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

sid.inpe.br/mtc-m21c/2019/04.15.15.09-TDI

IDENTIFYING EFFICIENT APPROACHES TO AUTOMATICALLY GENERATE TEST CASES IN MODEL BASED TESTING

Matheus Monteiro Mariano

Master's Dissertation of the Graduate Course in Applied Computing, guided by Drs. Nandamudi Lankalapalli Vijaykumar, and Érica Ferreira de Souza, approved in April 26, 2019.

URL of the original document:
<http://urlib.net/8JMKD3MGP3W34R/3T5LM62>

INPE
São José dos Campos
2019

PUBLISHED BY:

Instituto Nacional de Pesquisas Espaciais - INPE
Gabinete do Diretor (GBDIR)
Serviço de Informação e Documentação (SESID)
CEP 12.227-010
São José dos Campos - SP - Brasil
Tel.:(012) 3208-6923/7348
E-mail: pubtc@inpe.br

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

sid.inpe.br/mtc-m21c/2019/04.15.15.09-TDI

IDENTIFYING EFFICIENT APPROACHES TO AUTOMATICALLY GENERATE TEST CASES IN MODEL BASED TESTING

Matheus Monteiro Mariano

Master's Dissertation of the Graduate Course in Applied Computing, guided by Drs. Nandamudi Lankalapalli Vijaykumar, and Érica Ferreira de Souza, approved in April 26, 2019.

URL of the original document:
<http://urlib.net/8JMKD3MGP3W34R/3T5LM62>

INPE
São José dos Campos
2019

Cataloging in Publication Data

Mariano, Matheus Monteiro.

M337i Identifying efficient approaches to automatically generate test cases in model based testing / Matheus Monteiro Mariano. – São José dos Campos : INPE, 2019.

xvi + 58 p. ; (sid.inpe.br/mtc-m21c/2019/04.15.15.09-TDI)

Dissertation (Master in Applied Computing) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2019.

Guiding : Drs. Nandamudi Lankalapalli Vijaykumar, and Érica Ferreira de Souza.

1. Software testing. 2. Model-based testing. 3. Systematic mapping. 4. Graph-based algorithms. 5. Finite state machine. I.Title.

CDU 004.415.53

Aluno (a): *Matheus Monteiro Mariano*

Título: "IDENTIFYING EFFICIENT APPROACHES TO AUTOMATICALLY GENERATE TEST CASES IN MODEL BASED TESTING"

Aprovado (a) pela Banca Examinadora em cumprimento ao requisito exigido para obtenção do Título de *Mestre* em *Computação Aplicada*

Dr. Thales Sehn Körting

Presidente / INPE / São José dos Campos - SP

( ) Participação por Video - Conferência

(X) Aprovado *( ) Reprovado*

Dr. Nandamudi Lankalapalli Vijaykumar

Orientador(a) / INPE / SJCampos - SP

( ) Participação por Video - Conferência

(X) Aprovado *( ) Reprovado*

Dra. Érica Ferreira de Souza

Orientador(a) / UTFPR / Curitiba - PR

(X) Participação por Video - Conferência

(X) Aprovado *( ) Reprovado*

Dr. Celso Luiz Mendes

Membro da Banca / INPE / São José dos Campos - SP

( ) Participação por Video - Conferência

( ) Aprovado *( ) Reprovado*

Este trabalho foi aprovado por:

( ) maioria simples

(X) unanimidade

São José dos Campos, 26 de abril de 2019

Aprovado (a) pela Banca Examinadora em cumprimento ao requisito exigido para obtenção do Título de ***Mestre*** em

Computação Aplicada

Dr. **André Takeshi Endo**

Convidado(a) / UTFPR / Curitiba - PR

(X) Participação por Video - Conferência

(X) Aprovado ***( ) Reprovado***

Dr. **Luciana Brasil Rebelo dos Santos**

Convidado(a) / IFSP / Jacareí - SP

( ) Participação por Video - Conferência

(X) Aprovado ***( ) Reprovado***

Este trabalho foi aprovado por:

( ) maioria simples

(X) unanimidade

São José dos Campos, 26 de abril de 2019

ACKNOWLEDGEMENTS

First of all, I would like to gratefully thank to God, who guided me in my footsteps. I would like to thank my father Edvaldo dos Santos Mariano, my mother Romilda da Silva Monteiro Mariano, my brothers Rodrigo Monteiro Mariano and Rafael Monteiro Mariano, and all my friends of INPE for their support, encouragement and patience during the time I devoted to this work.

Next, I need to sincerely thank my advisors Dr. Érica Ferreira de Souza and Dr. Nandamudi Lankalapalli Vijaykumar, for their guidance, understanding, and most important, friendship during the development of this work. Without their mentorship, I may never have gotten to where I am today. It was crucial so I could have the strength to go ahead.

I must also express my gratitude to the Brazilian Institute for Space Research (INPE), specially to LABAC, for it valued institutional support and the necessary human and physical resources essential to perform all the research activities.

And finally, my sincere recognition and gratefulness to Coordination for the Improvement of Higher Education Personnel (Coordenação de Aperfeiçcoamento de Pessoal de Nível Superior, CAPES), for providing the funding which allowed me to undertake this research, and all my friends that believed in my work and support me on my crowdfunding, sharing it or financing it, that provided me the experience in presenting this work in Chile.

ABSTRACT

Context: Model Based Testing (MBT) has attracted a lot of attention from researchers since it has proved efficient in using formal models to represent reactive systems' behavior in order to guide test case generation. Such systems are mostly specified and verified using Finite State Machine (FSM), a formal modeling technique commonly used to represent systems' behavior. There is a plethora of test generation algorithms in the literature. Most of them are graph-based once a FSM can be considered as a graph. Nevertheless, there is a lack of studies on analyzing cost and efficiency of FSM-based test generation algorithms. **Objective:** This dissertation aims to investigate and compare graph-based algorithms employed to generate test cases from FSM models. In particular, we compare the Chinese Postman Problem (CPP) and H-Switch Cover (HSC) algorithms with the well-known breadth-first and depth-first search (BFS, DFS) algorithms in the context of covering all-transitions (AT) and all-transition-pairs (ATP) criteria in a FSM. **Method:** First, a systematic literature mapping was conducted to summarize the methods that have been adopted in MBT, considering FSM. Second, the main methods found were implemented and analyzed on random and real-world FSMs that represent embedded systems of space applications. For the evaluation of studies, we considered analyses in terms of cost (time), efficiency (mutant analysis) and coverage of the generated test cases (number of test cases, average length of test cases, largest and smallest test cases, etc.). **Results:** In general, CPP presented the best results with the FSMs used in terms of number of test cases and test suite size. In addition, CPP also presented low distribution of average length compared to other algorithms.

Keywords: software testing, model-based testing, systematic mapping, graph-based algorithms, finite state machine.

IDENTIFICANDO ABORDAGENS EFICIENTES DE BUSCA EM GRAFO PARA GERAÇÃO AUTOMÁTICA DE CASOS DE TESTE PARA TESTE BASEADO EM MODELO

RESUMO

Contexto: Teste Baseado em Modelo (TBM) tem atraído muita atenção de pesquisadores da área de Teste de Software, uma vez que se mostrou eficaz usando modelos formais para representar o comportamento do sistema a fim de orientar a geração de casos de teste. Estes software são geralmente especificados e verificados usando uma técnica de modelagem formal chamada Máquina de Estados Finitos (MEF). Existem diversos algoritmos de geração de casos de teste na literatura. A maioria são baseadas em grafo, uma vez que uma MEF pode ser considerada como um grafo. No entanto, há falta de estudos que analisam o custo e a eficiência de algoritmos de geração de casos de teste baseado em MEF. **Objetivo:** Esta dissertação tem como objetivo investigar e comparar algoritmos baseados em grafo aplicados á geração de casos de teste a partir de modelos MEFs. Em particular, comparamos os algoritmos do Problema do Carteiro Chinês (PCC) e o $H - SwitchCover$ com os algoritmos clássicos da literatura, Busca em Largura e Busca em Profundidade (BL e BF), no contexto de cobrir uma MEF com os critérios todas-as-transições (TT) e todos-os-pares-de-transições (TPT). **Método:** Primeiro, um mapeamento sistemático da literatura foi conduzido para sumarizar os métodos que tem sido adotados em TBM, considerando as MEFs. Segundo, os principais métodos foram implementados e analisados com MEFs aleatórias e reais que representam sistemas embarcados de aplicaÇões espaciais. Para a avaliação dos estudos, foram considerados análises em termos de custo (tempo), eficiência (análise de mutante) e cobertura dos casos de teste gerados (número dos casos de teste, tamanho médio dos casos de teste, os maiores e menores casos de teste, etc.). **Resultados:** No geral, o PCC apresentou os melhores resultados para as MEFs usadas em termo de número dos casos de teste e tamanho da suíte de teste. Além disso, o PCC também apresentou baixa distribuição do tamanho médio comparado aos outros algoritmos.

Palavras-chaves: teste de software, teste baseado em modelo, mapeamento sistemático, algoritmos baseados em grafo, máquina de estado finito.

LIST OF FIGURES

Page

LIST OF TABLES

Page

CONTENTS

Page

1 INTRODUCTION

With the growing evolution of computing, the development of software has become more complex to meet the demands for new and efficient technologies. Today, newest techniques that solve specific problems come up frequently, both for facilitating the development of applications and delivering quickly software with more quality. In the context of software development companies, software engineers need to invest more in looking for methods, techniques or tools to ensure the quality in their software and set up a competitive edge with their competitors.

In order to assure software quality, several efforts in academia and industry have been dedicated for more refined test strategies (SOUZA et al., 2017). The mission of National Institute for Space Research (INPE) is to produce science and technology in space activities and terrestrial environment, offering products and services that benefit the nation. Most of the products developed at INPE need software in systems considered critical and complex. The systems developed at INPE are considered critical because, if they present some defect, even simple ones, it may lead to serious damage or loss both financially and socially. For example, if a software that sends an alert of landslide fails, its alert will not be duly delivered to its intended audience and there might be several negative consequences. Similarly, if a software embedded in a satellite fails, the ground stations may not receive proper information to be used in several applications, causing financial losses and affecting research that need such data. So, space missions require organized and manageable activities of testing in the development of reliable software (ARANTES et al., 2014).

In such scenarios, activities related to Verification, Validation and Testing (VV&T) have been used throughout the software development process. These activities analyze the compliance of the developed software with the specification (MYERS; SANDLER, 2004). So, software testing can be defined as an execution process of a program with intention of finding errors (MYERS et al., 1976). During the testing activity, a model can be created to guide the test process, analyzing the stimulus (or input values) to verify whether the behavior of implemented software is in conformance with its specification. The inputs are known, and they are triggered on the implementation. Then, after applying the input values on original software, it is necessary to observe whether the output values of the software implementation are according to the specified model. According to El-Far and Whittaker (2001), software testing requires the use of a model to guide efforts on generation and selection of test cases. In this context, Model-Based Testing (MBT) has attracted a lot of attention of re-

searchers in the area, since there are several advantages in using models to represent the behavior of a system to guide generation of test cases (UTTING; LEGEARD, 2010; ZANDER et al., 2017). MBT can bring some benefits, such as high rate of failure detection, reduction of cost and time, traceability, and ease of updating software requirements (UTTING; LEGEARD, 2010).

One of the main features of MBT is the automated generation of test cases from a formal test model. In this sense, several formal techniques have been investigated to specify the system behavior as a test model, such as Statecharts (HAREL, 1987), Finite State Machine (FSM) (LEE; YANNAKAKIS, 1996), Specification and Description Language (SDL) (ELLSBERGER et al., 1997) and Labelled Transition Systems (LTS) (TRETMANS, 2008). Each modeling technique has specific methods and criteria to traverse a model and automatically generate test cases. However, depending on the nature of the system, the test sequence generated can be very large or even infinite, exceeding capacity of exercising the generated test cases on the implementation (BLACK, 2008; RAPPS; WEYUKER, 1985).

1.1 Motivation

Currently, the Laboratory of Computing and Applied Mathematics (LABAC) at INPE works with formal models for software specification. The Laboratory has developed a tool called WEB-PerformCharts that generates steady-state probabilities based on Markov Chains represented in Statecharts and also generates test cases when the system is modeled in Statecharts or FSMs (VIJAYKUMAR, 1999; ARANTES et al., 2013; SOUZA et al., 2008).

FSM is a formal modeling technique normally used for testing due to its rigor and simplicity. FSM considers states, inputs, outputs and transitions between states. When an input (or event) is stimulated, a transition occurs generating a change from a state to another and optionally, an output. Also, one can say that a FSM represents the behavior of a reactive system due to the fact that changes of states are based on stimulus (PINHEIRO et al., 2014; PONTES et al., 2014).

Among the different methods to generate test cases from FSM, one can cite: Transition Tour (TT), Unique Input/Output Sequence (UIO), Distinguishing Sequence (DS) (SIDHU; LEUNG, 1989) and Switch Cover (PIMONT; RAULT, 1976). The Switch Cover method, in particular, also known as edge-pair coverage, defines that all adjacent pairs of transitions are covered at least once and it is based on graph-based algorithms (AMMANN; OFFUTT, 2008). Switch Cover is one of the oldest methods

and has been investigated by different research groups (PIMONT; RAULT, 1976; MARTINS et al., 1999), including INPE (ARANTES et al., 2013; SOUZA et al., 2008). A new version of this method, H-Switch Cover (HSC), was recently created (SOUZA et al., 2017). The main change of HSC method is the performance and the capacity in dealing with more complex FSM.

Literature shows the efforts of the scientific community in the analysis of cost and efficiency of test cases generated from different models (SOUZA et al., 2008; AO et al., 2009; DOROFEEVA et al., 2010; ENDO; SIMAO, 2013; SOUZA et al., 2017). Also, there is a lot of effort in development and evaluation of methods and criteria that select a set of test cases more efficiently from FSMs. The main motivation of this work is to identify and analyze graph-based algorithms employed to generate test cases from FSMs, consequently contributing to LABAC's research, for example, with the advancement of the WEB-PerformCharts tool, making the tool more updated and relevant .

1.2 Purpose of work

The objective of this dissertation is to compare graph-based algorithms employed to generate test cases from FSMs in the context of MBT. This work is divided in two main steps:

1. In order to determine which algorithms would be analyzed in this study, we conducted a systematic mapping to identify which are the main approaches used in the context of MBT, such as methods, criteria, formal models and evaluation of test cases generated. A mapping study provides a broad overview of a research area in order to determine whether there is research evidence on a particular topic. Results of a mapping study help identifying gaps in order to suggest future research (KITCHENHAM, 2007).

2. With the results from systematic mapping, the main algorithms found were analyzed with experiments. In particular, we compare the Chinese Postman Problem (CPP) and H-Switch Cover (HSC) algorithms with the well-known Depth-First Search (DFS) and Breadth-First Search (BFS) algorithms in the context of covering all-transitions (AT) and all-transition-pairs (ATP) criteria in FSMs. The study was conducted on random and real-world MEFs that represent embedded systems of space applications. For the evaluation of studies, we considered the number of test cases, test suite size, average length of sequences, standard deviation and distribution,

generation time and mutant analysis.

1.3 Dissertation Structure

This document is structured as follows. Chapter 2 presents the main underlying concepts of this dissertation, with main concepts of Software Testing, Model-Based Testing, methods and test criteria to FSM and related work. Chapter 3 formally introduces the methods and procedures used to conduct the systematic mapping. In Chapter 4, the experimental study is presented with the methods and criteria identified. Lastly, conclusions, remarks, and future directions are described in Chapter 5.

2 THEORETICAL FOUNDATION

In this chapter, some of the most important concepts in the research areas studied are briefly discussed. Section 2.1 presents the main concepts about Software Testing; Section 2.2 presents the Model-Based Testing approach; Section 2.3 presents the main methods and test criteria from FSM; and Section 2.4 presents the works related to this project.

2.1 Software Testing concepts

Despite the software testing area being old in Software Engineering, there is much debate about which would be the best definition about testing a system. Many definitions about Software Testing were proposed in the literature, but one of the most well-know definitions was proposed by Myers et al. (1976) that defines testing as an execution process of a program with intention of finding errors.

