

TEST CASE GENERATION FROM STATE MACHINE WITH OCL
CONSTRAINTS USING SEARCH-BASED TECHNIQUES

ANEESA ALI ALI SAEED

FACULTY OF COMPUTER SCIENCE AND INFORMATION
TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2017

TEST CASE GENERATION FROM STATE MACHINE
WITH OCL CONSTRAINTS USING SEARCH-BASED
TECHNIQUES

ANEESA ALI ALI SAEED

THESIS SUBMITTED IN FULFILMENT OF THE
REQUIREMENTS FOR THE DEGREE OF DOCTOR
OF PHILOSOPHY

FACULTY OF COMPUTER SCIENCE AND
INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2017

UNIVERSITI MALAYA

ORIGINAL LITERARY WORK DECLARATION

Name of Candidate: ANEESA ALI ALI SAEED (I.C./Passport No.:)

Registration/Matrix No.: WHA130037

Name of Degree: Doctor of Philosophy (PhD)

Title of Project Paper/Research Report/Dissertation/Thesis ("this Work"):

Test Case Generation from State Machine with OCL Constraints using Search-Based Techniques

Field of Study: Software Testing

I do solemnly and sincerely declare that:

- (1) I am the sole author/writer of this Work;
- (2) This work is original;
- (3) Any use of any work in which copyright exists was done by way of fair dealing and for permitted purposes and any excerpt or extract from, or reference to or reproduction of any copyright work has been disclosed expressly and sufficiently and the title of the Work and its authorship have been acknowledged in this Work;
- (4) I do not have any actual knowledge nor do I ought reasonably to know that the making of this work constitutes an infringement of any copyright work;
- (5) I hereby assign all and every rights in the copyright to this Work to the University of Malaya ("UM"), who henceforth shall be owner of the copyright in this Work and that any reproduction or use in any form or by any means whatsoever is prohibited without the written consent of UM having been first had and obtained;
- (6) I am fully aware that if in the course of making this Work I have infringed any copyright whether intentionally or otherwise, I may be subject to legal action or any other action as may be determined by UM.

Candidate's Signature

Date

Subscribed and solemnly declared before,

Witness's Signature

Date

Name:

Designation:

ABSTRACT

Software testing consumes half of the entire software development cost where test case generation is the most cost consuming activity in the whole process. The emergence of automatic test case generation has helped in reducing the cost eventually. Recently, model-based testing (MBT) for automatic test case generation gains interest in industry and academia due to its provision of systematic, automated, and comprehensive testing. One of the input models for MBT is state machine model which currently widely utilized to model embedded systems. Generating test cases from Unified Modeling Language (UML) state machine models has two major challenges: generating feasible paths, and generating data to satisfy the paths. The existing infeasible path detection methods are restricted to extended finite state machine (EFSM) models with integer data type only. For detecting infeasible paths that involve Object Constraints Language (OCL) constraints, new method is needed to cover all the sophisticated constructs of OCL. For test data generation, the existing search-based techniques (SBTs) have been applied to satisfy only one OCL constraint by time. In order to generate optimal data to satisfy whole constraints in the feasible path, new method with SBTs is necessary to satisfy the whole constraints at the same time of the whole path executing. This thesis presents a method for generating feasible test cases from UML state machine models with OCL constraints. One contribution of this thesis is developing an efficient technique for detecting automatically infeasible paths that contain transitions with conflicted OCL constraints. A model-driven approach was used for generating abstract test cases from the feasible paths. This model driven approach was integrated with the proposed infeasible path detection method which based on analyzing various OCL constructs and operations. The second contribution of this thesis is developing an accurate search-based test data generator for generating

automatically optimal test data to satisfy the whole constraints in the path. In the proposed search-based test data generator, a whole constraints analyzer and a fitness function that evolves itself based on the error feedback were proposed. The whole constraint analyzer and the fitness function were combined with four SBTs (genetic algorithm, evolutionary algorithm, simulating annealing, and quantum genetic algorithm). Case study evaluation was conducted based on three industrial open source case studies in order to evaluate empirically the significant of the performance of the proposed method. The results were statically analyzed using t-test to show the significance of the proposed method compared to the existing methods. The results show that the proposed infeasible path detection method was efficient and detect 99 percent of the infeasible paths in the three industrial systems. The results of the proposed search-based test data generator show significant performance compared to the existing search-based test data generator.

ABSTRAK

Pengujian perisian menggunakan hampir separuh daripada keseluruhan kos pembangunan perisian. Penjanaan kes ujian adalah aktiviti utama yang menggunakan kos tersebut. Pengenerasian kes ujian secara automatik dari model yang dipanggil ujian berasaskan model (Model-based testing) (MBT), kini sedang mendapat tempat di dalam industri serta akademik berikutan keupayaannya untuk menguji secara sistematik, automatik dan menyeluruh. Salah satu kemungkinan model input yang digunakan untuk MBT ialah mesin keadaan (state machine), yang kini digunakan secara meluas untuk memodelkan system terbenam (embedded systems). Penjanaan kes-kes ujian dari model mesin keadaan UML mempunyai dua cabaran utama: menjana laluan boleh dilaksana, dan menjana data yang memuaskan laluan. Dari literatur, kaedah pengesanan laluan yang tidak boleh dilaksana yang sedia ada adalah terhad untuk mesin keadaan finit lanjutan dengan jenis data integer sahaja. Untuk mengesan laluan yang tidak boleh dilaksana yang mengandungi kekangan OCL, kaedah baru diperlukan untuk menampung segala binaan canggih OCL. Bagi penjanaan data ujian, kaedah berasaskan pencarian sedia ada telah digunakan untuk memuaskan hanya satu kekangan OCL dalam satu masa. Untuk menjana data yang optimal untuk memuaskan keseluruhan kekangan di dalam laluan boleh dilaksana, kaedah baru dengan SBTs adalah perlu untuk memuaskan keseluruhan kekangan, dan pada masa yang sama, keseluruhan laluan pelaksanaan. Tesis ini membentangkan suatu kaedah untuk menjana kes-kes ujian boleh dilaksana dari model mesin keadaan UML dengan kekangan OCL. Sumbangan pertama tesis ini adalah teknik yang berkesan untuk mengesan secara automatik laluan tidak boleh dilaksana yang mempunyai peralihan yang mengandungi konflik kekangan OCL. Pendekatan yang didorong oleh model digunakan untuk penjanaan kes ujian abstrak dari laluan boleh dilaksana. Pendekatan ini disepadukan

dengan analisis statik yang dicadangkan dimana ia menganalisa variasi binaan dan operasi OCL untuk mengesan laluan tidak boleh dilaksana. Sumbangan kedua tesis ini ialah satu teknik penjanaan data ujian optimal secara automatik untuk memuaskan keseluruhan kekangan di dalam laluan. Di dalam penjana data ujian kami, keseluruhan penganalisa kekangan dan fungsi kecergasan berevolusi sendiri berdasarkan maklumbalas ralat telah dicadangkan. Keseluruhan penganalisa kekangan dan fungsi kecerdasan telah digabungkan dengan empat SBTs (algoritma genetik, algoritma evolusi, simulasi penyepuhlindungan dan algoritma genetik kuantum). Penilaian kajian kes dijalankan berdasarkan tiga kajian kes sumber terbuka industri untuk menilai secara empirikal kelebihan prestasi kaedah kami. Keputusan dianalisa secara statistic menggunakan t-test untuk menunjukkan kelebihan kaedah kami dibandingkan dengan kaedah yang sedia ada. Keputusan menunjukkan bahawa kaedah pengesanan laluan tidak boleh dilaksana kami adalah berkesan dan sekitar 99 peratus laluan tidak boleh dilaksana dikesan di dalam tiga sistem industri. Keputusan penjanaan data ujian kami menunjukkan kelebihan prestasi dibandingkan dengan penjana data ujian berdasarkan pencarian yang terkini.

