 2008. v. 5000, p. 196-215. Lecture Notes in Computer Science (LNCS).

[Fraser et al. 2009] FRASER, G.; WOTAWA, F.; AMMANN, P. E. Testing with model checkers: a
survey. Software Testing, Verification and Reliability, v. 19, p. 215–261, 2009.

[Zannier et al. 2006] ZANNIER, C.; MELNIK, G.; MAURER, F. On the success of empirical studies in the international conference on software engineering. In Proceedings of the 28th International
Conference on Software Engineering, ICSE '06, p. 341-350, New York, NY, USA, 2006. ACM.

[Lemos et al. 2013] LEMOS, O. A. L.; FERRARI, F. C.; ELER, M. M.; MALDONADO, J. C.; MASIERO, P. C. Evaluation studies of software testing research in Brazil and in the World: A survey of two premier software engineering conferences. J. Syst. Softw., 86(4):951-969, April 2013.

[Chess e West 2007] CHESS, B.; WEST, J. Secure Programming with Static Analysis. Addison-Wesley Professional, 2007, 624 p.

[BALERA, 2017] BALERA, Juliana Marino. T-TUPLE REALLOCATION: UM ALGORITMO PARA CRIAR MATRIZES DE COBERTURA COM NÍVEIS VARIADOS PARA APOIAR A GERAÇÃO DE CASOS DE TESTE DE SOFTWARE. 2014. 50f. Trabalho de Graduação FATEC de São José dos Campos: Professor Jessen Vidal.

[Beauchemin et al. 2005] Catherine Beauchemin; John Samuel; Jack Tuszynski. A simple cellular automaton model for influenza A viral infections. Journal of Theoretical Biology, v. 1, n. 232, p. 223–234, 2005.

[Lima 2017] Lima, Ciro Grippi Barbosa, 2013. PROCEDIMENTO PARA INTRODUÇÃO DE AGILIDADE EM TESTES DE
SOFTWARE.

[RIOS 2017] RIOS, Emerson, 2013. Clinica sobre conceitos básicos de teste de software.

[Unicamp 2014]Unicamp, 2014. Testes caixa preta. Disponível em: <http://www.ic.unicamp.br/~meidanis/courses/mc626/2014s1/materiais/slides/Aula13-Testes-caixa-preta-combinatoria.pdf> acessado em: 02 jan. 2017.

[Balera e Santiago Júnior 2016] BALERA, J. M.; SANTIAGO JÚNIOR, V. A. 2016. A Controlled Experiment for Combinatorial Testing. In Proceedings of the 1st Brazilian Symposium on Systematic and Automated Software Testing (SAST). ACM, New York, NY, USA, Article 2, 10 pages. https://doi.org/10.1145/2993288.2993289

[Balera e Santiago Júnior 2017] BALERA, J. M.; SANTIAGO JÚNIOR, V. A. 2017. An algorithm for combinatorial interaction testing: definitions and rigorous evaluations. Journal of Software Engineering Research and Development 5, 1 (28 Dec 2017), 41. https://doi.org/10.1186/s40411-017-0043-z

[Santiago Júnior e Silva 2017] SANTIAGO JÚNIOR, V. A.; SILVA, F. E. C. 2017. From Statecharts into Model Checking: A Hierarchy-based Translation and Specification Patterns Properties to Generate Test Cases. In Proceedings of the 2nd Brazilian Symposium on Systematic and Automated Software Testing (SAST). ACM, New York, NY, USA, Article 2, 10 pages. https://doi.org/10.1145/3128473.3128475

[Portugal et al. 2018] I. Portugal, P. Alencar, and D. Cowan. 2018. The use of machine learning algorithmsin recommender systems: A systematic review. Expert Systems withApplications 97 (2018), 205 – 227. https://doi.org/10.1016/j.eswa.2017.12.020