Another definition with different interpretations in Software Engineering is the concepts about Fault, Error and Failure. This work will use the definition given by Institute of Electrical and Electronics Engineers (IEEE) 610.12-1990 (INSTITUTE OF ELECTRIC AND ELECTRONIC ENGINEERS, 1990). According to this standard, the definitions are:

- Fault: is the lowest level, being a software's reaction due to the difference between the value obtained and an expected value. For example, a logic problem.
- Error: the syntactic manifestation of the fault, in which, software moves to a state from an incorrect instruction or command. For example, an object calls a method, but this object is void, so the program will manifest a "Null Pointer Exception".
- Failure: when the software responds differently from the one expected by user. As a result, the program will manifest a fault or error to the user. For example, a calculator should make a sum, but it shows a division result to user.

System problems can occur in several ways, but the main cause is the human error (DELAMARO et al., 2007). To avoid such problems, some activities can help in finding errors before the systems are finalized, such as Validation, Verification and Testing (VV&T). VV&T dedicates efforts to make a software model cohesive with

the implementation and an approach to make this is to check if the system will behave as expected. This activity should not be limited only in the final product, but from the initial conception of the project to secure the quality of software developed (MARUCCI et al., 2002).

VV&T activities can be divided in two types: static and dynamic. Static activities are conducted through technical reviews and inspections, and do not require the execution of code. Dynamic activities involve the code execution, being the main objective inserting input values and to check if the output matches with the expected result. So, testing a software dynamically involves analyzing the behavior of program from a finite set of test cases (INSTITUTE OF ELECTRIC AND ELECTRONIC ENGINEERS, 2004). IEEE defines "test case" as a set of inputs, condition of execution and expected results developed for a specific goal. For example, test a path of a program or check the conformity of a software requirement (INSTITUTE OF ELECTRIC AND ELECTRONIC ENGINEERS, 1990).

2.1.1 Testing levels

Test can be applied in different levels of software. Each level analyzes a specific behavior of a program, and for each level a specific technique can be applied. The three main levels are (PRESSMAN, 2005):

- Unit testing: focuses in identifying faults in small and independent components of software, such as methods, functions or classes.
- Integration testing: responsible in testing the combination of small units of software, avoiding that unforeseen circumstances may lead to a communication error among the units;
- System testing: verifies if the behavior of software is in conformance with requirements, analyzing the internal flow of a program.

2.1.2 Testing techniques

There are several testing techniques that a developer can apply in a testing level. The three main testing techniques are: Structural testing, Functional testing and Fault-Based Testing.

Structural testing has the objective of analyzing the software based on the program code. It checks how the code flows. So, a test group learns about the algorithm, checks

its data flow, creates test cases to check the methods or functions and makes changes in the code based on this analysis. A common representation of this technique is Control Flow Graph (CFG) (ENDO; SIMAO, 2013), which indicates a flow of program by nodes and edges. Nodes represent a code block and an edge is a possible branch from this block code to another. The main criteria used in the Structural testing are:

- all-nodes: requires that all nodes of CFG are traversed at least once. This criterion indicates that all lines of code are executed and covered.
- all-edges: requires that all edges of CFG are traversed at least once. This indicates that all the code is covered and all deflections of system are executed and covered.

Functional testing is a technique that generates tests without the code, analyzing if the behavior of software is in accordance with the specification (MALDONADO, 1991). This technique might employ an abstraction of software, based on its specification, that represents the behavior of system. When an input value is applied in the model, it will produce a specific output. So, a test case is applying a input value set that will produce a specific output. This output will be evaluated and compared to the outputs generated in the final implementation by applying the same input values, thus analyzing whether the system conforms to what was specified. The higher the input set, the better the test result is. But it is impossible to check all the possible inputs. So, it is important to define a input set, based on some criteria, that maximizes all the possibilities. The main criteria for this technique are:

- Partitioning in equivalence classes: partitions the input domain in a finite number of equivalence classes. The tester chooses any input values of a class and assumes that any other value of this class behaves in an equivalent way in the software.
- Boundary value analysis: this is a complement to equivalence partitioning, which focuses on improving the test cases by checking the borders of the class values, analyzing the values below the limit, above the limit and at the limit.
- Cause and effect chart: defines test requirements from the combination of input conditions. A graph represents the relations of cause (input conditions) and effects (actions). This graph is converted into a decision table,

considering a model that charts the situations of system and the combinations of input values to generate test cases.

Fault-Based Testing is a technique that uses the knowledge about the more common failures in the development to define test cases. This knowledge varies in function of used resources (as language and techniques) in the software development. Its main criterion is the mutant analysis. In this criterion, a test case set is analyzed if it can identify the difference of behavior between an original program and a mutant program (DELAMARO et al., 2007). This criterion generates, from a program P, a mutant set P', that is similar to the original program, but has some faults inserted in the code. Each mutant is executed using a test case set. If a mutant shows a different behavior from the original program, and if test case identifies the mutant P' then the mutant is considered "dead". Otherwise, the mutant is considered "alive". But if there is no test case that distinguishes the mutant in the original program, it is considered "equivalent". Therefore, the mutation analysis relates the number of dead mutants with the number of mutants generated minus the equivalent.

2.2 Model-Based Testing - MBT

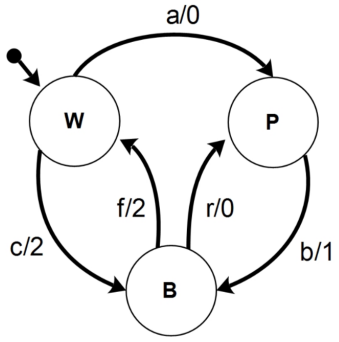
MBT is an approach that represents the behavior of a system as a model (DALAL et al., 1999), using an abstraction of software. Software modeling is important to conduct the test activity because it enables to identify the behavior and its expected result. In MBT, the model can be designed using a modeling technique that needs to be formal, that is, it must have no ambiguity (UTTING; LEGEARD, 2010). And also, the model needs to be accurate to support relevant test cases. Some of the most well-known techniques to represent test models for a system are Finite State Machine (FSM) (LEE; YANNAKAKIS, 1996), Statecharts (HAREL, 1987), Specification and Description Language (SDL) (ELLSBERGER et al., 1997), and Labelled Transition Systems (LTS) (TRETMANS, 2008). Since INPE uses FSMs and Statecharts formal models for software specification and generation of test cases, these models will receive more details in the next subsections.

2.2.1 Finite State Machine - FSM

FSM is a formal model used to represent the behavior of a system based on software's specification. A FSM has a set of states and transition arcs between these states (GILL et al., 1962; LEE; YANNAKAKIS, 1996). Transition arcs have labels known as events (or inputs). When an event is stimulated, a change from one state to another takes place. A state is the representation of a system aspect. A transition is a reaction

of FSM from a specific input value stimulated in a state, that changes the current state to another. A test case (or test sequence) is a path starting from the initial state and returning to itself, listing the input values along this path. A sequence can be obtained by means of a method or test criterion. This sequence must be exercised in the final software to check if the output obtained after running the code is the same specified in FSM model. That is, if the behavior of software is in accordance with the specification.

Figure 2.1 - Example of a FSM with three states (W, P, C) and five transitions (a/0, c/2, b/1, f/2, r/0). Source: (AMARAL, 2005)



The formal notations herein are based on Dorofeeva et al. (2005). Figure 2.1 will be used as an example. Formally, a FSM M is a 7-tuple$(S, s_0, I, O, D, \delta, \lambda)$, where:

- S is a finite set of states with initial state s_0 (W, P and B, with W being the initial state);
- I is a finite set of inputs (a, c, f, r and b);
- O is a finite set of outputs (0, 1 and 2);
- $D \subseteq S \times I$ is a specification domain;
- $\delta : D \to S$ is a transition function; and
- $\lambda : D \to O$ is an output function.

FSMs have some properties that are mentioned below:

- Mealy and Moore machines: the theory from both machines are similar, but the Mealy machine is processed as a finite automaton, which generates

an output for each transition and depends on the current state and its inputs (BRITO et al., 2003), whereas the Moore machine generates a output for each state of machine that can be a null value, and depends only of current state of machine. Despite this, the processing of both machines is the same. In this work, we only use Mealy machines.

- Completely specified: a FSM is completely specified if it treats all possible inputs in all states of the machine, being that the FSM has transitions to all possible types of inputs in a state. Otherwise, it is considered partially specified.
- Strongly connected: a FSM is defined as strongly connected if for each pair of states there is a connection between them. It is considered initially connected if from the initial state it is possible to access all other states of FSM.
- Minimal or reduced: a FSM is defined as minimal if there are no two equivalent states, that is, different inputs which generates same outputs.
- Deterministic and nondeterministic: a FSM is considered deterministic if any state of machine has only one possible transition from an input to another state. Otherwise, it will be nondeterministic, because a state will have more than one transition from the same input, producing several distinct outputs.

2.2.2 *Statecharts*

Statecharts is a formal modeling technique used to specify complex reactive systems (HAREL, 1987). It can be considered as an evolution of state-transition diagrams. It ends up in a cleaner representation of system behavior based on the following features:

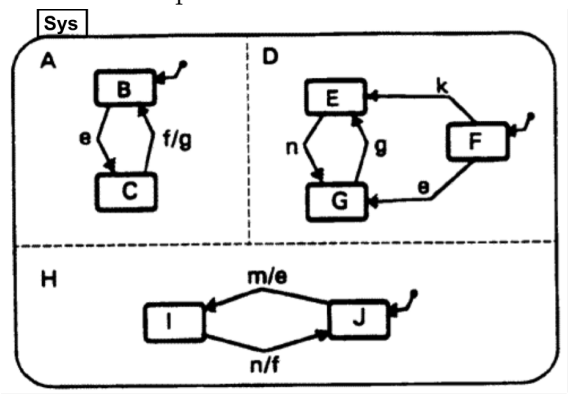
- Hierarchy of states: states can be encapsulated as super states and sub states;
- Orthogonality: Concurrent activities can be represented, i.e., there may be parallel states;
- Entry by history: Once a State becomes active, its initial state is activated. However, in some situations, it may be interesting to return to the last active state. This is possible by means of Entry by History feature; and

- Broadcasting: allows synchronization between orthogonal states of a system. A broadcasting occurs when an internal event associated with a transition is triggered, this event most likely will fire other transitions, making a chain reaction.

Statecharts also describes states, events and transitions, besides conditions, actions, expressions and variables. A state can be divided in two groups: basic states, that have no substates, and non-basic states, that have substates. A superstate (a non-basic state) can be decomposed in two types: XOR, when this state cannot be simultaneously in more than one substate, and AND, when the state is simultaneously in more than one substate. These two state types describe depth or hierarchy and concurrent representation.

Figure 2.2 shows an example represented in Statecharts. The example has a superstate root Sys, of type AND, with three substates: A, D and H. These substates of Sys will be activated at any instant at the same time. A, D and H are considered XOR substates, because when they are activated only one of their basic substates can become active. These basic states are: B and C (of substate A); E, F and G (of substate D); and, I and J (of substate H). The initial states of each substate A, D and H are B, F and J, respectively. Each component has its own logic where the reaction occurs based on events e, f, g, n, k and m. For example, the initial state of component D is F. However, the initial state of example as a whole, is B, F, J. To avoid a confusion between initial states of each component or superstate and global state, the term "configuration" is used to indicate the active basic state in each superstate.

Figure 2.2 - An example of Statecharts. Source: Harel (1987)



2.3 Test criteria and methods to FSM

As the MBT approach depends on a model, consisting of a set of states and transitions among these states, FSMs are convenient to represent the behavior of a software. As previously presented, there are other types of models that can be used to represent the behavior of system and generate test cases, but due to its simplicity and similarity with graphs, FSM will be used within the scope of this work.

FSM describes the dynamics of software based on input stimuli present in the transition arcs. But, depending on input set or length it becomes impracticable to analyze all the possibile transitions. So, strategies are needed to traverse a FSM and, at the same time, maintain a significant coverage. There are several approaches to generate test cases in a software represented in FSM. But, to do this, it is essential to understand some concepts of test criteria and methods that will be presented next.

2.3.1 Test criteria

A test criterion (or coverage criterion) is a definition of which elements of FSM should be covered to generate a test sequence (or transition sequence). Usually, a method has an objective to reach a coverage criterion, but a coverage criterion does not necessarily need to be attached to a method. This is an important step of test case generation, because depending on the criterion, different means can be used to traverse a graph. The main test criteria (AMMANN; OFFUTT, 2008; MATHUR, 2008) are:

- all-nodes: all reachable nodes of graph (or states of FSM) must be covered. Using Figure 2.1, a path that complies this criterion is: $W \rightarrow P \rightarrow B \rightarrow W$.
- all-edges: all edges of graph (or transitions of FSM) must be covered. So, naturally, all nodes will be covered. An output to this criterion by Figure 2.1 is: $[a/0] \rightarrow [b/1] \rightarrow [r/0] \rightarrow [b/1] \rightarrow [f/2] \rightarrow [c/2] \rightarrow [f/2]$.
- edge-pairs: this criterion requires all reachable paths of length 2 of a graph are covered. So, it verifies if there is an edge pair that is reachable. An output to this criterion using Figure 2.1 is: $[a/0 - b/1] \rightarrow [b/1 - r/0] \rightarrow [b/1 - f/2] \rightarrow [c/2 - r0] \rightarrow [c/2 - f/2]$.
- fault model: the test cases are generated from the model to cover all fault types identified in the domain. In FSMs, a fault model can be defined from four categories of faults: operation error, transfer error, extra-state error

and absent-state error.

2.3.2 Methods

Over the years, several algorithms for automatic generation of test cases from FSM model were proposed. In this dissertation, we consider a method an approach used to find a set of test cases, and an algorithm is an implementation of this method. In general, a method aims to achieve a test criterion by covering a model. Some known methods are presented below:

Distinguished Sequence (DS) is a method proposed by Gonenc (1970) that uses distinguished sequences to generate test cases. These sequences, when inserted as input to each state of FSM, will produce distinct outputs, called distinguished sequences. If one analyzes the output produced by FSM from a specific distinguished sequence, it one can infer which state the machine was originally. The ideal situation is that there is a smaller sequence to generate a smaller number of test cases. It is possible to use this method only on deterministic, complete, strongly related and minimal FSMs.

W-method was proposed by Chow (1978) and is one of the oldest test methods. This method generates a test sequence set that, when tested, analyzes if the model is in conformity with specification. This method generates two test sets: description (W) and transition coverage (P). This method needs that FSMs be deterministic, complete, initially connected and minimal.

Transition Tour (TT or T-method) was proposed by Naito (1981) and traverses a FSM departing at the initial state and visiting all transitions at least once, returning to its initial state, generating a unique test sequence. In this method, the test sequence (or transition tour) is generated when randomly applying the input values in the FSM, until all transitions are covered. TT can be applied only on deterministic, strongly related and fully specified FSMs.

Unique Input/Output (UIO) (SIDHU; LEUNG, 1989) is a method that, such as DS, generates a distinguished sequence among the states of FSM. But the analysis is conducted from the input and output of a certain state with respect to all other states of FSM, checking if the input and output are unique in the FSM. From the sequence generated, one can verify if the FSM was in a particular state. So, for each verification of a state the sequence can be unique. This method is applied only on deterministic, partially connected, complete and minimal FSMs.

HSI method (DOROFEEVA et al., 2010) emerged as an improvement of W-method to cover complete and partial FSM with lower cost. The difference between W-method and HSI is in the generation of description set of states that will be tested. The set of HSI is generated by separation sequence that distinguishes each state pair of FSMs.

In graph theory (BONDY; MURTY, 2008), BFS is a search algorithm in graphs used to perform a search or traverse a graph in spanning tree format. It starts at a root vertex and explores all the neighboring vertices before moving to the next level neighbors. Similar to BFS, DFS is also an algorithm for traversing a graph. DFS visits each vertex of a graph, starting at a root vertex. Then, it traverses as far as possible along each branch before backtracking, that is, the algorithm moves along a spanning tree of that graph.

CPP is a classical optimal-path algorithm that solves the problem of finding the best path in a graph visiting all edges (EDMONDS; JOHNSON, 1973). This problem is focused on graphs that are not balanced, that is, the number of input and output edges on a vertex is not the same. This algorithm has three major steps if the graph is not balanced: 1) find the minimal path from a state to any another state; 2) find the maximal matching between the unbalanced states; and 3) insert the minimal path between two unbalanced states to balance the graph. If graph is balanced, a Eulerian cycle search algorithm can be used to find the minimal path that visits all edges.