ACKNOWLEDGEMENTS

First and foremost, all praise belongs to Almighty Allah, the Lord of the Universe, who has enabled me to accomplish and complete this work successfully.

I would like to extend my sincere gratitude to my supervisors Dr. Siti Hafizah Ab Hamid and Dr. Asmiza Abdul Sani for their constant support, patience and constructive comments. The guidance you have bestowed has been truly beneficial beyond an academic perspective. You have provided many opportunities for me to expand my knowledge and experience that have been crucial to my academic career. You have challenged me and never let me settle for anything other than my best; for that I am grateful.

The constant support and encouragement my family has given throughout this process has been a blessing. I would like to thank my mother for her support in my pursuit of a PhD degree. The unconditioned love, support and encouragement of my husband, Bashar, has been my biggest motivation to finish this thesis. Therefore, I dedicate this work to them.

TABLE OF CONTENTS

Abstract	iii
Abstrak	v
Acknowledgements	vii
Table of Contents	viii
List of Figures	xii
List of Tables.....	xv
List of Appendices	xviii
CHAPTER 1: INTRODUCTION	1
1.1 Overview	1
1.2 Problem Statement	5
1.3 Research Objectives	6
1.4 Scope of Work.....	7
1.5 Proposed Methodology	7
1.6 Thesis Outline	10
CHAPTER 2: STATE-BASED TESTING AND SEARCH-BASED TECHNIQUES FOR MODEL-BASED TESTING: A REVIEW	13
2.1 Background	14
2.1.1 Unified Modeling Language (UML)	14
2.1.2 Object Constraint Language (OCL)	16
2.1.3 Model-based Testing (MBT)	16
2.1.4 Search-based Test Data Generation.....	19
2.1.5 State-based Testing	23
2.2 Search-based Techniques for Model-based Testing: Systematic Review	26
2.2.1 Systematic Review Methodology	26
2.2.2 Taxonomy	32

2.2.3	State-of-the-Art	46
2.2.4	Challenges	64
2.3	State-based Testing: Review	68
2.3.1	Taxonomy	68
2.3.2	UML State-based Testing	76
2.3.3	Penalty-based Infeasible Path Detection.....	79
2.4	Research Gaps.....	82
2.4.1	UML state-based testing.....	82
2.4.2	Infeasible path detection.....	82
2.4.3	Search-based test data generation in MBT.....	83
 CHAPTER 3: PERFORMANCE ANALYSIS FOR THE INFEASIBLE DETECTION METHOD AND SEARCH-BASED TEST DATA GENERATOR.....		84
3.1	Case Study Design	85
3.1.1	Case Study Objectives.....	85
3.1.2	Case and Subject Selection.....	85
3.1.3	Data collection procedures	88
3.2	Case Study Execution	89
3.2.1	Preparation of Input Models.....	89
3.2.2	Infeasible Paths Detection	90
3.2.3	Search-based Test Data Generation.....	91
3.2.4	Test Case Generation and Execution	92
3.3	Results and Discussion	95
3.3.1	Model-Driven approach	95
3.3.2	Infeasible Path Detection.....	96
3.3.3	Search-based Test Data Generation.....	97
3.4	Threats to Validity.....	101

3.5	Conclusion	102
CHAPTER 4: METHOD FOR GENERATING FEASIBLE EXECUTABLE TEST CASES FROM UML STATE MACHINE MODELS WITH OCL CONSTRAINTS		104
4.1	Proposed Method for Feasible Executable Test Case Generation.....	105
4.1.1	Model-driven Path Generator	105
4.1.2	Search-based Test Data Generator.....	112
4.2	Significance of the Proposed Method	118
4.3	Conclusion	120
CHAPTER 5: EVALUATION OF THE PROPOSED METHOD		121
5.1	Case Study Method	122
5.1.1	Research Questions for the proposed method	122
5.1.2	Case and Subject Selection for the Proposed Method.....	123
5.1.3	Data collection procedures for the proposed method	124
5.2	Case Study Execution for the proposed method	124
5.2.1	Preparation of Input Models.....	125
5.2.2	Test Case Generation and Execution	125
5.2.3	Comparison Baseline.....	127
5.3	Empirical Evaluation Results of the Proposed Method	128
5.3.1	Infeasible Path Detection.....	128
5.3.2	Search-based Test Data Generator.....	129
5.3.3	Comparison Results	135
5.3.4	Overall Discussion.....	142
5.4	Threats to Validity.....	146
5.5	Conclusion	146
CHAPTER 6: CONCLUSIONS AND FUTURE WORK		148
6.1	Restatement of Research Aim	148

6.2 Contributions	151
6.3 Significance of the Work.....	154
6.4 Limitation and Future Work.....	156
6.5 International Scholarly Publications	157
References	159
Appendices.....	178

University of Malaya

LIST OF FIGURES

Figure 1.1: Research Methodology.	8
Figure 1.2: Schematic presentation of the thesis outline.....	12
Figure 2.1: Semantic presentation of chapter 2 outline.....	14
Figure 2.2: An Example of UML state machine model.	15
Figure 2.3: The process of model-based testing.	17
Figure 2.4: Timeline graph of the key development between MBT and SBTs	21
Figure 2.5: Systematic Literature Review Methodology	27
Figure 2.6: The output of search process and study selection steps	29
Figure 2.7: Taxonomy of classifying the applications of SBTs for MBT	33
Figure 2.8: Full structural view of problem category	38
Figure 2.9: Full structural view of solution category	43
Figure 2.10: Full structural view of evaluation category	46
Figure 2.11: Distribution of papers based on the purpose category.	48
Figure 2.12: Distribution of papers based on the application domain sub-category. ..	49
Figure 2.13: Distribution of papers based on the model type sub-category.	50
Figure 2.14: Distribution of papers based on the modeling language sub-category. ...	50
Figure 2.15: Distribution of papers based on the testing level sub-category.	50
Figure 2.16: Distribution of papers based on the dimensionality sub-category.	51
Figure 2.17: Distribution of papers based on the adequacy criteria sub-category.	52
Figure 2.18: Distribution of papers based on the quality attributes sub-category.	52
Figure 2.19: Distribution of papers based on the constraints sub-category.	53
Figure 2.20: Distribution of papers based on the model transformation sub-category.	54
Figure 2.21: Distribution of papers based on the fitness function sub-category.	55
Figure 2.22: Distribution of papers based on the Type of Search-Based Techniques sub-category.	56

Figure 2.23: Distribution of papers based on the constraint handling sub-category. . .	57
Figure 2.24: Distribution of papers based on the landscape visualization sub-category.	57
Figure 2.25: The Taxonomy of State-based Testing.....	69
Figure 2.26: An example of the path generation from UML state machine models	81
Figure 3.1: Semantic presentation of chapter 3 outline.....	84
Figure 3.2: The model transformation approach architecture used for model-based testing.	93
Figure 3.3: The transition tree (test model) metamodel.....	94
Figure 3.4: The results of the detection method.....	97
Figure 3.5: Box plot graph of success rate for RS, GA and SA in both case studies..	101
Figure 4.1: Semantic presentation of chapter 4 outline.....	104
Figure 4.2: The proposed method.	106
Figure 4.3: The proposed method of infeasible path detection.	107
Figure 4.4: The proposed method for optimizing whole constraints.	114
Figure 5.1: Semantic presentation of chapter 5 outline.....	122
Figure 5.2: The results of the proposed infeasible path detection method.....	129
Figure 5.3: The results of success rate of the proposed method with GA, EA, SA, and QGA techniques in case study 1 (CSM).	130
Figure 5.4: The results of success rate of the proposed method with GA, EA, SA, and QGA techniques in case study 2 (TIS).....	131
Figure 5.5: The results of success rate of the proposed method with GA, EA, SA, and QGA techniques in case study 3 (EU-Rent).	131
Figure 5.6: The results of the proposed infeasible path detection method with Kalaji approach.	136
Figure 5.7: The results of success rate of the proposed method and EsOCL solver with four SBTs in case study 1 (CSM) from Table 5.6.	137
Figure 5.8: The results of success rate of the proposed method and EsOCL solver with four SBTs in case study 2 (TIS) from Table 5.7.....	139