Tabel 2.1 presents which test criteria is part of each method previously presented.

Table 2.1 - Relation between test criteria and test methods

Criterion	**Method**
all-state (node)	BFS, DFS
all-transition (edge)	BFS, DFS, TT
edge-pair	BFS, Switch Cover, H-Switch Cover
fault model	W, HSI, DS, UIO

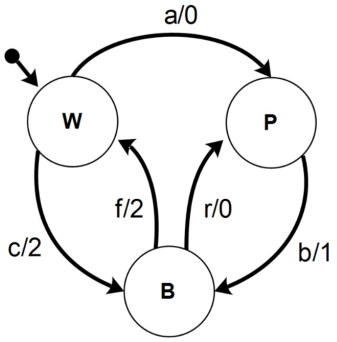
Since H-Switch Cover method is an update version of switch cover classic algorithm developed at INPE (SOUZA et al., 2008), this method will be presented with more details in next subsection.

2.3.2.1 *Switch Cover* and *H-Switch Cover*

Switch Cover is a classic method that specifies all edge-pair transitions must be executed at least once (PIMONT; RAULT, 1976). The level of complexity of this method depends on the size of FSM, because the greater the number of states, more is the time it will take to traverse the FSM.

One of the main characteristics of Switch Cover criterion is to generate a graph from an FSM, known as Dual Graph. Formally, a graph is an ordered pair $G = (V, E)$ consisting of a set V of vertices (or nodes) and a set E of edges (or arcs). Each edge in a graph connects two vertices (BONDY; MURTY, 2008). From the Dual Graph, methods for graphs can be used to generate test sequences. The classical Switch Cover algorithm, for example, after obtaining the Dual Graph, has to balance the graph. Then the balanced graph is traversed, generating test cases always starting at an initial vertex and returning to it. Using the FSM shown in Figure 2.3, the dual graph is constructed as follows:

Figure 2.3 - An example of FSM. Source: (MARIANO et al., 2016)

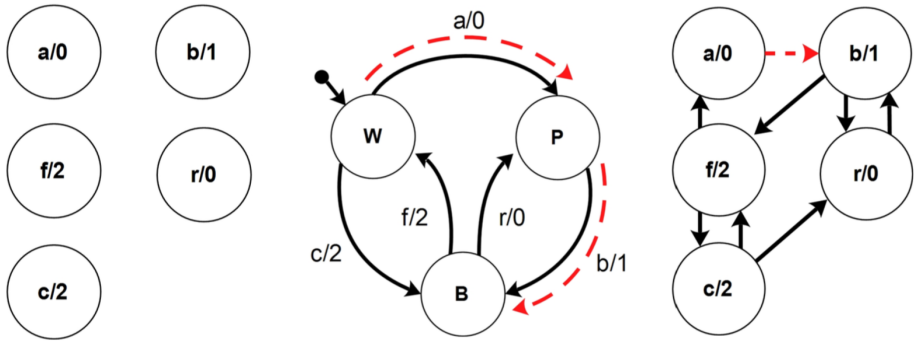


a) **Create vertices from the original FSM.** The transitions of the original FSM of Figure 2.3 must be converted into vertices (or nodes) (Figure 2.4(a)), in which W is the initial state. So, in this new graph, the new vertices "a" and "c" become the initial vertices for the method, because they were the transitions leaving state W.

b) **Add vertices.** Based on the transitions of the original FSM, the graph vertices must be created. For example, in the original FSM, there is a transition a from state W to state P (Figure 2.4(b)), and there is a transition b leaving state P. Therefore, in the new graph that is being created, an edge is added connecting the new vertices a and b (Figure

2.4(c)). The same procedure is applied to all other pairs of transitions of the original FSM. Figure 2.4(c) shows the complete graph. After this process a directed graph is obtained. A directed graph or digraph is formed by vertices connected by directed edges.

Figure 2.4 - Creation of the dual graph from the original FSM. Source: (MARIANO et al., 2016)



(a) Constructing the Dual Graph.

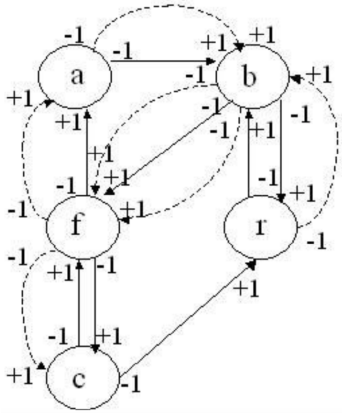
(b) Edges creation from original FSM.

(c) New edges and vertices of graph.

c) **Balacing.** The graph must be balanced analyzing the polarities of vertices. Balancing is obtained by duplicating the edges in such a way that the number of incoming edges is equal to the number of outgoing edges of the vertex (i.e. the degree of the vertex is zero). In practice, each vertex must have the same number of input and output arcs, permitting the graph to posses an Eulerian cycle where each edge is visited only once (LIPSCHUTZ; LIPSON, 1997). For the classic Switch Cover algorithm, an example of this increased number of edges can be seen in Figure 2.5, which shows the balanced Dual Graph. The dashed edges mean the addition of new edges due to balancing process.

This work will use a new version of this method, H-Switch Cover (HSC) (SOUZA et al., 2017). Its main feature is the ability to cope up with complex FSMs, improved performance and the use of Hierholzer algorithm (NEUMANN, 2004) as a heuristic to ensure that all edges are visited exactly once. The Hierholzer algorithm is based on the Euler theorem (NAITO, 1981) that generates an Eulerian cycle. The algorithm steps are:

Figure 2.5 - Balanced graph by classic Switch Cover algorithm.



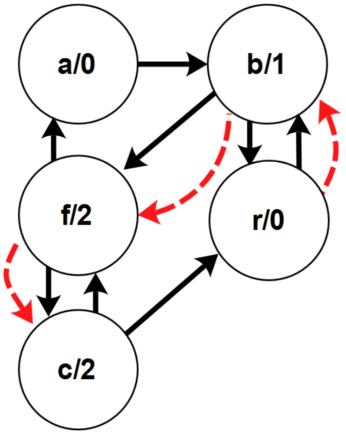
1. Start with any edge of the initial vertex and select edges not yet visited until a cycle is closed;

2. If there are still any unvisited edges, start with an edge that is a part of an already existing cycle and create a new cycle as in the first step; and

3. If there are no more edges to be visited, an Eulerian cycle must be constructed from the existing cycles, joining them from a common edge.

Figure 2.6 shows the result of balancing by HSC. Noticeably the graph resulted by the balance step with HSC is smaller than the one obtained by classic version of algorithm. An example can be analyzed in the vertex r, in which it has two edges of inputs and one of output. This vertex needs a new edge for vertex b to be balanced.

Figure 2.6 - Graph balanced by HSC algorithm. Source: (MARIANO et al., 2016)



Finally, there is now a need to generate the test sequences traversing the Dual Graph, starting at any initial vertex and returning to itself. Using the state a as being initial, the following Eulerian cycle (and test sequences) are generated: *abfcfcrbrbfa*.

2.4 Related work

The first step of this master's work was conducting a systematic mapping in order to identify which are the main approaches used in the context of MBT. Therefore, we present below some studies that have conducted other secondary studies (systematic mappings or systematic literature reviews) in the same context. Next, we present some work that conducted experiments to compare different FSM methods to generate test cases.

Barmi et al. (BARMI et al., 2011) conducted a systematic mapping with respect to the alignment of specification and testing of functional and non-functional requirements to identify gaps in the literature. The map summarizes studies and discusses them within sub categories with MBT and traceability. The main results showed that the papers that have linked the specification with test requirements focused on model-centric approaches, code-centric approaches, traceability, formal approaches and test cases, with focus on problems of MBT and traceability.

Siavashi and Truscan (SIAVASHI; TRUSCAN, 2014) conducted a systematic review in order to understand how an environment model can be used in MBT and challenges. Environment modeling is an activity that specifies a part of the real world, in which the system is integrated. From the characteristics and advantages, the authors conclude that using environment models can be helpful in robustness testing, safety testing and regression testing. However, methodological aspects of creating environment models are only discussed in a limited number of studies.

Gurbuz and Tekinerdogan (GURBUZ; TEKINERDOGAN, 2017) presented a systematic mapping to describe the advances in MBT for software safety. The results show that the safety area of MBT is broad and applicable to several domains with focus on reducing costs and increasing test coverage. Nevertheless, the solutions proposed are focused on specific domains, making the solution less generic and less reusable.

More recently, Khan et al. (KHAN et al., 2017) showed a systematic review in order to analyze the quality of empirical studies with MBT. The results identified that the community of MBT needs to provide more details in reporting guidelines, and few papers were applied in industry artifacts.

Each presented study conducted some kind of secondary study on MBT, but with different purposes. In the same way as related works, we also conducted a secondary study on MBT, however, our focus was to systematically map the methods, criteria, formal models for generating test cases and the evaluation strategies of the generated test cases.

So, now we will show other publications from the literature that propose and compare methods that generate test cases from FSM. Souza et al. (2008) presented an empirical evaluation of cost and efficiency between two test criteria of the Statechart Coverage Criteria Family (FCCS) (SOUZA, 2000) all-transitions and all-simple-paths criteria, and the Switch Cover criterion for FSMs. Mutation analysis was used to evaluate the criteria. Cost is defined according to the size of test suite and the time needed by a test suite to kill a mutant. Efficiency is related to the ability a test suite has to kill mutants (mutation score). The results showed that two FCCS criteria and Switch Cover method presented the same efficiency. But, all-simple-paths criterion presented a better cost because its test set is smaller and was faster in identifying the mutants than other test sets, meaning that the criterion can detect software faults faster than other criteria.

In Simao et al. (AO et al., 2009), the authors describe an experimental comparison among different coverage criteria for FSMs, such as state, transition, initialization fault, and transition fault coverage. To this, an approach was developed to generate suitable test cases for each one of the criteria without depending on the chosen method. First, a test suite that suits a chosen criterion was generated, and then it is minimized using a generalized greedy algorithm. The results showed that the proposed approach was faster than others found in the literature, despite a little loss in the reduction power.

Dorofeeva et al. (2010) presented an overview and an experimental evaluation of FSM methods: DS, W, Wp, HSI, UIO, and H. They analyzed the criteria on different aspects, such as test suite length and derivation time. The experiments are conducted on randomly generated specifications and on two real-world protocols. The results showed that the methods, except UIO, produce a test set almost independent of specification length, and W-method produces a larger test set than other methods. About complete test set, H method overcomes the other methods with its test set being 40% of the W-method test set.

Endo and Simao (2013) presented an experimental study that compared the complete test sets generated automatically using the methods W, HSI, SPY, H, and

P. They analyzed the number of test cases and their length, the total cost (i.e., the length) of each test suite, and the effectiveness of the methods using mutation testing for FSMs. In order to conduct the study, FSMs were randomly generated varying number of states, inputs, outputs, and transitions. The results showed that SPY produced, on average, greater test cases and HSI had smaller test cases. This indicates that the most recent methods in the literature produce a smaller test suite than traditional methods. The P method generates a shorter test set. And also, it was identified that all methods had a rate of 92% of fault detections.

Mariano et al. (2016) conducted an experiment that compared classic algorithms of generation of test cases for FSM models, such as Depth-First Search (DFS) and Breadth-First Search (BFS), with H-Switch Cover method. This experimental study considered randomly generated and real-world FSMs. The analysis considered the number of test cases, length of test suite, average of test sequences and generation time of test cases. Generally, the three algorithms showed similar results when compared with randomly and real-world FSMs. In terms of number of test cases and length, H-Switch Cover method presented better results. But, considering the average length of test cases, DFS presented the best result. Finally, in terms of time to generate test cases, DFS and BFS presented better performance.

Finally, in Souza et al. (2017), an investigation of cost and efficiency was conducted comparing the FSM criteria: H-Switch Cover, UIO and DS. In order to analyze the test cases' efficiency, mutation analysis (mutation score) was adopted. With respect to cost, the size of the test set generated (number of events) was considered. The investigation was conducted considering two embedded software products (space applications). As a result, in terms of efficiency, the three methods presented good performances about score mutation. In terms of cost, considering the quantity of events, in the first case study UIO and DS were better; but in the second case study, H-Switch Cover was better. Also, in the second case study, H-Switch Cover got a better performance in relation to the average and standard deviation of mutation scores and quantity of events of test cases generated. In relation to cost (time to detect a fault), H-Switch Cover was the fastest, because it presented smaller number of test cases.

3 APPROACHES TO GENERATE TEST CASES IN MODEL-BASED TESTING: A SYSTEMATIC MAPPING

The first step to achieve the objective of this dissertation is to investigate approaches to automatically generate test cases in MBT, such as methods, criteria, formal models for generating test cases. To achieve this, we conducted a systematic mapping to summarize the main approaches used in such a context. A systematic mapping provides a broad overview of an area of research, determining whether there is research evidence on a particular topic (KITCHENHAM, 2007; PETERSEN et al., 2015). Conducting the mapping study and its results can be referred to in MARIANO et al. (2019). This chapter is divided in the following sections: Section 3.1 shows the protocol of systematic mapping. Section 3.2 shows the how the mapping was conducted. Section 3.3 shows the results of this systematic mapping. Section 3.4 shows the discussion of results.

3.1 Research Protocol

The research method was defined based on the guidelines given in Kitchenham (2007) and in Petersen et al. (2015). The method involves three main phases: *(i)* **Planning:** refers to the pre-review activities, and establishes a protocol, defining the research questions, inclusion and exclusion criteria, sources of studies, search string, and mapping procedures; *(ii)* **Conducting:** focuses on searching and selecting the studies in order to extract and synthesize data from included ones; *(iii)* **Reporting:** is the final phase and it writes up the results and circulates them to potentially interested parties. Following, the main parts of the mapping protocol used in this work are presented.

3.1.1 Research Questions

This mapping study aims at answering the following Research Questions (RQs):

RQ1. When and where have the studies been published?

RQ2. What are the methods used in the generation of test cases?

RQ3. What are the test criteria used in the generation of test cases?

RQ4. What modeling technique is presented for generating test cases?

RQ5. What evaluation strategy is carried out in the selected study?

3.1.2 Inclusion and Exclusion Criteria

The selection criteria are organized in one inclusion criterion (IC) and seven exclusion criteria (EC). The inclusion criterion is: (IC1) The study must present an empirical approach for test case generation in MBT. The exclusion criteria are: (EC1) The study is published as a short paper or extended abstract; (EC2) The study is not written in English; (EC3) The study is an older version (less updated) of another study already considered; (EC4) The study is not a primary study, such as editorials, summaries of keynotes, workshops, and tutorials; (EC5) The study does not present an empirical approach to generate test cases; (EC6) The full paper is not available; and (EC7) Studies published before the year 2000. The EC7 was defined based on Offutt and Abdurazik (1999) study. This study is referred to as one of the first MBT approaches. Thus, in this mapping we consider only studies published after that date.

3.1.3 Keywords and Search String

The search string considered two areas, MBT and test case generation (Table 3.1), and it was applied in three metadata fields (namely, title, abstract and keywords). The search string was modified to be adapted to particularities of each source.

Table 3.1 - Keywords for the search string

Areas	**Keywords**
MBT	"MBT", "model based test", "model based testing", "model based tests", "MDT", "model driven test", "model driven testing", "model driven tests"
Test Case	"test criteria", "test criterion", "testing criteria", "testing criterion", "generation method", "generation algorithm", "test cases generation"
Search String: ("MBT" OR "model based test" OR "model based testing" OR "model based tests" OR "MDT" OR "model driven test" OR "model driven testing" OR "model driven tests") AND ("test criteria" OR "test criterion" OR "testing criteria" OR "testing criterion" OR "generation method" OR "generation algorithm" OR "test cases generation")	

3.1.4 Source

We chose to work with Scopus[1] database, since this source is considered the largest abstract and citation database of peer-reviewed literature, with more than 60 million records. In addition, Scopus attaches papers of other international publishers, including Cambridge University Press, Institute of Electrical and Electronics Engineers (IEEE), Nature Publishing Group, Springer, Wiley-Blackwell, and Elsevier.