Figure 5.9: The results of success rate of the proposed method and EsOCL solver with four SBTs in case study 3 (EU-Rent) from Table 5.8. 140

Figure 5.10: Box plot graph of success rate for the proposed method with four SBTs in all case studies. 144

University of Malaya

LIST OF TABLES

Table 2.1: The detailed result of the optimization process sub-category.....	58
Table 2.2: Used dataset for evaluation	59
Table 2.3: The distribution of papers based on the baseline sub-category.....	60
Table 2.4: The distribution of papers based on the effectiveness measures sub-category.	60
Table 2.5: The distribution of papers based on the cost measurements sub-category.	61
Table 2.6: The distribution of papers based on the statistical test sub-category.	61
Table 2.7: The cross analysis between the adequacy criteria sub-category with the fitness function, the Type of SBTs, the landscape visualization, and the optimization process sub-categories.....	63
Table 2.8: The cross analysis between the constraint and the constraint handling sub-categories.	64
Table 2.9: Summary of studies focused on path generation from UML state machine with OCL constraints.....	79
Table 2.10: Summary of studies focused on data generation from UML state machine with OCL constraints.....	80
Table 2.11: Summary of Infeasible path detection studies.	82
Table 3.1: Case studies description.	90
Table 3.2: Description of OCL constraints.	90
Table 3.3: Configuration of search-based techniques.	95
Table 3.4: Result of model-driven approach.	96
Table 3.5: The results of successful rate and generation time for each GA, SA and RS techniques in case study 1 (CSM).	98
Table 3.6: The results of successful rate and generation time for each GA, SA and RS techniques in case study2 (TIS).	99
Table 3.7: The results of the paired t-test based on successful rate.	100
Table 3.8: The results of the paired t-test based on generation time.	100
Table 4.1: Values of K.....	108

Table 4.2: The penalty value of the basic relations.....	109
Table 4.3: The penalty value of the boolean relations.	110
Table 4.4: The penalty value of the operations of collection data type.....	111
Table 4.5: The distance calculation of boolean operations.	115
Table 4.6: The fitness calculation of the basic numerical relations.	116
Table 5.1: Case studies description.	125
Table 5.2: Configuration of search-based techniques.	127
Table 5.3: The results of generation time of the proposed method with GA, EA, SA, and QGA techniques in case study 1 (CSM).	132
Table 5.4: The results of generation time of the proposed method with GA, EA, SA, and QGA techniques in case study 2 (TIS).....	133
Table 5.5: The results of generation time of the proposed method with GA, EA, SA, and QGA techniques in case study 3 (EU-Rent).....	134
Table 5.6: The results of success rate of the proposed method and OCL solver with GA, EA, SA, and QGA techniques in case study 1 (CSM).....	137
Table 5.7: The results of success rate of the proposed method and EsOCL solver with GA, EA, SA, and QGA techniques in case study 2 (TIS).	138
Table 5.8: The results of success rate of the proposed method and EsOCL solver with GA, EA, SA, and QGA techniques in case study 3 (Eu-Rent).	139
Table 5.9: The results of generation time of the proposed method and EsOCL solver with GA, EA, SA, and QGA techniques in case study 1 (CSM).	141
Table 5.10: The results of generation time of the proposed method and EsOCL solver with GA, EA, SA, and QGA techniques in case study 2 (TIS).	142
Table 5.11: The results of generation time of the proposed method and EsOCL solver with GA, EA, SA, and QGA techniques in case study 3 (EU-Rent).	143
Table 5.12: The results of the paired t-test based on success rate.....	145
Table 5.13: The results of the paired t-test based on generation time.	145
Table A.1: The list of keywords used in the search process.....	178
Table A.2: The classification of the reviewed papers based on the problem category and purpose category.....	179
Table A.3: The classification of the reviewed papers based on the solution category and purpose category.....	192

LIST OF ABBREVIATIONS

Sbts	Search-Based Techniques
MBT	Model-Based Testing
GA	Genetic Algorithm
SA	Simulating Annealing
EA	Evolutionary Algorithm
UML	Unified Modeling Language
OCL	Object Constraint Language
RTP	All Round Trip
QGA	Quantum Genetic Algorithm
EFSM	Extended Finite State Machine
FSM	Finite State Machine
SUT	System Under Test
AT	All Transitions
ATP	All Transitions Pairs
FP	Full Predicate
HC	Hill Climbing
AVM	Alternating Variable Method
TIS	Turn Indicator System
CSM	Ceiling Speed Monitoring
EU-Rent	European Rental System
SBSE	Search-Based Software Engineering
SBST	Search-Based Software Testing
RS	Random Search
EMF	Eclipse Modeling Framework

LIST OF APPENDICES

Appendix A: Analysis Tables.....	178
----------------------------------	-----

University of Malaya

CHAPTER 1: INTRODUCTION

This chapter introduces a holistic view of the research undertaken in this thesis. An overview of the research on test case generation from models with search-based techniques (SBTs) and state the research problem is presented. Moreover, the chapter specifies the aim and objectives of this study and describes the methodology proposed to achieve the aim and objectives.

The remainder of this chapter is as follows. Section 1.1 presents overview for undertaking this research and highlight the significance of the work. Section 1.2 introduces the identified research problem to be addressed in this thesis. Section 1.3 presents the aim and objectives of this study following with proposed methodology in Section 1.5. Finally, Section 1.6 presents the thesis outline.

1.1 Overview

Software testing is the most consuming activity in development cycle, where in the cost of software testing consumes almost half of the entire software development cost (Luo, 2001; Harman, Mansouri, & Zhang, 2009). Therefore, automating software testing process reduces its cost. Automatic generating and executing test cases play main roles in automating the software testing process and reducing significantly the cost as well. Recently, several researches were established in automatic test case generation for ensuring robustness of the software (Iqbal, Arcuri, & Briand, 2012a, 2012b; Ali, Iqbal, Khalid, & Arcuri, 2015; Khurana & Chillar, 2015; Alshahwan & Harman, 2012).

Robustness, as defined by an IEEE Standard (*IEEE Standard Glossary of Software Engineering Terminology*, n.d.), is the degree to which a system or component enable to function correctly in the presence of invalid inputs or stressful environment conditions. A system should be robust enough to handle the possible abnormal situations that occur

in its operating environment, and invalid inputs due to several significant activities in our daily life which are directly or indirectly relied on embedded, control and communication systems (European Union, 2014). For example, the utilization of smart phones and tele-presence systems has been rapidly growing. Assuring the correctness of these systems behavior is very important and such behavior is commonly referred as robustness behavior. Testing of these systems contributes significantly on correcting their functioning, whose behavior is inherently unpredictable. Ensuring the robustness of the system can be achieved by testing the behavior of a system in faulty situations in its operating environment. One option to systematically ensure robustness in testing is to use model-based testing (MBT), which is a systematic, rigorous, automated way of conducting testing, and applied in early stage (modeling stage) compared to its counterpart which are code-based testing and manual testing (Ali, Hemmati, Holt, Arisholm, & Briand, 2010).

MBT aims to produce executable test cases by consistently analyzing the behavioral design models of a software system. MBT makes the testing process more efficient and simple because models are easier in maintenance and also the fault will be discovered in early stage (modeling) that will reduce the cost of fixing the faults. The low cost of test case generation by applying MBT will reduce the cost of the whole testing process. MBT gained increasing interest in both industry and academia and this is visible from several academic studies (Zhan & Clark, 2008; Iqbal et al., 2012b; Lindlar, Windisch, & Wegener, 2010) and industrial projects (D-MINT, n.d.; Feldstein, 2005). Five models were used as the input of MBT in the literature which are activity, use case, sequence, class and state machine models.

State machine model is widely utilized to model the behavior of the most critical and complex system components that exhibit state-driven behavior (L. C. Briand, Labiche, & Wang, 2004). A great number of today's embedded and control systems are modeled by

state machine (Ali, Iqbal, & Arcuri, 2013; Vos et al., 2012; Lindlar et al., 2010). Creating complete and correct state machine models is a critical concern. The language used for modeling state-behavior systems are Unified Modeling Language (UML)(Pender, 2003), and Simulink (Dabney & Harman, n.d.). UML has been developed to support the design of complex object-oriented systems (Sarma & Mall, 2009). Recently it has become a defacto standard modeling language for industrial softwares (Ali, 2011) because it provides a unified, precise, and consistent way to communicate information among different people involved in software development. To develop constraints for models, Object Constraint Language (OCL) is used, which is an extension language for writing constraints on UML models. OCL is widely accepted in the literature (Ali et al., 2013) and the software supports OCL is now growing (Gogolla, Büttner, & Richters, 2007). UML and its sub-language OCL are regarded as central ingredients of model-centric software production. Generating test cases from UML state machine models includes two steps: 1) abstract test case generation and 2) test data generation.

For generating abstract test cases from UML state machine models, several tools and approaches have been proposed for generate abstract test cases (Weiß leder & Schlingloff, 2008; Lefticaru & Ipate, 2008b; Friske & Schlingloff, 2007; Sarma & Mall, 2009; L. Briand, Labiche, & Lin, 2010), but none of them can be extended and configured to various contexts (such as coverage criteria, scripting language, and test models). This drawback motivated the researchers to propose extensible and configured model-driven approach (model transformations) to generate abstract test cases (Ali, Hemmati, et al., 2010) and test oracle (Lamancha, Polo, Caivano, Piattini, & Visaggio, 2013). However, none of these tools and model-driven approach included infeasible path detection. Detecting infeasible paths is important part due to no data can be generated for executing the infeasible paths. Therefore, the test data generation for infeasible paths is a time-consuming

task.

For test data generation, six techniques were proposed in the literature: symbolic execution (Prelguskas & Bareisa, 2012), model checker (Swarup Mohalik, Ambar A. Gadkari, Anand Yeolekar & Ramesh, 2014; Hamon, De Moura, & Rushby, 2004), theorem prover (Cantenot, Ambert, & Bouquet, 2014; Brucker & Wolff, 2013), constraint solving (Vishal, Kovacioglu, Kherazi, & Mousavi, 2012), random search (Huang, Liu, Xie, & Chen, 2015), and SBTs (Ali et al., 2015; Blanco, Tuya, & Adenso-Díaz, 2009; Iqbal et al., 2012a). However, each of the first five techniques has limitations. Specifically, symbolic execution suffers from three fundamental problems, that limit its effectiveness on real world software, which are path explosion, path divergence and solving only small and linear models. Model checking suffers from the space explosion and run out of memory when the input models are complex. The problems of theorem prover and constraint solving is undecidable for non-trivial domains of inputs and need to write the models in specific formula such as encoded formalism. Because of the simplicity of random search technique, it is not efficient when applied for complex systems. Therefore, recently researchers applied SBTs which are efficient when applied for complex models. This efficient performance of SBTs is due to they utilize heuristics to obtain optimal or near optimal solutions for solving the problems that have large search space at an affordable computational time cost. SBTs outperformed other test data generation techniques such as model checking (Nilsson, Offutt, & Mellin, 2006; Hänsel, Rose, Herber, & Glesner, 2011; Wenzel, Kirner, Rieder, & Puschner, 2008), random (Ali et al., 2013, 2015; Harman & McMinn, 2010), and constraint solver (Ali et al., 2013). Recently, SBTs have been applied for automatically solving OCL constraints in a test case (Ali et al., 2013, 2015). However, only one constraint can be solved at one time, that leads to generate conflict data which does not satisfy all the constraints of the test case at one execution run, and this is not

practical in industrial context.

Therefore, it is essential to study the test case generation from UML state machine models with OCL constraints and develop a solution to generate feasible test cases with optimal test data from UML state machine with OCL constraints. This solution is deemed improving infeasible path detection and test data generator.

1.2 Problem Statement

Two challenges in generating executable test cases for UML state machine models with OCL constraints, which are 1) generating feasible abstract test cases and 2) generating optimal test data for satisfying whole test case constraints to execute the generated test cases. For the first challenge of feasible abstract test cases generation, a recent extensible and configured model-driven approach (model transformations) is proposed to generate executable test cases (Ali, Hemmati, et al., 2010) from UML models. However, this model-driven approach generates a high number of infeasible test cases because it does not include the infeasible path detection mechanism. This existing model-driven approach still needs to be enhanced to generate only feasible test cases. For the infeasible path detection perspective, several studies proposed infeasibility detection approaches for extended finite state machine (EFSM) models (A. S. Kalaji, Hierons, & Swift, 2011; K. Derderian, Hierons, Harman, & Guo, 2009; Yang, Chen, Xu, Wong, & Zhang, 2011; Shirole, 2011; Núñez, Merayo, Hierons, & Núñez, 2012). The recent approaches in finding infeasible paths (A. S. Kalaji et al., 2011; Yang et al., 2011) rely on the penalty values which are limited to integer data type only with its basic relations (<, >, <=, >=, =, <>). To apply these existing EFSM-based infeasible path detection for UML state machine models with OCL constraints, the UML models must be transformed into EFSM models. However, this is not suitable for the UML models with OCL that contain complex guards conditions (such as includes, if else then, implies) and data types (such as enumeration, tuples, StateIsOcl)

because EFSM includes constraints with the basic data types only (such as Integer and Boolean).

For the second challenge of test data generation for solving OCL constraints in UML state machine models, the latest endeavor is SBTs because of their capability for solving the problems that have large search space at an affordable computational time cost. Applying these SBTs for solving OCL constraints is non-trivial task because a proper fitness function should be carefully developed to solve the sophisticated constructs of OCL. Recent OCL solver (Ali et al., 2013, 2015) was proposed based on a set of SBTs that calculate the branch distance of each OCL data type and operation. However, this OCL solver processes one constraint at one time only. This limits the optimality of the generated data that satisfy all the OCL constraints of a test case because all the constraints should be satisfied by the generated data at one time when executing the test case.

1.3 Research Objectives

This research is undertaken with the aim to develop a method to generate executable feasible test cases from UML state machine models with OCL constraints. The aim is achieved by fulfilling the following objectives:

- To review the current state-based testing and the applications of search-based techniques for model-based testing to generate executable test cases.
- To analyze the existing model-driven approach with infeasible path detection for generating abstract test cases and search-based test data generation for satisfying all OCL constraints in each abstract test case.
- To develop a method with an infeasible path detection method and a fitness function that evolves itself using error feedback for satisfying whole OCL constraints in each abstract test case.

- To evaluate the ability of the proposed method to detect infeasible paths and generate optimal data using three industrial case studies of embedded systems.

1.4 Scope of Work

The scope of this thesis is test case generation from state machine models with OCL constraints using model-driven approach with SBTs. This thesis excludes other UML models such as sequence, activity, and the rest. Furthermore, other test data generation techniques are excluded from this research such as model checking and symbolic execution. The fattening and checking the consistency of the state models are out of this thesis scope as well.

1.5 Proposed Methodology

The following steps as shown in Figure 1.1 were followed in order to achieve the aim and objectives of this research.