The studies returned from sources in the searching phase were cataloged and stored appropriately. This catalog helped us in the classification and analysis procedures. With the studies remained after applying IC and EC, the snowballing process was conducted. First, referring to papers cited in these remaining studies (backward snowballing) and second, referring to those studies that cited these remaining studies (forward snowballing).

3.1.5 Assessment

Before conducting the mapping study, we tested the mapping protocol. This test was conducted in order to verify its feasibility and adequacy, based on a pre-selected set of studies considered relevant to our investigation. First, the review process was conducted by the author of this dissertation, and, only then, three specialists carried out the review validation. The specialists analyzed the studies using three different samples.

3.2 Conducting the mapping study

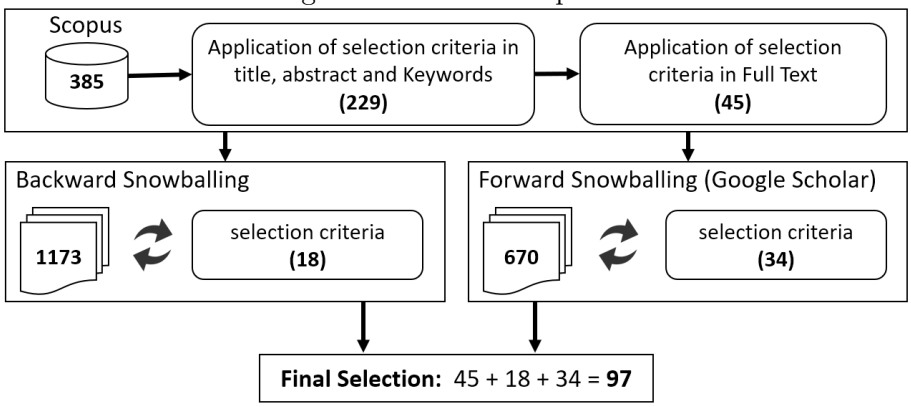
In this section the main steps that we performed in this mapping study are discussed. Then we followed a selection process as shown in Figure 3.1.

We considered the studies published until March 2018. As a result, 385 publications were returned by Scopus. The selection process was divided into two stages. In the 1^{st} stage, inclusion and exclusion criteria were applied considering the title, abstract and keywords, so 156 publications (approximately 41%) were eliminated. Although the publications mention, in the abstract, the terms contained in the search string, they did not present an approach of test case generation in MBT context. In the 2^{nd} stage, the exclusion criteria were applied considering the entire text, resulting in 45 studies (approximately 12% of total articles).

Over these 45 studies considered relevant, we performed backward and forward snow-

[1] *https://www.scopus.com/home.uri*

Figure 3.1 - Selection process



balling. The backward snowballing resulted in 1173 studies. From these 1173 studies, the selection criteria were applied considering the title, abstract and keywords. Next, the selection criteria were applied considering the full text. This stage was carried in an iterative form. Three iterations were conducted and 18 new studies were selected. The forward snowballing returned 670 studies. Four iterations were conducted, resulting in 34 new studies. As a result, we got a final set of 97 studies[2].

3.3 Data Extraction and Synthesis

After selecting the primary studies, we analyzed each one in order to answer the research questions. Next, we present the main findings from data extraction and synthesis. The complete mapping of individual studies and its categories can be accessed at: *https://goo.gl/MGfa9f*.

In data extraction and synthesis, notice that for some RQs the sum of all classifications might be greater than the total number of papers in the systematic mapping. This occurs because a given paper can answer a RQ with more than one target (models, methods) of our investigation. For instance, a given paper may answer RQ2 with more than one mentioned method and, for example, answer RQ3 with 3 mentioned criteria.

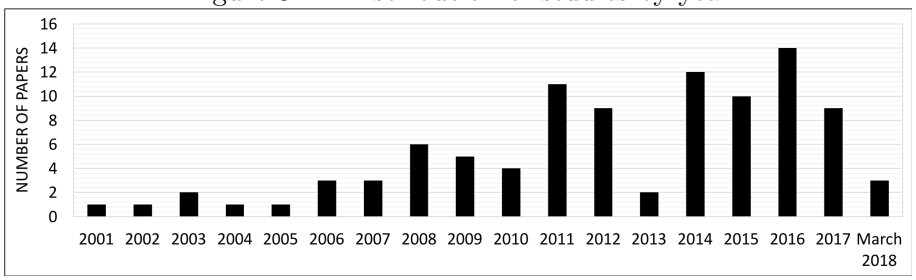
RQ1. When and where have the studies been published?

Figure 3.2 shows the distribution of the studies by years. The results suggest that research on MBT has been increasing, with a significant increase from 2014 to 2017. In fact, the period between 2011-March 2018 represents 72.2% of all publications of

[2]The references of the 97 selected studies can be accessed at: *https://goo.gl/MGfa9f.*

this mapping. That indicates the increased interest in MBT recently. In addition, in relation to the publication vehicle, Conferences are the sources with most publications representing 54.6% and Journals 41.2%. 4.1% of total number of papers we did not identify the source of publication.

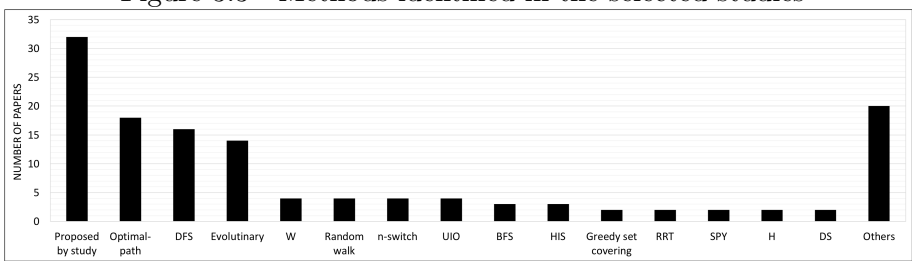
Figure 3.2 - Distribution of studies by year.



RQ2. What are the methods used to generate test cases?

This RQ led us to identify the methods that have been used in MBT. Figure 3.3 shows the main methods identified in the systematic mapping. A total of 33 types of methods were identified. Most of the papers propose a new method (24.1%), always to improve an existing method in terms of efficiency and coverage. For example, Hessel and Pettersson (HESSEL; PETTERSSON, 2007) proposed a new global algorithm that uses less memory and time to generate a test suite. In Dorofeeva and Koufareva (DOROFEEVA; KOUFAREVA, 2002), there was a proposal of a new modification of W-method that has some improvements to the classic algorithm. Despite based on a classic method, it is considered as a new method proposed by the author. The same occurs in Souza et al. (SOUZA et al., 2017) by proposing a new method called H-Switch Cover based on a classic algorithm n-switch set cover. So, both methods were classified in this research question (H-Switch Cover and n-switch set cover).

Figure 3.3 - Methods identified in the selected studies



Optimal-path methods have also been used in the generation of test cases (13.5%), as shown in Figure 3.3. These methods are very useful to generate test cases with maximum coverage of a system model, while optimizing the number of test cases. Different types of methods were selected in this group. For example, Belli and Hollmann (BELLI; HOLLMANN, 2008) employed the k-transition coverage to generate a test case for each of length k. Then, it is minimized using the Floyd-Warshall algorithm to generate the shortest path. In Julliand et al. (JULLIAND et al., 2018), a method was proposed to generate test cases from a predicate abstraction of a system specified as an event-based system. This method uses the Chinese Postman Problem to cover the transitions of the given model.

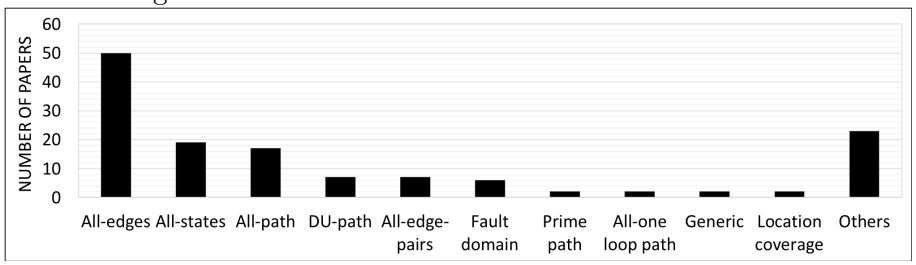
Finally, we also identified that the well-known classic methods continue to be used in MBT. Depth-First Search (DFS) method, for instance, was the most mentioned (12%). This method was used to generate a complete and fast coverage of the model, since it is well-known and very simple to implement. Many studies used the DFS concurrently with another criterion or method, such as Takagi et al. (TAKAGI et al., 2010) that uses DFS to satisfy the n-switch coverage criterion or applied in a tool to automate all the MBT process like (CARTAXO et al., 2008). We also found other studies that do not focus on generation of test cases, but recommend or cite the method to be used for test case generation. For example, Rocha and Martins (ROCHA; MARTINS, 2008) focus on the generation of stubs to execute the test cases in an environment, and recommend that DFS, for example, can be used to generate test cases.

RQ3. What are the criteria used in the generation of test cases?

RQ3 focuses on verifying the criteria used in the studies identified. Figure 3.4 shows the main criteria found in the selected studies. We identified 29 types of criteria, and the three most mentioned criteria are related to graph coverage: *All-edges* of a graph (or equivalent representation, such as transitions of an FSM) with 36.5%, followed by *all-states* (13.9%) and *all-paths* (12.4%).

Many methods identified in RQ2 use a single criterion to generate test cases. For instance, Belli et al. (BELLI et al., 2009) propose a method that uses an optimal-path finding algorithm to generate test cases covering *all transitions*. Devroey et al. (DEVROEY et al., 2014) proposed an approach to generate a suite of abstract test cases that satisfies the *all-states* criterion. In Vu et al. (VU et al., 2015), the authors proposed a method to automate test data generation and generated all possible test scenarios. That study does not directly mention the criterion used by algorithm, but

Figure 3.4 - Criteria identified in the selected studies



it is implicit that *all-paths* is used.

Other studies adopt a method to cover two or more criteria. For instance, Hessel and Pettersson (HESSEL; PETTERSSON, 2007) apply both *all-transitions* and *all-states* to generate test cases using the global algorithm. Belli et al. (BELLI et al., 2014) introduce an event-oriented approach to MBT of Web Service Compositions. In order to do this, their algorithm uses three different criteria to generate test cases: 2-length sequences (*all edges*), 3-length sequences (*all edge-pairs*), and 4-length sequences.

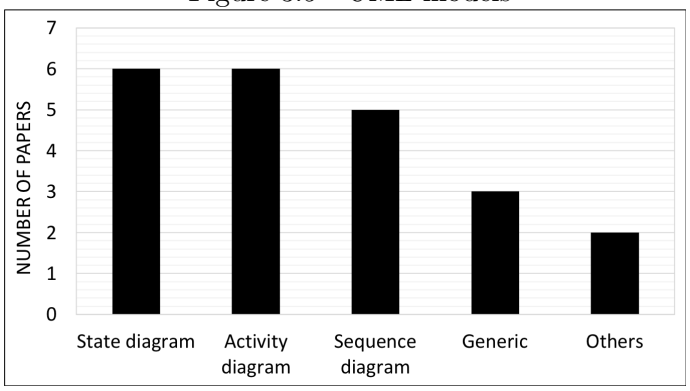
RQ4. What formal model is presented for generating test cases?

A large number of formal models for generating test cases were observed. However, a large number of Unified Modeling Language (UML) models used in the papers drew our attention. So, we analyzed this RQ in two steps. First, we analyzed only UML models, in order to identify which UML diagram was the most used. Then, all other formal modeling techniques were analyzed.

Figure 3.5 shows the distribution of UML models. State diagram is the most used. Some studies convert a UML diagram to another formal model to generate test cases. For example, in Anbunathan and Basu (ANBUNATHAN; BASU, 2013), a state diagram is converted into Extended FSM (EFSM), and then to Control Flow Graph (CFG), to generate test cases applying some criteria like transition coverage, path coverage, or dataflow coverage. In other studies, the authors use state diagrams to generate test cases directly. For instance, Li and Offutt (2017) analyze the ability of test oracles in revealing failures. To do so, test suites are generated from UML state diagrams applying ten elicited test oracle strategies.

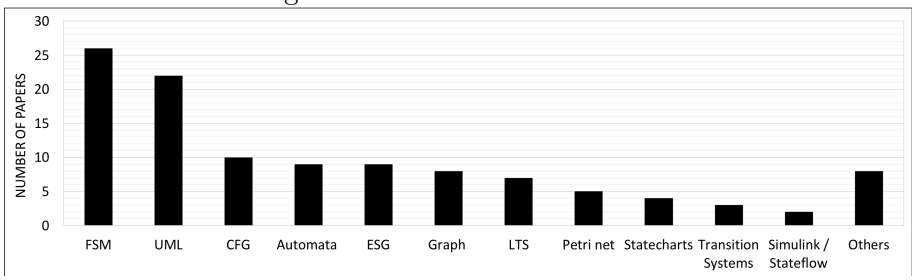
Figure 3.6 presents the main identified formal models. In general, similar to UML diagrams, formal models based on states are very much accepted in the literature,

Figure 3.5 - UML models



with FSM being the most used to represent a system. In Majeed and Ryu (MAJEED; RYU, 2016), for instance, a novel testing approach is proposed to combine OS-level replay mechanism with MBT and generate test cases applying the n-switch set cover in an FSM. Also, other studies proposed an extended version of FSM for a specific problem; for example, Yao et al. (YAO et al., 2014) proposed a new model (Pipelined EFSM) for Software-Defined Networking data plane. Despite state diagram representation in general (e.g., FSM, Statecharts, UML state diagram) can be considered as a graph representation, other formal models based on graphs (like event-driven models) are also popular in literature.

Figure 3.6 - Formal models used



RQ5. What evaluation strategy is carried out in the selected study?

Finally, RQ5 verifies what are the most used evaluation strategies in the literature. This evaluation strategy is related to the test cases generated and also the model used. We separated the evaluations into two categories (test case and model), because we observed that these were the most used in the selected studies.

Evaluations related to test cases were more commonly used in the selected studies. Some evaluations are: number of test cases (18.2%), fault detection (11.5%) and mutants analysis (7.4%). In Belli et al. Belli et al. (2013), for instance, a mutation-based approach was proposed in order to test Go-Back functions modeled by pushdown automata. In Ali et al. (2007), a technique is proposed that combines UML diagrams and Statecharts to generate an intermediate model (SCOTEM) to generate test paths, also defining coverage criteria to be used in the model. Validation of this technique analyzes fault detection capability, generating mutant programs. Belli et al. (2012) proposes a framework that uses PDA related to context-free language to generate a small test set, proposing new mutation operators and a novel criterion to generate negative test cases. Evaluation of the framework is based on the analysis of the number of negative and positive test cases generated, as well as their average length, number of events executed, number of faults and the performance of fault detection.

With respect to model evaluations, they were based mainly on number of edges (7.4%), number of states (6.8%) and fault coverage (4.1%) in a model. For example, in Satpathy et al. (SATPATHY et al., 2012), they propose a method to cover parallel and hierarchical stateflow model, without flattening the components of the model, and generate test cases. The approach is evaluated comparing the tool proposed with another commercial tool in coverage of states and transitions. Cartaxo et al. (CARTAXO et al., 2011) propose using a similarity function to generate less redundant test suite, for the purpose of effectiveness. The approach is compared with a random selection and uses the transition coverage to evaluate it.

Finally, we identified evaluations for both test cases and models, such as time (24.3%) and memory used (2.7%). The time is the most mentioned evaluation in this systematic mapping. Some studies use the time as a performance metric of computation time. For example, Dang and Nahhal (DANG; NAHHAL, 2009) proposed a framework to use conformance testing in continuous and hybrid systems and a method based on Rapidly-exploring Random Tree, called RRT, to generate test cases. The time is used to analyze the efficiency of the method in linear systems, comparing both methods. Another study that also uses RRT algorithm is presented by Proch and Prabhat (PROCH; MISHRA, 2014) in which they proposed an approach to generate directed test cases in hybrid systems. In order to validate the approach, time is used to compare RRT proposed with the state-of-art RRT technique. In Wang et al. (WANG et al., 2016) time is used in an empirical evaluation of a tool, Tansuo, that uses a proposed approach to generate navigation graphs for dynamic web applica-

tions using combinatorial testing. The paper also analyzes how much time is needed to generate a navigation graph and to generate test cases using the t-way coverage criterion.

3.4 Discussion

In this Section, directions to several approaches associated to MBT have been identified. Some relevant points are discussed as follows.

MBT is a research topic that shows itself in a constant growth, as can be seen in RQ1. This can also be confirmed by the number of secondary studies that have been conducted. We have identified ten secondary studies with different purposes and each with a considerable number of primary studies returned, which we believe are in the strong interest in this research area.

Another result that makes us infer the constant growth of interest in this area of research is the number of contributions on new methods proposed for the generation of test cases. As can be observed in RQ2, there is a great effort spent in developing new test methods; out of the 97 selected studies, 32 proposed a new method for generating test cases from formal models. We also identified 29 test criteria in RQ3. The formal models used were also quite varied with more than 24. And, with respect to RQ5 we identified 145 types of evaluations conducted in the selected studies considering the test cases generated and models used.

One of the main features of MBT is the automated generation of test cases usually based on a formal representation of the software specification. We noted by the systematic mapping that the industry has an inclination to employing UML models (22 UML models), due to the approximation with user and focus on systems that need the user control. On the other hand, in the case of academy, models like FSM (26) and Statecharts (4) are more used. In particular, FSMs have been heavily adopted to generate test cases for different kinds of applications, such as Web applications and embedded systems, specially reactive systems. FSM is commonly used for testing due to its rigor and simplicity. Great effort has been spent on the development of methods and criteria for FSM that select or generate effective test sets capable of revealing all faults from a given domain, as can be seen in RQ4. In general, these models (FSM and Statecharts) apply graph methods to generate test cases, either to obtain the best path or high coverage.

With respect to the evaluation strategy carried out in the selected studies, we have

identified that the researchers usually divide the evaluations into two main categories: test cases and formal models. Evaluations related to test cases are more commonly used (39 different type of evaluations). Different analyses in the test cases are conducted to understand the behavior of the test methods. A tester can identify the trade-offs between test case characteristics, such as test suite length, and fault detection ratio, as shown in RQ5, and consequently choose the best method to fit for a given project.

It is important to emphasize that our focus was on studies that addressed, mainly, methods for test case generation. We identified 97 studies addressing methods, formal models and evaluation strategies on MBT. The major contribution of this Section was to summarize and highlight the main aspects associated to MBT, but focused on the methods, formal models and evaluation strategies most used in the existing literature. These results could be of interest to several researchers involved with MBT. We believe that our summarization can help to direct researchers in their future research providing a pointer to appropriately position new activities in this research topic.

4 GRAPH-BASED METHODS EMPLOYED TO GENERATE TEST CASES FROM FSMS: AN EXPERIMENTAL STUDY

The second step of this dissertation was to compare graph-based methods employed to generate test cases from FSM models considering the mapping results presented before. It is divided in the following sections: Section 4.1 shows which methods and criteria we used for evaluation. Section 4.2 defines how the experimental study was conducted in this work. Section 4.3 presents the results of this study. Section 4.4 shows the discussion of results.

4.1 Criteria and methods analyzed

In this study, considering the results found by the systematic mapping, we decided to investigate two criteria and four methods, divided by two main groups of methods, employed to generate test cases from FSM models: graph-search methods (Breadth-First Search, Depth-First Search) and Eulerian methods (H-Switch Cover and Chinese Postman Problem). BFS, DFS and CPP were chosen due to the results obtained from Systematic Mapping. On the other hand, HSC was chosen as it is an Eulerian method and moreover it is already being used in the WEB-PerformCharts (SOUZA et al., 2008).

In graph theory (BONDY; MURTY, 2008), BFS and DFS are used to perform search for a particular vertex or traverse a graph. BFS focuses on traversing a graph, represented in the format of a spanning tree, exploring all the neighboring vertices of a certain level and moving to the neighbors of the next level. On the other hand, DFS visits each vertex, of a given level, starting at the root and traverses each branch before backtracking and moving on to the next branch.

As presented in Section 2.3.2.1, HSC (SOUZA et al., 2017) is a method, based on the classical Switch Cover (PIMONT; RAULT, 1976), that specifies the ATP criterion must be executed at least once. One of the main characteristics of Switch Cover is to generate a transition-pair balanced graph from a FSM, known as Dual Graph. The main feature of HSC is the ability to cope with complex FSMs and the use of Hierholzer algorithm (NEUMANN, 2004) as a heuristic to ensure that all edges are visited exactly once. The Hierholzer algorithm is based on Euler theorem (NAITO, 1981) that generates an Eulerian cycle.

CPP is a classical optimal-path method that solves the problem of finding the best path in a graph visiting all edges (EDMONDS; JOHNSON, 1973). This problem is

focused on graphs that are not balanced, that is, the number of input and output edges on a vertex is not the same. This method has three major steps if the graph is not balanced: 1) find the minimal path from a state to any another state, using the Floyd-Warshall algorithm; 2) find the maximal matching between the unbalanced states, with the Hungarian Method; and 3) insert the minimal path between two unbalanced states to balance the graph. In this last step, BFS algorithm was used to search the path between two unbalanced nodes. If the graph is balanced, an Eulerian cycle search algorithm can be used to find the minimal path that visits all edges.

All the four methods were implemented using Java language (version 1.8). While BFS, DFS and H-Switch Cover were implemented using tree data structures, separating the states and transitions of a FSM in different classes, CPP used matrix, a simple data structure, mapping each row and column of the matrix as a state of FSM. All the code can be accessed at GitHub[1].

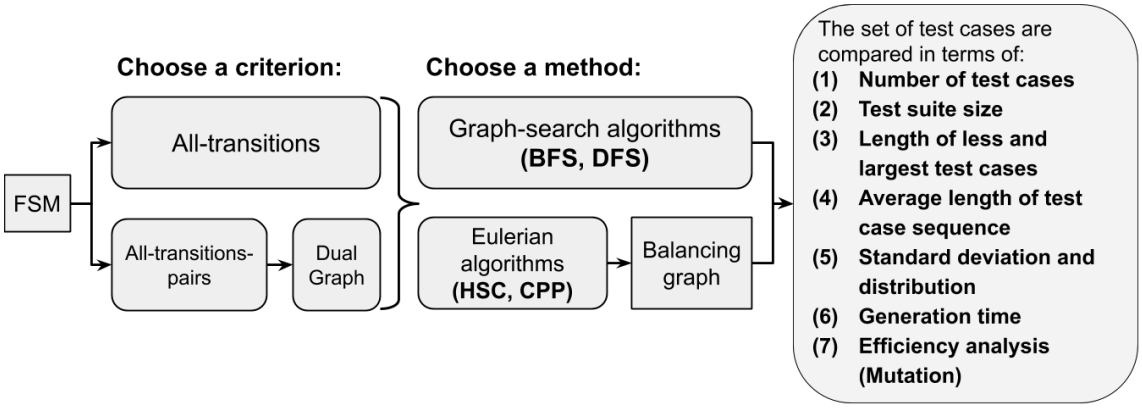
4.2 Experimental Study

Figure 4.1 shows the study overview of this work. From a FSM, a criterion can be chosen to define how the FSM will be covered. In all-transitions there is no need for another step, but in all-transition-pairs it is necessary to cover the original FSM and generate a Dual Graph. After this step, it is necessary yo choose a method to generate test cases: graph-search methods or Eulerian methods. In graph-search methods, there is for another step. So, with all-transitions criterion, the graph-search method will generate test case from the original FSM, or with all-transition-pairs, from the Dual Graph. On the other hand, for Eulerian methods it is necessary to balance the graph. In all-transitions, the Eulerian methods will balance the original FSM, but in all-transition-pairs they will balance the Dual Graph. After this, the method selected will generate test cases and the results will be compared with certain evaluations.

The methods were evaluated with 875 randomly generated FSMs and twenty-one from the real-world systems. We adopted reduced and deterministic FSMs generated in (ENDO; SIMAO, 2013) for the random case. We selected machines with four inputs, four outputs and the number of states ranging from four to twenty. For each FSM configuration, approximately 100 different FSMs were adopted, and the average metrics were calculated to reduce the influence of particular FSM characteristics on the results.

[1] https://github.com/MatheusMMariano/H-Switch_Cover

Figure 4.1 - Study overview



With respect to the real-world FSMs, two software products embedded into computers of space projects under development at National Institute for Space Research (INPE) were used in this experiment. The first software is Alpha, Proton and Electron monitoring eXperiment in the magnetosphere (APEX). The second software is Software for the Payload Data Handling Computer (SWPDC), which was developed in the context of the Quality of Space Application Embedded Software) (QSEE) research project. Details for the two software products used and their FSMs can be found in (SOUZA et al., 2017).

Evaluation of methods is not an easy task, because it is necessary to analyze parameters before defining if a method is more efficient than other. Each method has its peculiarities, which can be of advantage in some cases and a problem in others. So, evaluating a method does not only check if a method is more efficient, but also checks in which cases a method can be more or less advantageous. In our study, each method generated test cases by AT and ATP criteria. The methods were applied in graphs generated from FSMs (AT) and in a Dual Graph (ATP). The choice of these criteria was guided by the systematic mapping, which showed the most used methods and criteria in test case generation from FSMs. With respect to test algorithm evaluation, we analyzed:

- **Number of test cases.** We consider the number of test cases as the number of resets in a test suite, that is, number of test sequences.

- **Test suite size.** The number of input events as the test suite size was considered.

- **Length of smallest and largest test cases.** Length of the smallest and largest test cases for each FSM configuration. An average is calculated since each FSM configuration have approximately 100 FSMs.

- **Average length of sequences.** This considers the sum of number of input values in each test cases divided by the total number of test sequences.

- **Standard deviation and Distribution.** These metrics are calculated in relation to average length of test sequences.

- **Generation time.** This refers to the time (in milliseconds) needed to generate a test suite. Each configuration was executed 100 times and the average generation time was measured.

- **Efficiency.** The efficiency of a test case is related with the capacity in finding a fault in the code. Fault-Model Testing, for example, uses the information of most common types of faults in the software development to derive the test requirements. The mutant analysis criterion, shown in Section 2.1, inserts faults in the original software and generates mutants as a means to find the difference between the behavior of the original software with mutant, and a certain test case generated by original model. In this study, the test cases generated for each method and criterion was modified from a

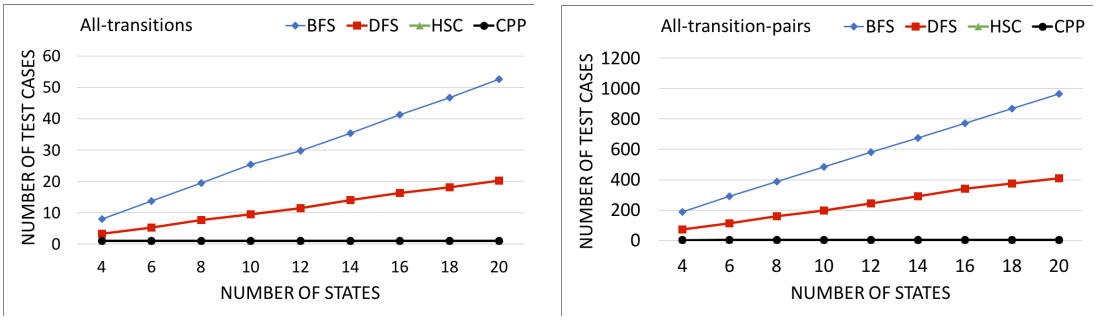
This experiment was run on a iMac computer (21.5-inch, Mid 2011), with an Inter Core i5 processor running at 2.7 GHz, memory of 8 GB (1333 MHz DDR3) and running macOS High Sierra on Version 10.13.6.

4.3 Results of test cases

4.3.1 Results for randomly generated FSMs

Number of test cases. Figure 4.2(a) shows how many test cases were generated by the methods for each FSM configuration with AT and Figure 4.2(b) by ATP criterion. BFS generated more test cases due to the construction of a spanning tree and number of leaf nodes increasing as graph size increases. The result of HSC and CPP is always the smallest number and the same in all FSM configurations since both produce a single Eulerian cycle for each initial state.

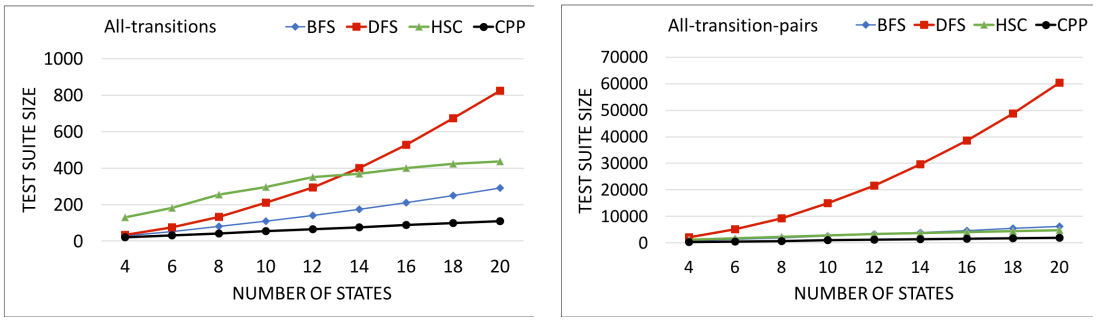
Figure 4.2 - Number of test cases results.



(a) (b)

Test suite size. Figures 4.3(a) and 4.3(b) show how the test suite size varies considering the number of states. CPP ended up turning out the best method in AT and ATP criteria, generating a minimum number of inputs with and without resets. DFS, on the other hand, generates more input values due to the nature of the depth of the graph right at the first time. Despite HSC and CPP balancing the graph and generating an Eulerian sequence, this result was expected since CPP generates a minimum balanced graph and smaller than the one generated by HSC.

Figure 4.3 - Test suite size results.

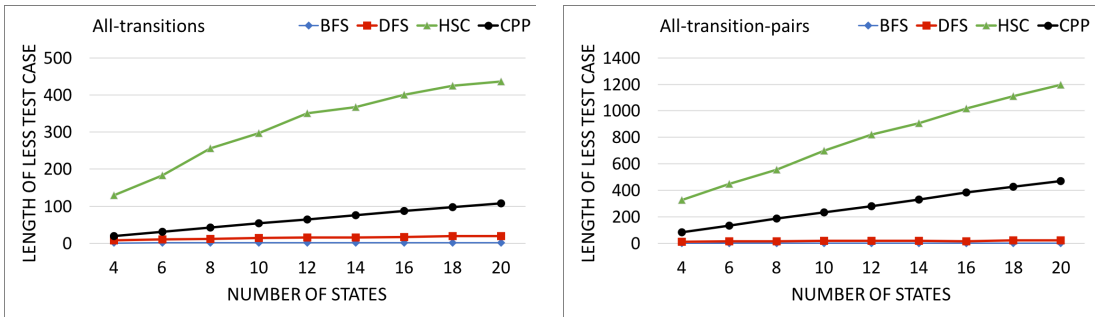


(a) (b)

Length of smallest and largest test cases. Figures 4.4(a) and 4.4(b) show the average length of shortest test case of each FSM configuration, and Figures 4.5(a), 4.5(b) show the average length of largest test case. In all cases, BFS produced test cases with the shortest length. With respect to methods that generate Eulerian cycle, CPP was better than HSC, since its balancing process generates a minimum graph. Also, the shortest and largest test case results are the same as HSC and CPP

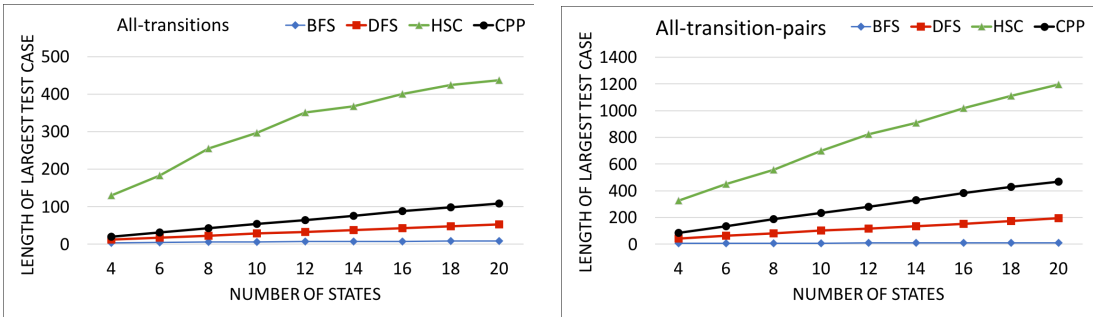
since Eulerian cycle generates a minimum sequence that visits all transitions.