- A comprehensive review and synthesis of the recent applications of SBTs for MBT were undertaken to identify the impact of SBTs on test case generation from models referring to scholarly digital libraries, particularly IEEE, ScienceDirect, Wiley, Springer, Google Scholar, and ACM. The impact of the state-based testing was also reviewed, and a taxonomy for the state-based testing was proposed. Several research gaps were identified through literature and The problems to be addressed in this thesis were also identified.
- The identified problems were investigated and their significance was verified through empirical case study analysis using UML state machine models of two industrial embedded systems. Using series of experiments on model-driven with infeasible path detection method, SBTs and non-SBTs, the performance was evaluated to verify

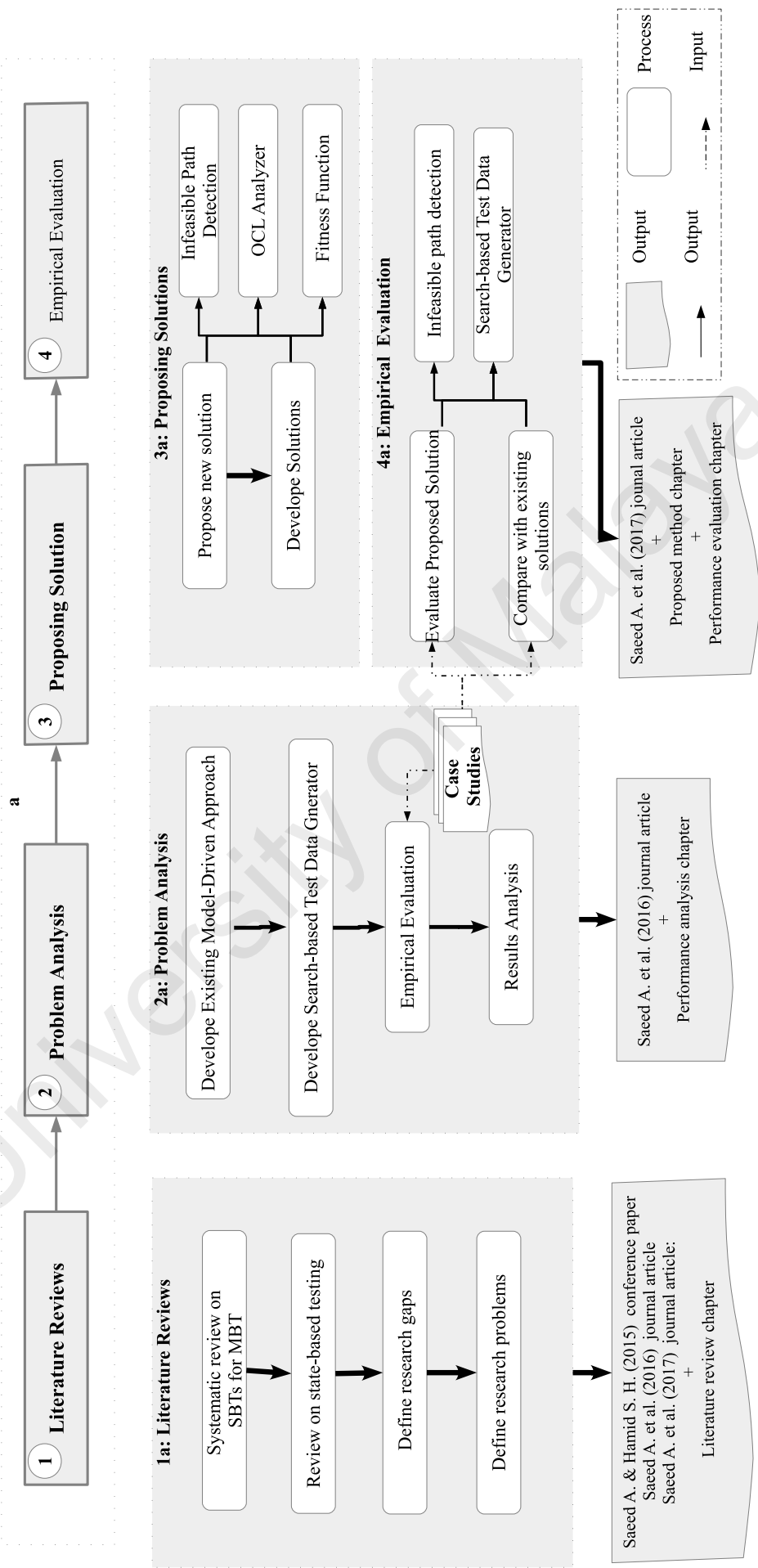


Figure 1.1: Research Methodology.

the cost and effectiveness of the existing recent methods of the identified research problems.

- To alleviate the identified problems, a proposed method was designed and implemented for generating feasible test cases from UML state machine models with OCL constraints. The proposed method consists of two steps: feasible abstract test case and test data generation. Abstract test case generation is a model-driven approach based on model-to-model and model-to-text transformations. First, transforming the input models into transition tree model based on All Round Trip coverage criterion (RTP) using model-to-model transformation. The output of this step is the generated test model (transition tree). Second, transforming the test model into executable test cases using model-to-text transformation, including traversing the test model (the transition tree) to get all paths in the transition tree, and check the feasibility of each path using the proposed static analysis infeasible path detection method. Each feasible path is transformed into one abstract test case. The output is a set of abstract test cases. In test data generation, the data is generated to satisfy whole constraints in each abstract test case. The OCL constraints in the test case were analyzed to get the dependency between the constraints clauses and the variables. All the clauses related to one variable were gathered as new constraint. A fitness function evolves itself based on error feedback was proposed to improve the performance of test data generator. The fitness function calculates the distance of the new constraint to lead SBTs to generate data that satisfy all the OCL constraints.
- The performance of the proposed method was evaluated via empirical case study analysis. Three industrial systems were used in this evaluation. Detection rate, generation time and success rate were opted as performance metrics in this evaluation.

Four SBTs (GA, EA,SA, and quantum GA (QGA)) were utilized. The standard setup of the experiments was applied. The results of performance evaluation were validated using comparison with the results of other recent methods. The statistical test was then conducted to show the significant performance of the proposed method.

1.6 Thesis Outline

The remainder of this thesis are organized as follows and represented in Figure 1.2.

- Chapter 2 reviews the research undertaken in the fields of SBTs for MBT and state-based testing. The chapter provides knowledge of MBT, search-based test data generation and state-based testing. This chapter also reviews applications of SBTs for MBT to identify and classify significant keys and presents the state-of-the art of the current research and the limitations. Furthermore, the aspects in the state-based testing were investigated to gain insight into the existing research space. The taxonomy of classifying these aspects of the state-based testing is also presented in this chapter. The existing studies in the state-based testing in the context of UML state machine models with OCL constraints and infeasible path detection are also reviewed. The existing research gaps are identified as future directions.
- Chapter 3 investigates and analyzes the performance of the model-driven approach with infeasible path detection and the SBTs in context of UML state machine with OCL constraints. Using empirical case study method, the effectiveness and cost of the existing methods are evaluated. The research problem is also verified and its significance is demonstrates.
- In chapter 4, a method is proposed to generate feasible executable test cases from UML state machine models with OCL constraints. The schematic presentation of the

method is demonstrated and the detailed of the method components are explained. Significance of the proposed method is also highlighted.

- Chapter 5 describes the followed performance evaluation methodology. The performance setup, the used case studies and the evaluation metrics are also described. Furthermore, this chapter presents the results of the performance evaluation and discuss the findings from two perspectives of effectiveness and cost. The results are compared and contrasted with the results of recent methods to validate the performance of the proposed method.
- Chapter 6 concludes the thesis by describing how the aim and objectives of the research are fulfilled. The main contributions are summarized and significance of the research and the proposed method in this thesis are highlighted. The publications are also listed including conference and journal articles that are produced from the research undertaken in this work. The limitation and future works are concluded at the end of this chapter.