Figure 4.4 - Length of smallest test cases results.



(a) (b)

Figure 4.5 - Length of largest test cases results.

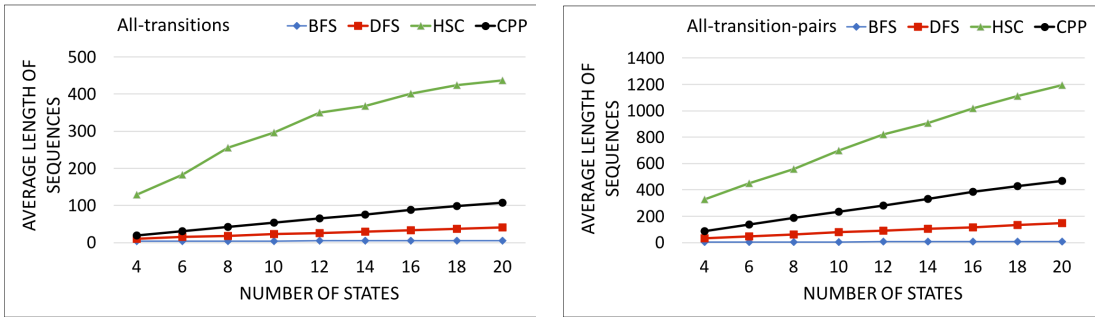


(a) (b)

Average length of test cases. Figures 4.6(a) and 4.6(b) show the average length of test cases generated by each FSM configuration. First, for each test suite an average was calculated and then the same process was repeated for each FSM configuration. BFS was the method that generated the smallest set of test sequences. Analyzing Eulerian methods, CPP generated a smaller sequence set than HSC, once this method generates a minimum balanced graph and, then, an Eulerian path. Also, all sequences of CPP are smaller than or equal to HSC.

Standard deviation. Figures 4.7(a) and 4.7(b) show the standard deviation for each method by FSM configuration. In all scenarios, DFS has high standard deviation and high variance of test cases. CPP has 0 standard deviation and variance in all the cases, indicating that the test cases generated had the same length without

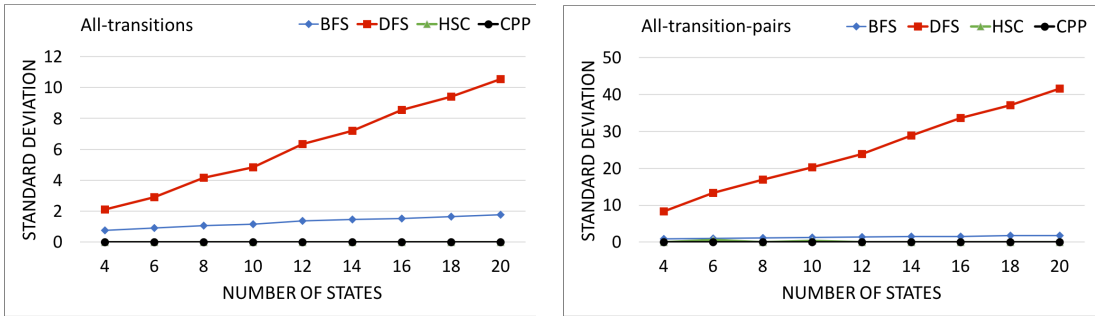
Figure 4.6 - Average length of test cases results.



(a) (b)

any change among them.

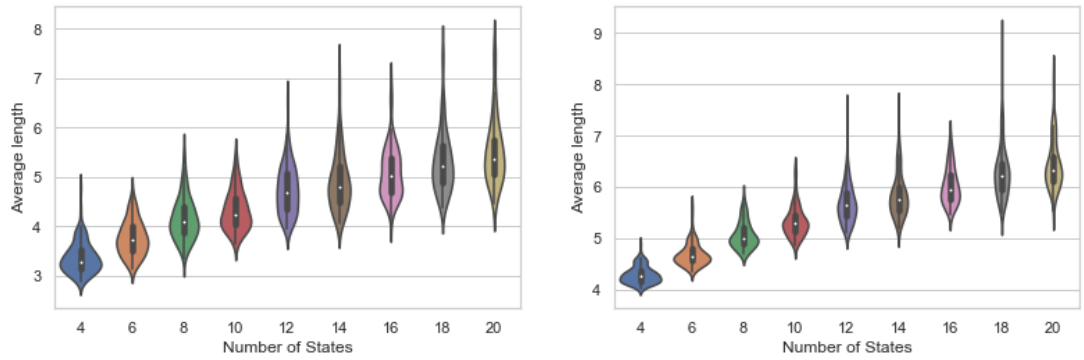
Figure 4.7 - Standard deviation of average of test cases results.



(a) (b)

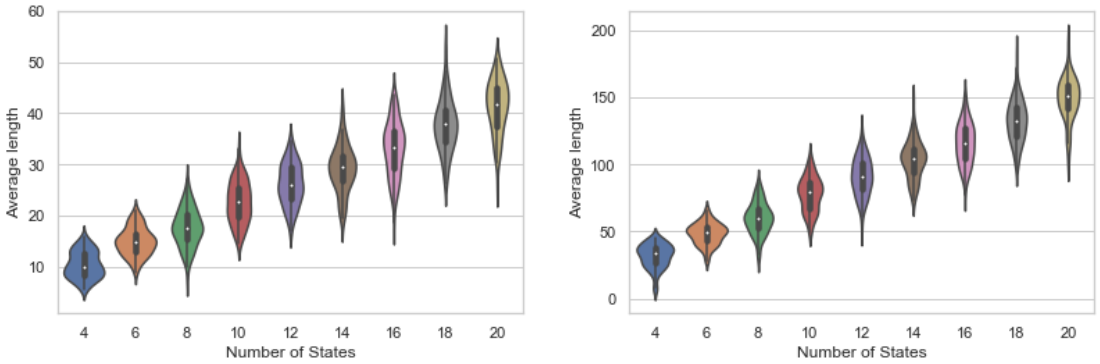
Distribution of average length of test sequences. Figures 4.8, 4.9, 4.10 and 4.11 show the distribution of average length of test sequences by number of states. This result is presented as a violin plot, a style that shows the dispersion of data set between the maximum and minimum points. The larger is the curve, higher is the number of sequences that have a similar length. White point represents an average value for all graphs. In general, CPP has a similar distribution in both AT and ATP, with length between minimum and maximum less than HSC. These results show how the balancing step of Eulerian algorithm can affect the performance of method. BFS shows a tendency in generating test sequences with length too long or too small as the number of states grow. In DFS the average seems to be varying, but the scale of DFS is larger than BFS. So, results show the behavior of standard variation in DFS.

Figure 4.8 - Dispersion of average length of test cases by BFS.



(a) AT criterion. (b) ATP criterion.

Figure 4.9 - Dispersion of average length of test cases by DFS.

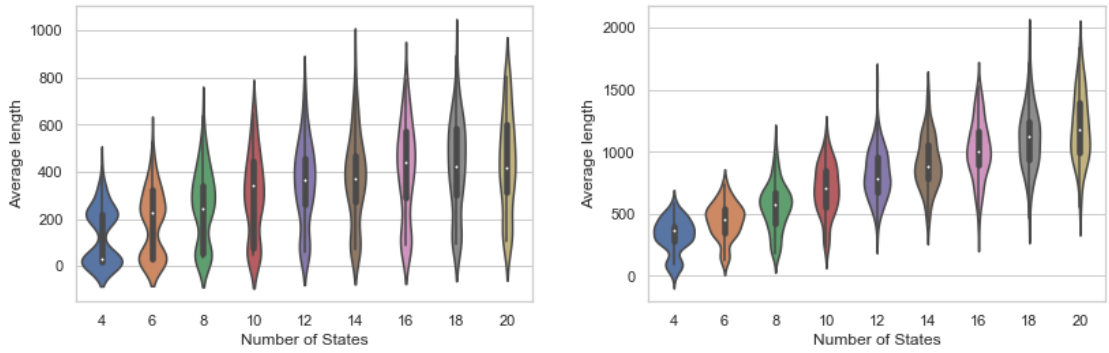


(a) AT criterion. (b) ATP criterion.

Generation time. Figures 4.12(a) and 4.12(b) show the generation time of test cases of each method. For AT criterion, BFS, DFS and CPP methods had similar time, with CPP being the method with smallest oscillation as the number of states increases, with HSC having a higher increase in time. These results are not repeated with ATP criterion, with BFS showing a higher growth than HSC. In both cases, CPP was the least oscillating algorithm in relation to generation time and, on average, it was the fastest to generate all test cases.

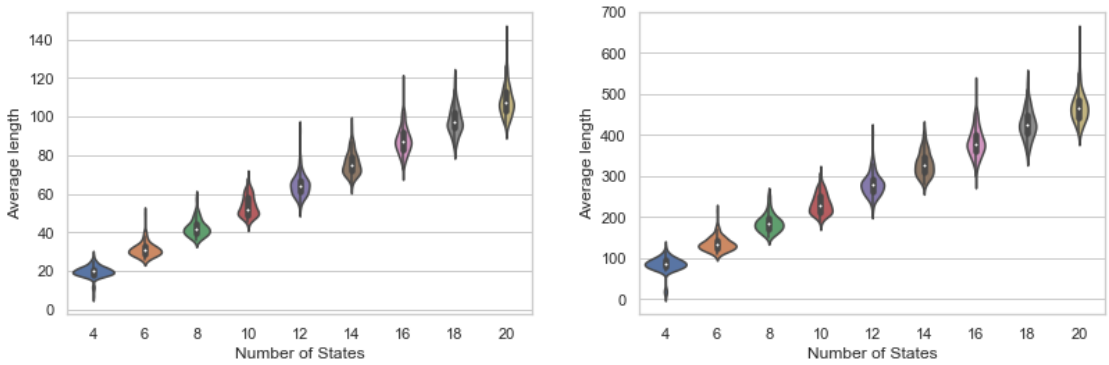
At a glance, it may seem strange to see CPP, naturally a costlier algorithm, to outperform BFS and DFS. But it is valid to mention that data structures of the three algorithms are different. CPP uses a matrix, a simple data structure, while BFS and DFS use graph structure. So, this aspect may be an impact factor for

Figure 4.10 - Dispersion of average length of test cases by HSC.



(a) AT criterion. (b) ATP criterion.

Figure 4.11 - Dispersion of average length of test cases by CPP.

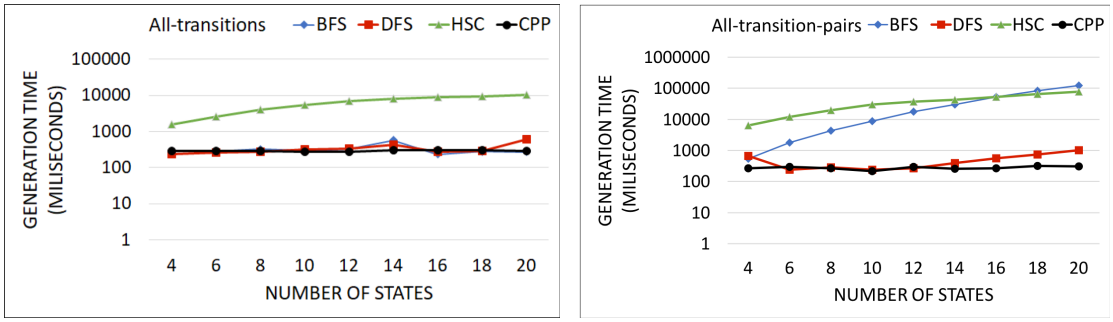


(a) AT criterion. (b) ATP criterion.

the behavior of the algorithm. The implementation of CPP was not in graph due to the complex formula of Hungarian Method to analyze the maximal-matching of unbalanced nodes, which would be more complex to implement.

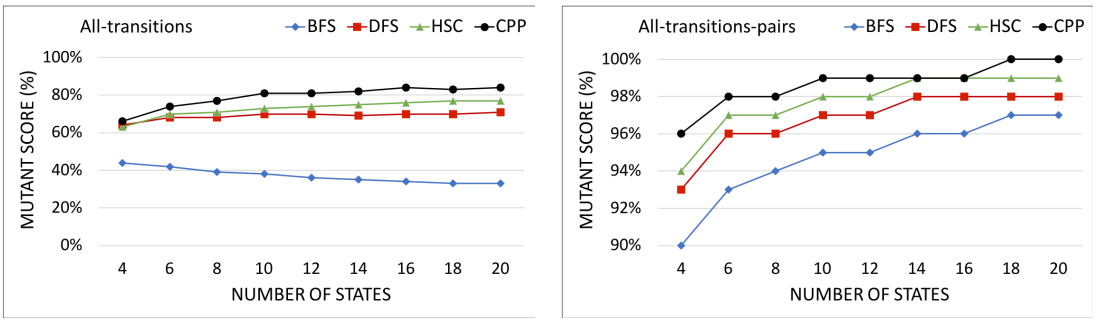
Efficiency. Figures 4.13(a) and 4.13(b) show the efficiency of test cases applying some mutant criteria. These results show the percentage of mutation score obtained by each method's test set. CPP performed better for AT and ATP criteria, but HSC had similar results as CPP, despite HSC generating a large test case compared with CPP. BFS in AT criterion showed not to be efficient with a decreased mutation score as the number of states increases.

Figure 4.12 - Generation time results.



(a) (b)

Figure 4.13 - Efficiency results.



(a) (b)

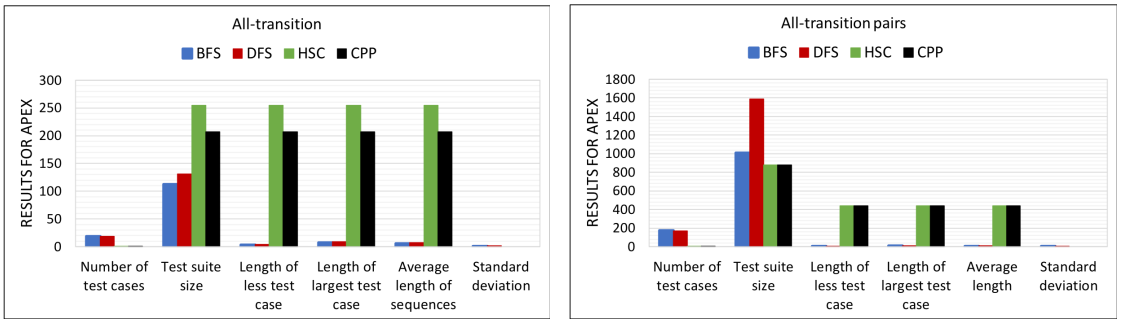
4.3.2 Results for real FSMs

Figure 4.14(a) shows the results of APEX with the four methods with AT criterion. BFS and DFS achieved the best results in four case studies, but regarding number of test cases, HSC and CPP naturally had the best results. Eulerian methods also had 0 standard deviation and variance, similarly as in the case of random FSMs. This scenario is repeated with ATP criterion, as shown in Figure 4.14(b), except with test suite size that HSC and CPP were better than BFS and DFS.

Figure 4.15 shows the results of efficiency analyses of each method applied in the APEX product. In general, CPP and HSC were better than DFS and BFS, with a small difference in ATP criterion. In ATP, HSC and CPP obtained the same result. Curiously, BFS performed better than DFS in ATP criterion, being the only case that this happens.

Figure 4.16 shows the average results of second case study referring to the 20 scenar-

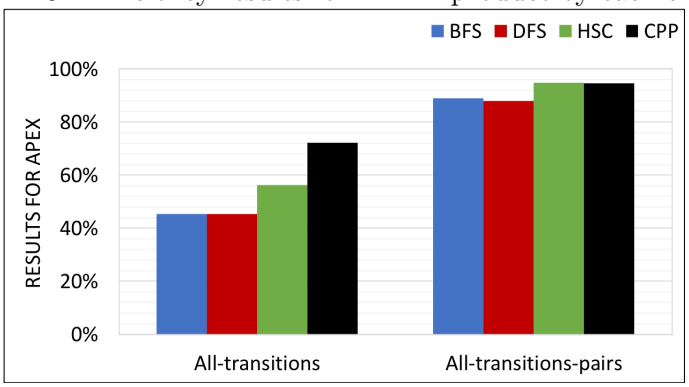
Figure 4.14 - Results for APEX product by each criterion.



(a) AT criterion.

(b) ATP criterion.

Figure 4.15 - Efficiency results for APEX product by each criterion.

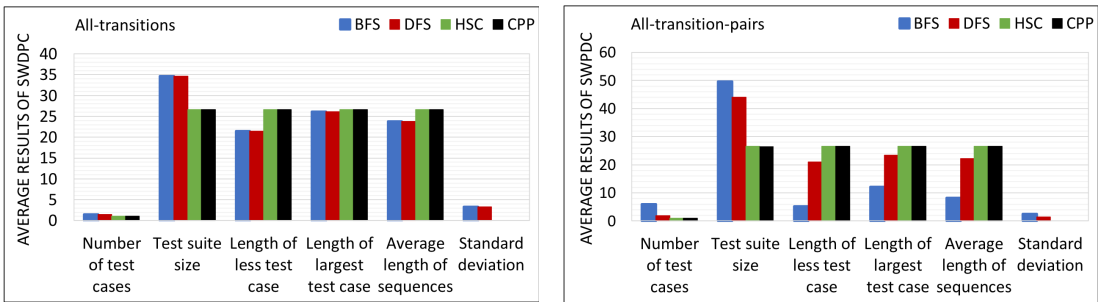


ios of SWPDC. We generated an average of 20 FSMs of SWPDC for each method. HSC and CPP achieved the best results in 4 out of 7 analyses. Between the graph-search methods, BFS presented the worst results. Eulerian methods, again, achieved the smaller number of test cases and test suite size. The length of smallest, largest test cases and the average length of test cases from Eulerian methods is the same because they generate a unique sequence with all transitions of FSM. Again, these 2 methods present 0 standard deviation and variance values. These scenarios are repeated in AT criterion. In ATP criterion, the results of BFS in test suite size analysis is much higher than others. In general, Eulerian methods results did not present a high variation among applied criteria.

Figure 4.17 shows the efficiency analyses of 20 scenarios of SWPDC. This result presents the distribution of mutation score resulting from applying each method and criterion. It should be noted that the outliers of violin plot are only a representation and do not represent the maximum and minimum values of this result (the maximum

and minimum are represented by vertical straight line in the center, with the white point being the average). CPP and HSC obtained similar results in AT and ATP criterion, with a higher range of distribution in comparison with BFS and DFS. In fact, one can see that the distribution of results is close to the maximum and minimum values. Due to this, the BFS and DFS methods were better than HSC and CPP. This happens because SWPDC has two particular cases of models: 1) when its states have only one transition to another state, without any branch; and 2) when only one state has a branch. In this case, DFS and BFS can outperform HSC and CPP. In ATP, this situation is more evident with the DFS result in comparison to others.

Figure 4.16 - Average results for 20 scenarios of SWPDC product.



(a) AT criterion.

(b) ATP criterion.

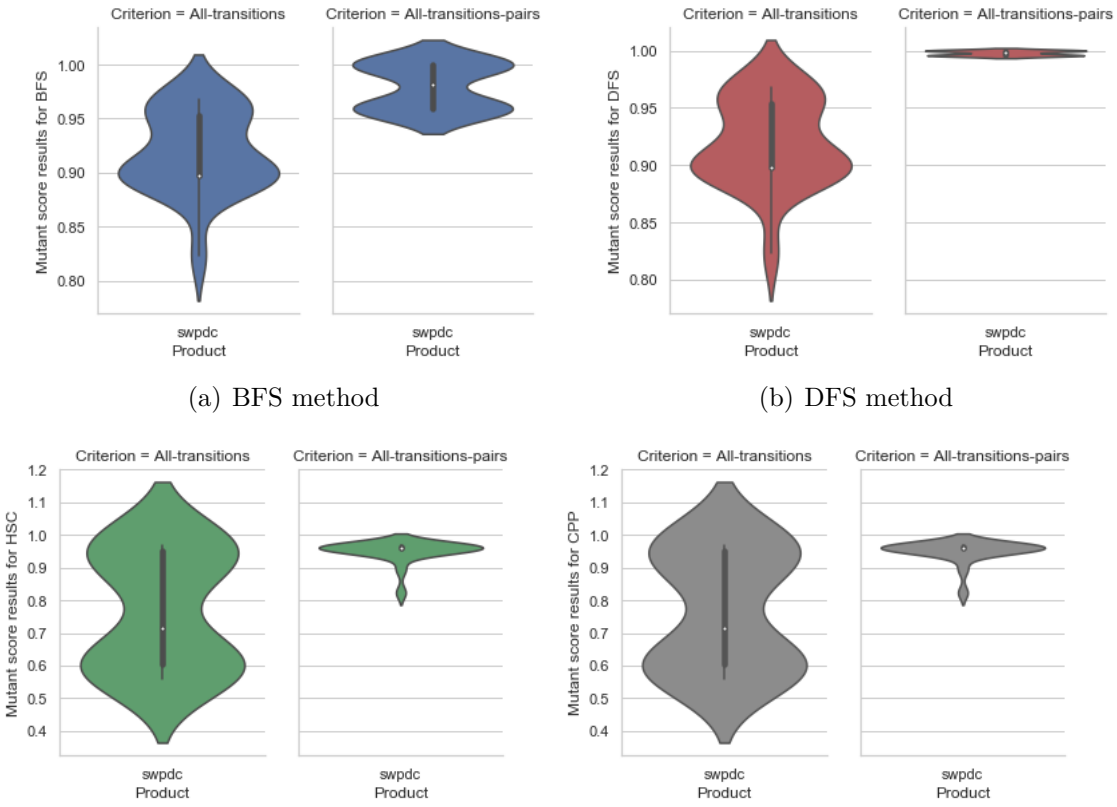
Figures 4.18(a) and 4.18(b) show the generation time in milliseconds. DFS was the fastest method with the AT criterion, and CPP was the best with AT pairs criterion. In relation to Eulerian methods, CPP was better than HSC in all cases, needing less time to generate test cases.

In relation to average distribution, Eulerian methods behave similarly with all criteria. BFS and DFS have the same distribution in AT criterion, but in ATP the average is different; for BFS the average is close to the maximum limit.

4.4 Discussion

This Section presented a comparative study of test case generation from four graph-based methods, considered the most investigated in the literature, according to the systematic mapping. Two methods are classic-search graphs: BFS and DFS, and other two are Eulerian methods: HSC, CPP. The criteria used are AT and ATP. The methods used the original graph to apply AT criterion, whereas a Dual Graph

Figure 4.17 - Average mutantion score results for 20 scenarios of SWPDC product.



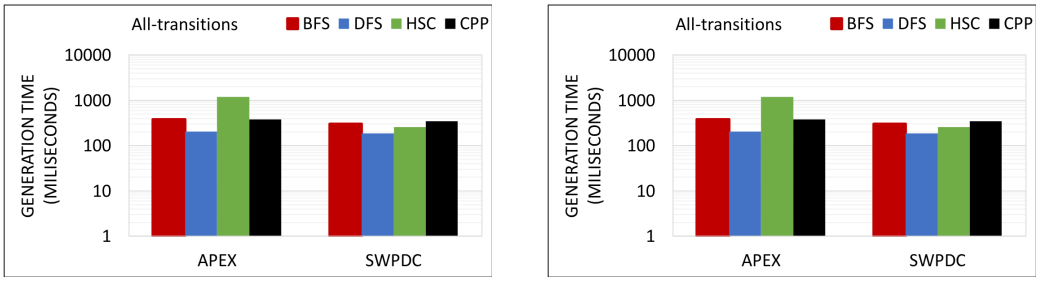
(a) BFS method

(b) DFS method

(c) HSC method

(d) CPP method

Figure 4.18 - Generation time results for real-world FSMs.



(a)

(b)

was used to apply the ATP criterion. 875 random FSMs and two real-world FSMs from embedded software of spatial applications were used in the experiment.

In general, CPP presented better results in all FSMs, both with AT and ATP. This method provides the best results in terms of number of test cases and test suite size,

having a better or similar behavior than HSC in all results. Also, CPP presents low distribution of average length, since its algorithm generates a minimum balanced graph and, so, a minimum test sequence. This method loses to BFS and DFS in terms of length of smallest and largest test case and the average length, but these results were expected since Eulerian methods generate a unique sequence with all transitions of graph.

Looking just at the results of APEX, CPP loses to BFS and DFS. Since APEX represents a unique FSM and this result does not repeat in ATP criterion, it is an indication that for small FSMs, CPP may not be a better choice than other methods. This happens because CPP is a brute-force algorithm. But, analyzing the generation time that CPP used to generate test cases, it is practically the same as BFS. So, even for this case, it is valid to say that BFS and CPP tied in results.

In general, we observed that Eulerian methods (mainly CPP) performed better when applied to large FSMs, and search graphs are better to smaller FSMs. It is valid to mention that, in all cases, CPP was better than HSC. This is because the balancing process of CPP is better than HSC, generating a minimum balanced graph. In our empirical analysis, the balanced graph of CPP is around three times smaller than the balanced graph generated by HSC.

The analyses of this study are not focused in showing, explicitly, what is the best method and criterion to generate test cases, but can guide a test analyst in evaluating which is the best for a particular problem. For example, for space application software, it is more important to generate as many test cases as possible, and in less time. So, CPP may be a better method to be used. If more precision is important, that is more test cases are necessary to cover the model, ATP can be used as an alternative. But, in general, these results showed that the behavior of algorithms naturally always depends on the structure of graph, as shown by the results of random and real-world FSMs. So, even to a space application, it is important to analyze how much precision is needed for test cases and what is the size of graph to generate test cases.

5 CONCLUSION

5.1 General consideration

In general, this dissertation is a conclusion of four years of research in MBT with objective of finding new methods for generation of test cases from FSMs. Initially, this research started with an experimental study investigating switch cover test sets and classic-graph methods (BFS, DFS) from the literature. With this research, we identified that the balacing heuristic of H-Switch Cover was not optimized. Initially, it was necessary to identify other methods for FSMs in the literature that balance the graph. But we generalized our work for other methods that generate test cases from a FSM with the purpose to see what methods are being used in MBT for generation of test cases from FSM. So, for this purpose a Systematic Mapping was conducted. Finally, condiering only graph-based algorithms, an experimental study was conducted for comparing the most used graph-based methods for MBT.

5.2 Contributions

The main contributions of this work are a systematic mapping made to summarize the MBT literature to investigate what are the most used methods to generate test cases for FSM. Besides that, a comparison of the most used methods in random and real-world FSMs was made to define what is the best method in certain cases.

The different analyses about the methods' behavior can support the choice of the best method to fit in a given project. The test expert can identify the trade offs between test case characteristics. It is at the discretion of the testing specialist to decide which algorithm can be used considering the experiment results and characteristics of the system being tested. So, we believe that the results of this dissertation can guide a test analyst, whether in INPE or in industry, to define what is the best method for a problem.

During this work, scientific papers were published in the following events:

MONTEIRO, M. M. ; Souza, E. F. ; ENDO, A. T. ; VIJAYKUMAR, N. L. . A comparative study of algorithms for generating switch cover test sets. In: XV Simpósio Brasileiro de Qualidade de Software (SBQS 2016), 2016, Maceió, Alagoas. XV Simpósio Brasileiro de Qualidade de Software (SBQS 2016), 2016. p. 1-15.

MONTEIRO, M. M. ; Souza, E. F. ; ENDO, A. T. ; VIJAYKUMAR, N. L. .

Analyzing graph-based algorithms employed to generate test cases from finite state machines (to appear). In: 20th IEEE Latin-American Test Symposium (LATS), 2019, Santiago. 20th IEEE Latin-American Test Symposium (LATS), 2019. p. 1-6.

MONTEIRO, M. M. ; Souza, E. F. ; ENDO, A. T. ; VIJAYKUMAR, N. L. . Identifying approaches to generate test cases in model-based testing: a systematic mapping (to appear). In: The 22nd Iberoamerican Conference on Software Engineering (CIbSE) - Experimental Software Engineering Track, 2019, Havana, Cuba. The 22nd Iberoamerican Conference on Software Engineering (CIbSE), 2019. p. 1-14. **(Best Paper Award)**.

5.3 Threats to validity and main limitations

The study selection and data extraction stages were initially performed by the author. In order to reduce some subjectivity, the advisors performed these same stages over a sample of studies. The samples were compared in order to detect possible bias.

Our review was limited by the search terms used in the Scopus database. Although Scopus is considered the largest abstract and citation database, we tried to overcome possible limitations by using backward and forward snowballing. We also considered papers from a control group to calibrate the string. The categorization of papers was based on research questions (method, criterion, formal model and evaluation strategies). Even so, we are possibly leaving some valuable studies out of our analysis, since we considered papers indexed just by the Scopus database, and those obtained from snowballing. However, the studies discussed in this mapping provide an overview of empirical research on outcomes of existing research on MBT.

The limitation of this dissertation is the FSM used in comparing methods. We can not claim that experimental results are true for all FSMs, since the random FSMs do not cover all types of FSMs. Also, even the real-world FSMs are space applications, they are relatively small compared to other experiments of INPE and other institutes. So, it is important to perform other studies with more real-world FSMs. But, we expect that these results show a tendency of methods' behavior with big and small FSMs in the use cases shown.

5.4 Future work

As future work we intend to implement the methods investigated in this project in WEB-PerformCharts[1] (ARANTES et al., 2014), a tool created in the Laboratory of Computng and Applied Mathematics (LABAC), at INPE, that uses formal models to specify a reactive system and generates test cases from a software specification. The following methods are already implemented: DS, UIO, TT and HSC. With incorporation of new methods on the tool, we expect this will allow other researchers conduct new experiments, and use it to generate test cases on their systems.