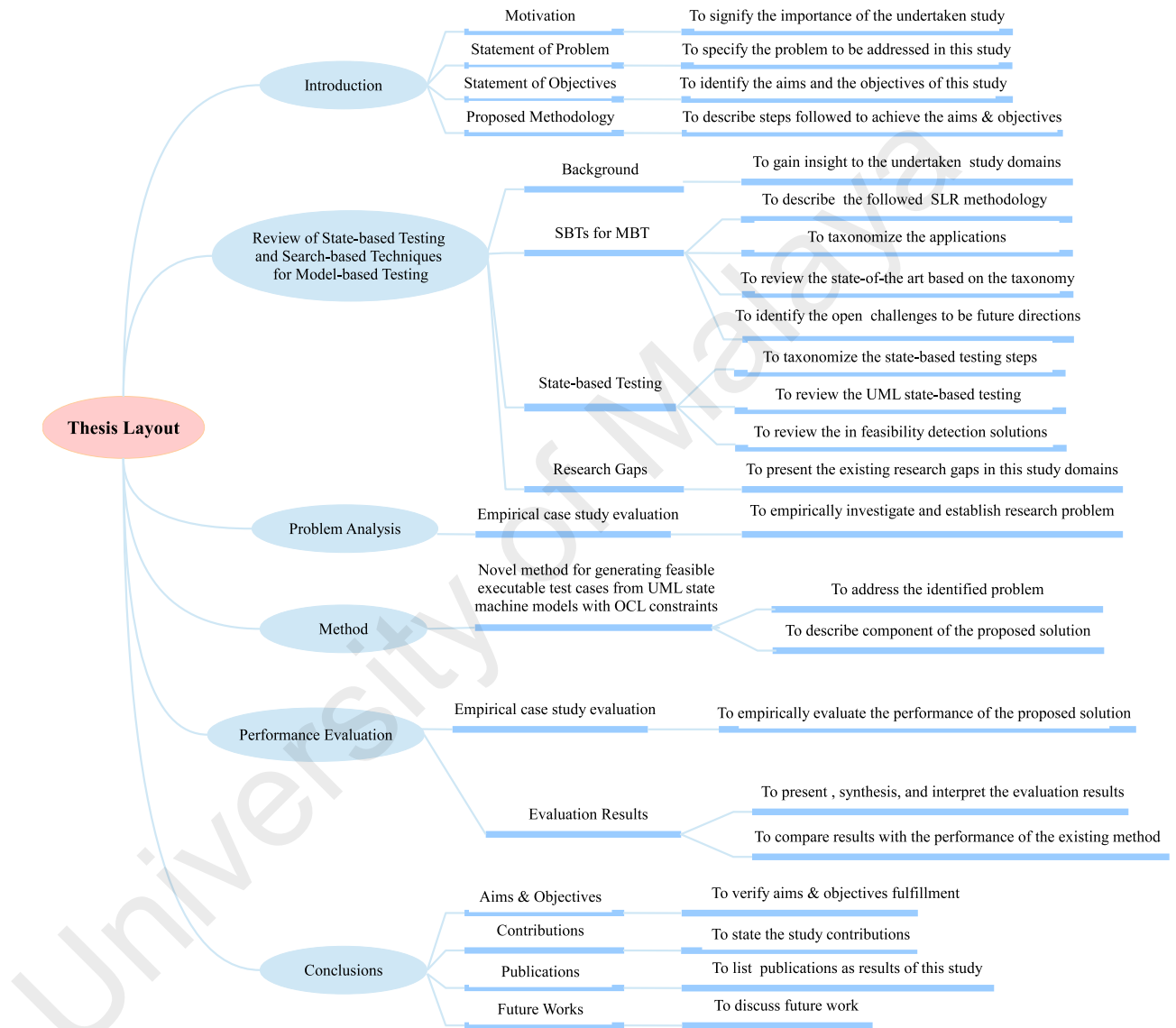


Figure 1.2: Schematic presentation of the thesis outline

CHAPTER 2: STATE-BASED TESTING AND SEARCH-BASED TECHNIQUES FOR MODEL-BASED TESTING: A REVIEW

This chapter reviews the SBTs for MBT and state-based testing domains to devise taxonomies. Systematic literature review was conducted from test case generation point of view to gain insight into how the SBTs were formulated for MBT, how solutions were proposed, how the solutions were assessed, and what is the testing purpose of conducting the research. Taxonomy of the applications of SBTs for MBT was presented and the detailed subclasses were identified. The overview of the current state-of-the-art of the applications of SBTs for MBT were critically reviewed. 72 varied applications were analyzed based on the taxonomy. A number of research gaps that can help to preside future directions were identified.

For state-based testing, the studies were reviewed from test case generation point of view and a taxonomy was proposed to classify the existing solutions in term of path and data generation. The related work of UML state-based testing with OCL constraints and EFSM-based infeasible path detection were comprehensively analyzed.

The remainder of this chapter is represented in Figure 2.1 and is organized as follows: Section 2.1 describes the background of the research areas related to this study. Section 2.2 presents the systematic review of SBTs for MBT, while section 2.3 reviews state-based testing. The finding research gaps are presented in section 2.4.

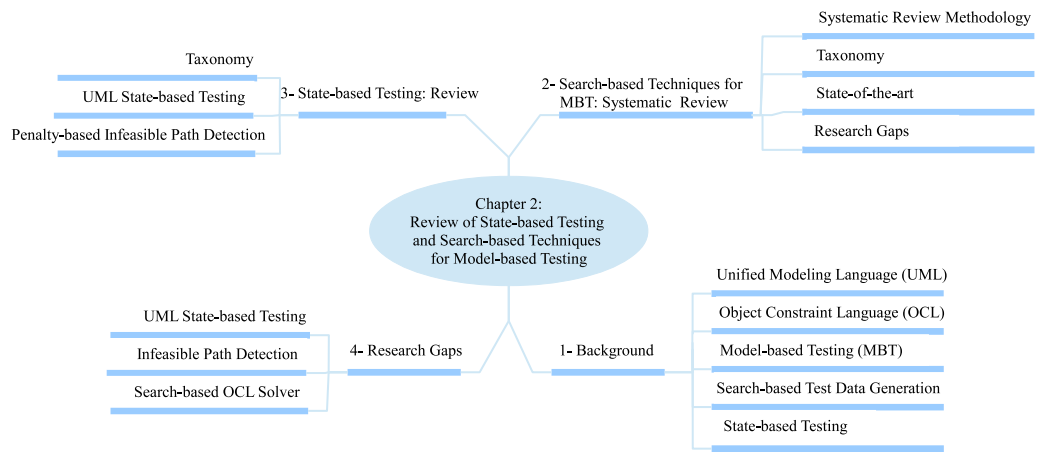


Figure 2.1: Semantic presentation of chapter 2 outline.

2.1 Background

This section presents the background on the modeling languages (UML and OCL), MBT, state-based testing and SBTs in the context of test case generation.

2.1.1 Unified Modeling Language (UML)

UML is a visual language that has been developed to support the design of complex object-oriented systems (Sarma & Mall, 2009). Recently it has become a defacto standard modeling language for industrial softwares because it provides a unified, precise, and consistent way to communicate information among different people involved in software development (Ali, 2011). UML models can be grouped into two classes: structural and behavioral models. The UML structural models are utilized to visual the static organization of the different items in the system, while behavioral models are used to model the dynamic perspectives of the system. The example of structural model is class model and of behavioral model is state machine model.

A state machine model consists of events, states and transitions. State refers to a model item may assume and followed by transition. The events can cause transitions to happen while the actions may happen in response to the events. States of an object are basically specified by the values that may assume for certain object attributes. Conceptually, an

object continues to be in a state, until an event causes it to transit to another state. A transition is a relationship between two states indicating a possible change from one state to another. The transition may have a guard to be satisfied to move into the next state.

The states in a state machine model are either simple or composite. A simple state does not have any sub-states, while a composite state consists of one or more regions. A region is a container for sub-states. The notion of a composite state makes a state machine model a hierarchical model. A composite state can either be sequential or concurrent. In a sequential type of composite state, the state is considered to be an exclusive- or of its sub-states. A composite state can be in any one of its sub-states, but not in more than one sub-state at any time. On the other hand, in a concurrent type, the state is determined by an and logic of its sub-states and the object is considered to be in all the concurrent states at the same time. Figure 2.2 presents an example for UML state machine models taken from Turn Indicator system (Peleska, Honisch, Lapschies, & Helge, 2011).

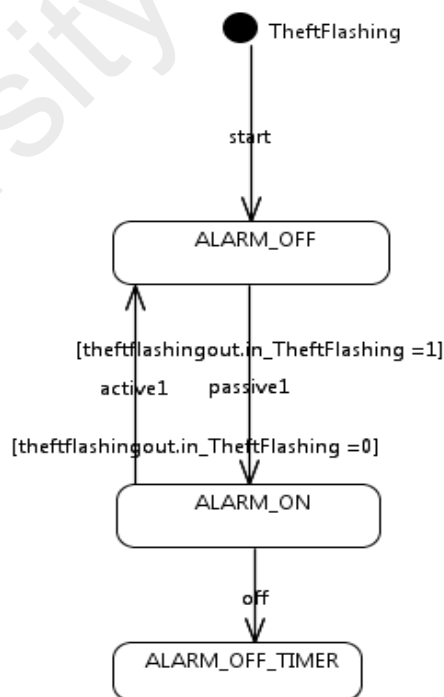


Figure 2.2: An Example of UML state machine model.

2.1.2 Object Constraint Language (OCL)

OCL is an extension language for writing constraints on UML models and it is widely accepted in the literature (Ali et al., 2013). OCL is based on first order logic and is at a higher expressive level than Boolean predicates written in programming languages such as C and Java. The constraints will be written at different levels of abstraction. It can be utilized to write class and state invariants, guards in state machines, constraints in sequence diagrams, and pre and post conditions of operations. The language is also utilized in writing constraints while defining UML profiles. Because of the ability of OCL to define constraints for different purposes through modeling, constraints play a significant role in the MBT. For example, in state-based testing, if the aim of a test case is to execute a guarded transition (where the guard is written in OCL based on input values of the trigger and/or state variables) in order to achieve full transition coverage, then it is essential to provide input values to the event that triggers the transition such that the values satisfy the guard. In testing, OCL evaluator is necessarily to be used to evaluate the generated data based on the constraints. OCL Evaluator checks whether a constraint on a UML model satisfies an instantiation of the model provided to it.

Two data types are supported by OCL: primitive data type which includes (Integer, Boolean, Real, and String) and complex data types which involves many types (enumerations, tuples, OCLState, Set, OrderedSet, Bag and Sequence). OCL provides the Undefined value if the data value is unknown. Each data type also has its own operations.

2.1.3 Model-based Testing (MBT)

MBT produces executable test cases by consistently analyzing the behavioral design models (abstract representation) of a software system by following a test strategy. Recently, MBT gained increasing interest in both industry and academia. This is visible from

several academic studies (Zhan & Clark, 2008; Iqbal et al., 2012b; Lindlar et al., 2010) and industrial projects (D-MINT, n.d.; Feldstein, 2005) on MBT. To fully automate MBT, three tasks are required: 1) constructing models from System under test (SUT) for testing, 2) deriving abstract test cases from the test model based on a test strategy, which is typically defined based on a test model and adequacy criteria to guide its traversal and 3) generating executable test cases by generating test data for executing abstract test cases as shown in Figure 2.3.

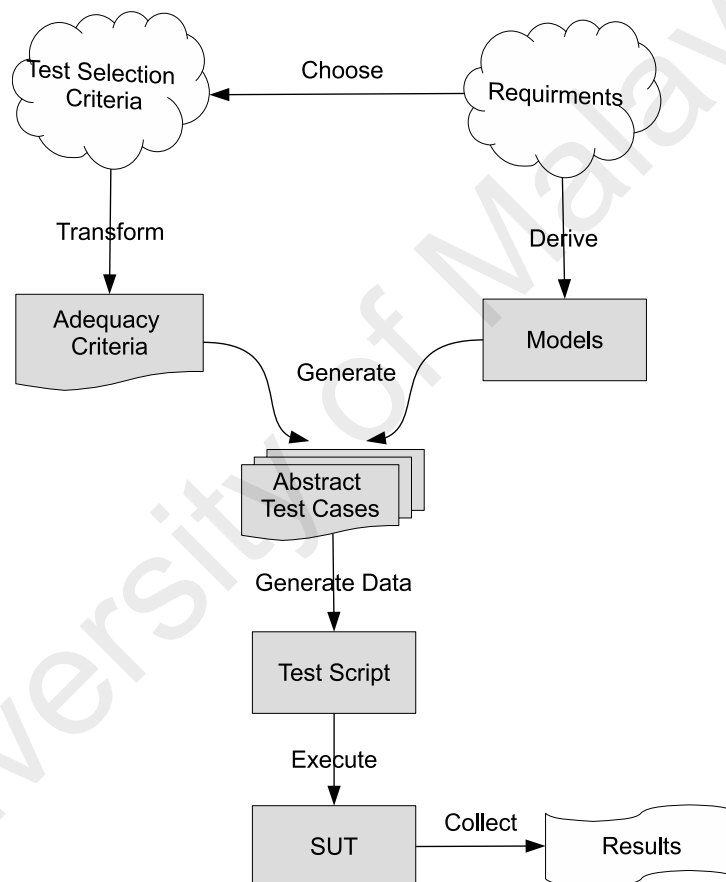


Figure 2.3: The process of model-based testing.

For first step, a model of the SUT is built from informal requirements or existing specification documents. This model is often called a test model, because the abstraction level and the focus of the model are directly linked with the testing objectives. In some cases, the test model could also be the design model of the SUT, but it is important to have some independence between the model used for test generation and any development

models, so that errors in the development model are not propagated into the generated tests (Pretschner & Philipps, 2005). For this reason, it is usual either to develop a test-specific model directly from the informal requirements, or to reuse just a few aspects of the development model as the basis for a test model, which is then validated against the information.

For second step, test adequacy criteria are chosen to define the test strategy so that it produces good test cases that fulfill the test objectives defined for the SUT. Defining a clear test strategy and test objectives for a system and associated development project contributes to produce a required test cases. Adequacy criteria can relate to a given functionality of the system (requirements-based test selection criteria), to the structure of the test model (state coverage, transition coverage, defuse dataflow coverage), to data coverage heuristics (pairwise, boundary value), to stochastic characterizations such as pure randomness or user profiles, to properties of the environment, and they can also relate to a well-defined set of faults. Once the model and the test strategy are defined, a set of abstract test cases is generated (Utting, Pretschner, & Legiard, 2011).

For the third step, the test data is generated to run the abstract test cases. For generating test data, several techniques have been proposed in the literature such as random (Anand et al., 2013), symbolic execution (Anand et al., 2013), model checking (Mohalik, Gadkari, Yeolekar, Shashidhar, & Ramesh, 2014), and SBTs (Ali et al., 2013). The latest endeavor is to deploy SBTs to MBT. It recently becomes a field of interest as reported in (Ali et al., 2013; Utting et al., 2011). The advantage is the capability of SBTs to find the optimal set of test cases in terms of maximum coverage criteria among all possible test cases at minimum cost. Specifically, the process of the test case generation can be formulated as an optimization process: The output of the test case generation could be hundreds of thousands of test cases for a certain SUT. From this context, there

is a need to select systematically those that adhere to particular coverage criteria at a reasonable cost and that are predicted to be fault detecting. Thus, the generation of test data can be reformulated as a search problem that aims to find the required or optimal set of test data from the space of the all possible test cases. Studies applied SBTs for MBT showed their significant performance compared to other techniques. For example, studies concluded that SBTs outperformed model checking for testing dynamic systems (Nilsson et al., 2006), and embedded real-time systems (Hänsel et al., 2011). Another study declared that the generating test cases using model checkers is more expensive than using heuristic techniques (Wenzel et al., 2008). After generating test cases and test data, test execution may be manual by a physical person or may be automated by a test execution environment that provides facilities to automatically execute the tests and record test verdicts.