[1] https://webperformcharts.lcc.unifesspa.edu.br/nova.html

REFERENCES

ALI, S.; BRIAND, L. C.; REHMAN, M. J.-u.; ASGHAR, H.; IQBAL, M. Z. Z.; NADEEM, A. A state-based approach to integration testing based on uml models. **Information and Software Technology**, v. 49, n. 11-12, p. 1087–1106, 2007. 29

AMARAL, A. **Geração de casos de testes para sistemas especificados em Statecharts**. 2005. 162 p. (INPE-14215-TDI/1116). Dissertação (Mestrado em Computação Aplicada) — Instituto Nacional de Pesquisas Espaciais (INPE), São José dos Campos, 2005. Available from: <http://urlib.net/sid.inpe.br/MTC-m13@80/2006/02.14.19.24>. Access in: 2017. xi, 9

AMMANN, P.; OFFUTT, J. **Introduction to software testing**. [S.l.]: Cambridge University Press, 2008. 2, 12

ANBUNATHAN, R.; BASU, A. Dataflow test case generation from uml class diagrams. In: IEEE INTERNATIONAL CONFERENCE ON COMPUTATIONAL INTELLIGENCE AND COMPUTING RESEARCH, 2013. **Proceedings...** IEEE, 2013. p. 1–9. 27

AO, A. S.; PETRENKO, A.; MALDONADO, J. Comparing finite state machine test coverage criteria. **IET Software**, v. 3, n. 2, p. 91–105, 2009. 3, 19

ARANTES, A. O.; SANTIAGO, V. A. de; VIJAYKUMAR, N. L.; SOUZA, E. F. D. Tool support for generating model-based test cases via web. **International Journal of Web Engineering and Technology iiWAS**, v. 9, n. 1, p. 62–96, 2014. 1, 49

ARANTES, A. O.; VIJAYKUMAR, N. L.; JUNIOR, V. A. de S.; GUIMARAES, D. Test case generation for critical systems through a collaborative web-based tool. In: IEEE INTERNATIONAL CONFERENCE ON COMPUTATIONAL INTELLIGENCE FOR MODELLING CONTROL AND AUTOMATION, 2008. **Proceedings...** IEEE, 2013. p. 163–168. 2, 3

BARMI, Z. A.; EBRAHIMI, A. H.; FELDT, R. Alignment of requirements specification and testing: a systematic mapping study. In: INTERNATIONAL CONFERENCE ON SOFTWARE TESTING, VERIFICATION AND VALIDATION WORKSHOPS, 2011. **Proceedings...** IEEE, 2011. p. 476–485. 18

BELLI, F.; BEYAZIT, M.; TAKAGI, T.; FURUKAWA, Z. Model-based mutation testing using pushdown automata. **IEICE Transactions on Information and Systems**, v. 95, n. 9, p. 2211–2218, 2012. 29

_____. Mutation testing of "go-back" functions based on pushdown automata. In: INTERNATIONAL CONFERENCE ON SOFTWARE TESTING, VERIFICATION AND VALIDATION, 2011. **Proceedings...** IEEE, 2013. p. 249–258. 29

BELLI, F.; ENDO, A. T.; LINSCHULTE, M.; SIMAO, A. A holistic approach to model-based testing of web service compositions. **Software: Practice and Experience**, v. 44, n. 2, p. 201–234, 2014. 27

BELLI, F.; HOLLMANN, A. Test generation and minimization with basic statecharts. In: SYMPOSIUM ON APPLIED COMPUTING, 2008. **Proceedings...** ACM, 2008. p. 718–723. 26

BELLI, F.; HOLLMANN, A.; KLEINSELBECK, M. A graph-model-based testing method compared with the classification tree method for test case generation. In: INTERNATIONAL CONFERENCE ON SECURE SOFTWARE INTEGRATION AND RELIABILITY IMPROVEMENT, 3., 2009. **Proceedings...** IEEE, 2009. p. 193–200. 26

BLACK, R. Advanced software test design techniques state diagrams, state tables, and switch coverage. **Testing Experience Magazine**, 2008. 2

BONDY, J.; MURTY, U. **Graduate texts in mathematics: graph theory**. USA: Springer, 2008. 14, 15, 33

BRITO, R. C.; MARTENDAL, D. M.; OLIVEIRA, H. E. M. de. **Máquinas de estados finitos de mealy e moore**. 2003. Available from: <http://www.inf.ufsc.br/~j.barreto/trabaluno/TC_roberta_diogo_henrique.pdf>. 10

CARTAXO, E. G.; ANDRADE, W. L.; NETO, F. G. O.; MACHADO, P. D. Lts-bt: a tool to generate and select functional test cases for embedded systems. In: SYMPOSIUM ON APPLIED COMPUTING, 2008. **Proceedings...** ACM, 2008. p. 1540–1544. 26

CARTAXO, E. G.; MACHADO, P. D.; NETO, F. G. O. On the use of a similarity function for test case selection in the context of model-based testing. **Software Testing, Verification and Reliability**, v. 21, n. 2, p. 75–100, 2011. 29

CHOW, T. S. Testing software design modeled by finite-state machines. **IEEE Transactions on Software Engineering**, n. 3, p. 178–187, 1978. 13

DALAL, S. R.; JAIN, A.; KARUNANITHI, N.; LEATON, J.; LOTT, C. M.; PATTON, G. C.; HOROWITZ, B. M. Model-based testing in practice. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 21., 1999. **Proceedings...** ACM, 1999. p. 285–294. 8

DANG, T.; NAHHAL, T. Coverage-guided test generation for continuous and hybrid systems. **Formal Methods in System Design**, v. 34, n. 2, p. 183–213, 2009. 29

DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. **Introdução ao teste de software**. [S.l.]: Elsevier, 2007. 5, 8

DEVROEY, X.; PERROUIN, G.; SCHOBBENS, P.-Y. Abstract test case generation for behavioural testing of software product lines. In: INTERNATIONAL SOFTWARE PRODUCT LINE CONFERENCE, 2., 2014. **Proceedings...** 2014. p. 86–93. 26

DOROFEEVA, M.; KOUFAREVA, I. Novel modification of the w-method. **Bulletin of the Novosibirsk Computing Center. Series: Computer Science**, n. 18, p. 69–80, 2002. 25

DOROFEEVA, R.; EL-FAKIH, K.; MAAG, S.; CAVALLI, A. R.; YEVTUSHENKO, N. Fsm-based conformance testing methods: a survey annotated with experimental evaluation. **Information and Software Technology**, v. 52, n. 12, p. 1286–1297, 2010. 3, 14, 19

DOROFEEVA, R.; EL-FAKIH, K.; YEVTUSHENKO, N. An improved conformance testing method. In: INTERNATIONAL CONFERENCE ON FORMAL TECHNIQUES FOR NETWORKED AND DISTRIBUTED SYSTEMS, 2005. **Proceedings...** 2005. p. 204–218. 9

EDMONDS, J.; JOHNSON, E. L. Matching, euler tours and the chinese postman. **Mathematical Programming**, v. 5, n. 1, p. 88–124, 1973. 14, 33

EL-FAR, I. K.; WHITTAKER, J. A. Model-based software testing. In: MARCIANIK, J. J. E. (Ed.). **Encyclopedia of software engineering**. [S.l.]: Wiley, 2001. 1

ELLSBERGER, J.; HOGREFE, D.; SARMA, A. **SDL: formal object-oriented language for communicating systems**. [S.l.]: Prentice Hall, 1997. 2, 8

ENDO, A. T.; SIMAO, A. Evaluating test suite characteristics, cost, and effectiveness of fsm-based testing methods. **Information and Software Technology**, v. 55, n. 6, p. 1045–1062, 2013. 3, 7, 19, 34

GILL, A. et al. **Introduction to the theory of finite-state machines**. [S.I.]: McGraw-Hill, 1962. 8

GONENC, G. A method for the design of fault detection experiments. **IEEE transactions on Computers**, v. 100, n. 6, p. 551–558, 1970. 13

GURBUZ, H. G.; TEKINERDOGAN, B. Model-based testing for software safety: a systematic mapping study. **Software Quality Journal**, p. 1–46, 2017. 18

HAREL, D. Statecharts: a visual formalism for complex systems. **Science of Computer Programming**, v. 8, n. 3, p. 231–274, 1987. xi, 2, 8, 10, 11

HESSEL, A.; PETTERSSON, P. A global algorithm for model-based test suite generation. **Electronic Notes in Theoretical Computer Science**, v. 190, n. 2, p. 47–59, 2007. 25, 27

INSTITUTE OF ELECTRIC AND ELECTRONIC ENGINEERS. IEEE Standard 610.12-1990: standard glossary of software engineering terminology. New York, 1990. 5, 6

_____. IEEE Standard 1012-2004: standard for software verification and validation. New York, 2004. 6

JULLIAND, J.; KOUCHNARENKO, O.; MASSON, P. A.; VOIRON, G. Test generation from event system abstractions to cover their states and transitions. **Programming and Computer Software**, v. 44, n. 1, p. 1–14, 2018. 26

KHAN, M. U.; IFTIKHAR, S.; IQBAL, M. Z.; SHERIN, S. Empirical studies omit reporting necessary details: a systematic literature review of reporting quality in model based testing. **Computer Standards & Interfaces**, 2017. 18

KITCHENHAM, B. A. **Guidelines for performing systematic literature reviews in software engineering**. Durham: Keele University, 2007. Technical Report EBSE-2007-01. 3, 21

LEE, D.; YANNAKAKIS, M. Principles and methods of testing finite state machines-a survey. **Proceedings of the IEEE**, v. 84, n. 8, p. 1090–1123, 1996. 2, 8

LI, N.; OFFUTT, J. Test oracle strategies for model-based testing. **IEEE Transactions on Software Engineering**, v. 43, n. 4, p. 372–395, 2017. 27

LIPSCHUTZ, S.; LIPSON, M. **Matemática discreta**. 2. ed. Porto Alegre: Bookman, 1997. 16

MAJEED, S.; RYU, M. Model-based replay testing for event-driven software. In: ANNUAL ACM SYMPOSIUM ON APPLIED COMPUTING, 31., 2016. **Proceedings...** ACM, 2016. p. 1527–1533. 28

MALDONADO, J. C. Critérios potenciais usos: uma contribuição ao teste estrutural de software. [S.I.]: FEE, 1991. 7

MARIANO, M. M.; SOUZA, É. F.; ENDO, A. T.; VIJAYKUMAR, N. L. A comparative study of algorithms for generating switch cover test sets. In: SIMPÓSIO BRASILEIRO DE QUALIDADE DE SOFTWARE, 15., 2016. **Anais...** Maceió, 2016. xi, 15, 16, 17, 20

MARIANO, M. M.; SOUZA, E. F.; ENDO, A. T.; VIJAYKUMAR, N. L. Identifying approaches to generate test cases in model-based testing: a systematic mapping. In: IBEROAMERICAN CONFERENCE ON SOFTWARE ENGINEERING, 22., 2019. **Proceedings...** Havana, 2019. p. 1–14. 21

MARTINS, E.; SABIÃO, S. B.; AMBROSIO, A. M. Condata: a tool for automating specification-based test case generation for communication systems. **Software Quality Journal**, v. 8, n. 4, p. 303–320, 1999. 3

MARUCCI, R. A.; FABBRI, S.; MALDONADO, J. C.; TRAVASSOS, G. H. Oorts/prodes: definição de técnicas de leitura para um processo de software orientado a objetos. In: SIMPÓSIO BRASILEIRO DE QUALIDADE DE SOFTWARE, 1., 2002. **Anais...** 2002. 6

MATHUR, A. P. **Foundations of software testing**. [S.l.]: Pearson Education, 2008. 12

MYERS, G. J.; SANDLER, C. **The art of software testing**. [S.l.]: John Wiley & Sons, 2004. 1

MYERS, G. J.; SANDLER, C.; BADGETT, T. **The art of software testing**. [S.l.]: John Wiley & Sons, 1976. 1, 5

NAITO, S. **Fault detection for sequential machines by transition-tours**. [S.l.: s.n.], 1981. 13, 16, 33

NEUMANN, F. Expected runtimes of evolutionary algorithms for the eulerian cycle problem. In: CONGRESS ON EVOLUTIONARY COMPUTATION, 2001. **Proceedings...** IEEE, 2004. p. 904–910. 16, 33

OFFUTT, J.; ABDURAZIK, A. Generating tests from uml specifications. In: INTERNATIONAL CONFERENCE ON THE UNIFIED MODELING LANGUAGE, 2., 1999, Fort Collins, CO. **Proceedings...** 1999. p. 416–429. 22

PETERSEN, K.; VAKKALANKA, S.; KUZNIARZ, L. Guidelines for conducting systematic mapping studies in software engineering: an update. **IEEE Transactions on Visualization and Computer Graphics**, v. 64, p. 1–18, 2015. 21

PIMONT, S.; RAULT, J.-C. A software reliability assessment based on a structural and behavioral analysis of programs. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 2., 1976. **Proceedings...** IEEE, 1976. p. 486–491. 2, 3, 15, 33

PINHEIRO, A. C.; SIMÃO, A.; AMBROSIO, A. M. Fsm-based test case generation methods applied to test the communication software on board the Itasat university satellite: a case study. **Journal of Aerospace Technology and Management**, v. 6, n. 4, p. 447–461, 2014. 2

PONTES, R. P.; VÉRAS, P. C.; AMBROSIO, A. M.; VILLANI, E. Contributions of model checking and cofi methodology to the development of space embedded software. **Empirical Software Engineering**, v. 19, n. 1, p. 39–68, 2014. 2

PRESSMAN, R. S. **Software engineering: a practitioner's approach**. [S.l.]: Palgrave Macmillan, 2005. 6

PROCH, S.; MISHRA, P. Directed test generation for hybrid systems. In: INTERNATIONAL SYMPOSIUM ON QUALITY ELECTRONIC DESIGN, 15., 2014. **Proceedings...** 2014. p. 156–162. 29

RAPPS, S.; WEYUKER, E. J. Selecting software test data using data flow information. **IEEE Transactions on Software Engineering**, n. 4, p. 367–375, 1985. 2

ROCHA, C. R.; MARTINS, E. A method for model based test harness generation for component testing. **Journal of the Brazilian Computer Society**, v. 14, n. 1, p. 7–23, 2008. 26

SATPATHY, M.; YEOLEKAR, A.; PERANANDAM, P.; RAMESH, S. Efficient coverage of parallel and hierarchical stateflow models for test case generation. **Software Testing, Verification and Reliability**, v. 22, n. 7, p. 457–479, 2012. 29

SIAVASHI, F.; TRUSCAN, D. Environment modeling in model-based testing: concepts, prospects and research challenges: a systematic literature review. In: INTERNATIONAL CONFERENCE ON EVALUATION AND ASSESSMENT IN SOFTWARE ENGINEERING, 19., 2015, New York. **Proceedings...** ACM, 2014. p. 30:1–30:6. Available from: <http://doi.acm.org/10.1145/2745802.2745830>. 18

SIDHU, D. P.; LEUNG, T.-K. Formal methods for protocol testing: a detailed study. **IEEE Transactions on Software Engineering**, v. 15, n. 4, p. 413–426, 1989. 2, 13

SOUZA, É. F.; JÚNIOR, V. A. S.; GUIMARAES, D.; VIJAYKUMAR, N. L. Evaluation of test criteria for space application software modeling in statecharts. In: INTERNATIONAL CONFERENCE ON COMPUTATIONAL INTELLIGENCE FOR MODELLING CONTROL AND AUTOMATION, 2008. **Proceedings...** 2008. p. 157–162. 2, 3, 14, 19, 33

SOUZA, É. F.; JÚNIOR, V. A. S.; VIJAYKUMAR, N. L. H-switch cover: a new test criterion to generate test case from finite state machines. **Software Quality Journal**, v. 25, n. 2, p. 373–405, 2017. 1, 3, 16, 20, 25, 33, 35

SOUZA, S. d. R. S. d. **Validação de especificações de sistemas reativos: definição e análise de critérios de teste**. Tese (Doutorado em Física Computacional) — Universidade de São Paulo, São Carlos, 2000. Available from: <http://www.teses.usp.br/teses/disponiveis/76/76132/tde-27112008-085629/>. 19

TAKAGI, T.; OYAIZU, N.; FURUKAWA, Z. Concurrent n-switch coverage criterion for generating test cases from place/transition nets. In: INTERNATIONAL CONFERENCE ON COMPUTER AND INFORMATION SCIENCE, 9., 2010. **Proceedings...** 2010. p. 782–787. 26

TRETMANS, J. Model based testing with labelled transition systems. In: **HIERONS, R.M.; BOWEN, J.P.; HARMAN, M. (Ed.). Formal methods and testing**. [S.l.]: Springer, 2008. p. 1–38. 2, 8

UTTING, M.; LEGEARD, B. **Practical model-based testing: a tools approach**. San Francisco, CA, USA: Morgan Kaufmann, 2010. 2, 8

VIJAYKUMAR, N. **Statecharts: their use in specifying and dealing with Performance Models**. Tese (Doutorado em Informática) — Instituto Tecnológico de Aeronáutica, São José dos Campos, 1999. 2

VU, T.-D.; HUNG, P. N.; NGUYEN, V.-H. A method for automated test data generation from sequence diagrams and object constraint language. In: INTERNATIONAL SYMPOSIUM ON INFORMATION AND COMMUNICATION TECHNOLOGY, 2015. **Proceedings...** 2015. p. 335–341. 26

WANG, W.; SAMPATH, S.; LEI, Y.; KACKER, R.; KUHN, R.; LAWRENCE, J. Using combinatorial testing to build navigation graphs for dynamic web applications. **Software Testing, Verification and Reliability**, v. 26, n. 4, p. 318–346, 2016. 29

YAO, J.; WANG, Z.; YIN, X.; SHIYZ, X.; WU, J. Formal modeling and systematic black-box testing of sdn data plane. In: INTERNATIONAL CONFERENCE ON NETWORK PROTOCOLS, 22., 2014. **Proceedings...** IEEE, 2014. p. 179–190. 28

ZANDER, J.; SCHIEFERDECKER, I.; MOSTERMAN, P. J. **Model-based testing for embedded systems**. [S.l.]: CRC press, 2017. 2