2.1.4 Search-based Test Data Generation

Search-based software engineering (SBSE) solves various problems in the software engineering domain by reformulating the problems as search problems (Clarke et al., 2003). Search-based test data generation, is a part from SBSE, focuses on using SBTs for test data generation. SBTs are a group of generic algorithms that utilized heuristics to obtain optimal or near optimal solutions, and to solve the problems that have large search space at an affordable computational time cost. Specifically, an automatic test data generation process enable to be represented as a search problem that aims to find optimal test data from the space of all of the probable test data (Clarke et al., 2003). The possible generated test data can be massive, therefore, there is a need to select the test data that comply with specific coverage criteria and are expected to be fault revealing at a reasonable cost. SBTs have been applied for automatically generating test case based on a test objective (coverage criteria), which represented as fitness function. The fitness function is to guide the search for test data that maximize the achievement of the test objective. Therefore, different

fitness functions were proposed to capture different test objectives such as structural testing (Harman & McMinn, 2010; Fraser & Arcuri, 2013b), functional testing (Ali, Iqbal, Arcuri, & Briand, 2011), stress testing (Woehrle, 2012), and non-functional properties testing (White, Arcuri, & Clark, 2011).

Generally, SBTs for the test data generation have been widely studied in the literature (Ali, Briand, Hemmati, & Panesar-Walawege, 2010; McMinn, 2004) and recently they have been applied for MBT. Figure 2.4 shows the timeline of the development of SBTs for the test data generation and then for MBT as well. The first application of SBTs for software engineering problems was probably for the test data generation, achieved by Miller and Spooner (Miller & Spooner, 1976). They used numerical maximization as a technique for generating test data for floating point computations. After a decade, this research area appeared again by the work of Korel (Korel, 1990), who proposed a practical test data generation approach, Alternating Variable Method (AVM) based on Hill Climbing (HC). The first use of genetic algorithm (GA) for test data generation was in 1992 (Xanthakis et al., 1992). They instrumented the program to measure the coverage of some structural criterion (branch coverage). Other early application on this area was conducted by Schoenauer and Xanthakis (Schoenauer & Xanthakis, 1993) which focused on developing improved techniques for constraint handling in GA. Davies was also early pioneer of SBTs for structural testing. They applied GA for generating test data for an expert system (Davies, McMaster, & Stark, 1994). The first use of local search for the structural test case generation (Roger & Korel, 1995), and the first application of SBTs for structural testing from Z specification (Sthamer & Morgannwg, 1995) were conducted in 1995. The first use for testability transformation to improve the evolutionary testing was investigated by (Mark Harman, Hierons, Robert, Sthamer, & Harmen, 2002; Harman et al., 2004), wherein the flag variable problem was first formulated as a testability

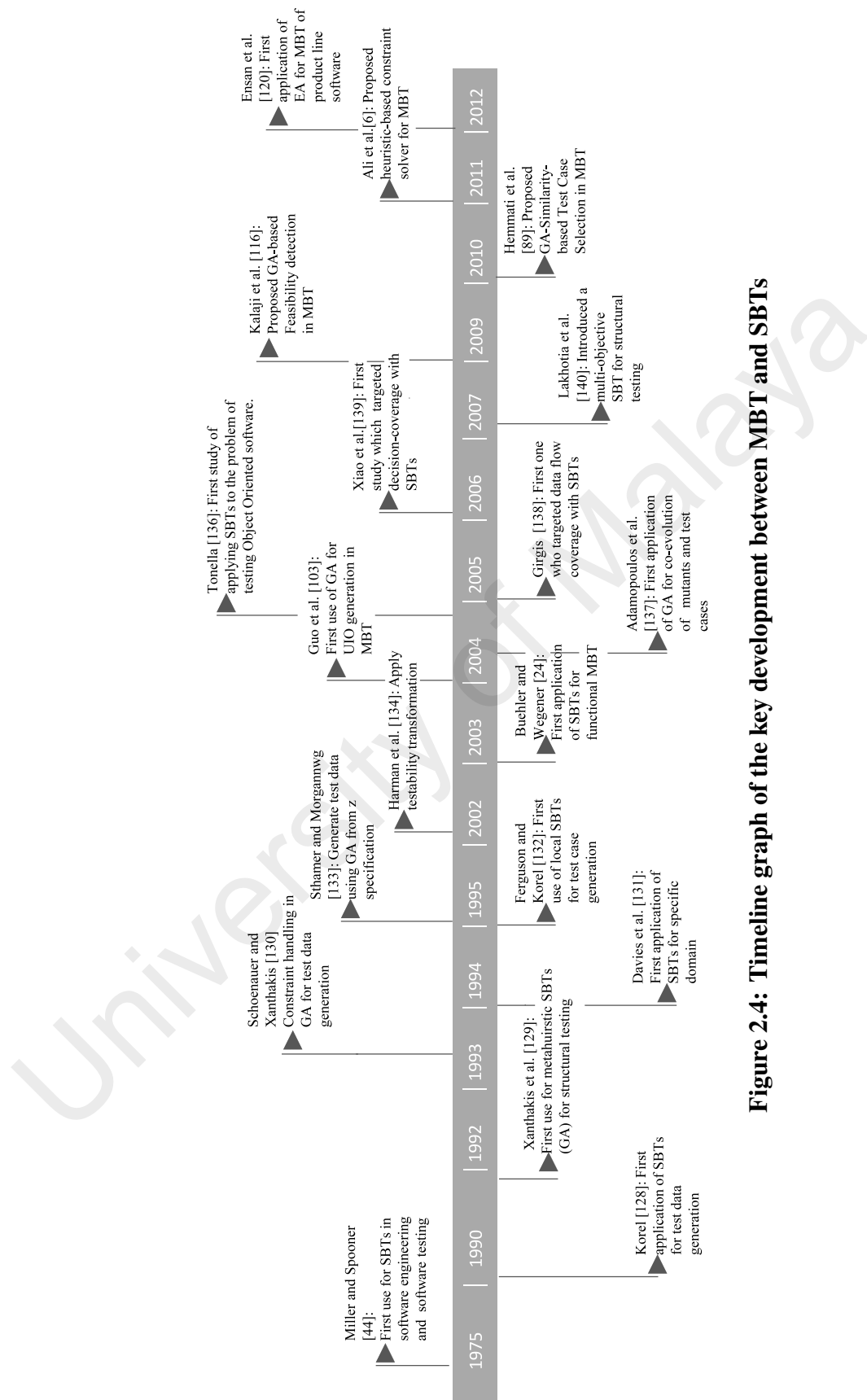


Figure 2.4: Timeline graph of the key development between MBT and SBTs

transformation problem . In 2004, Tonella (Tonella, 2004) was the first author who applied SBTs to the problem of testing object oriented software. In the same year, study (Adamopoulos, Harman, & Hierons, 2004) presented GA for co-evolution of mutants and test cases. Branch coverage is the common considered criteria in SBTs for structural testing, however study (Girgis, 2005) targeted data flow coverage and study (Xiao, El-Attar, Reformat, & Miller, 2006) targeted decision-coverage. Authors of (Harman, Lakhotia, & Mcminn, 2007) introduced a first multi-objective SBTs for structural test data generation that purposed for maximizing the coverage while also achieving other non-functional testing goals.

To the best of the author's knowledge, the first study used software models for the functional test data generation was done by Buehler and Wegener(Buehler & Wegener, 2003). Study (Guo, Hierons, Harman, & Derderian, 2004) was the early work of unique input/output (UIO) sequence generation using GA. Recent studies that combined concepts in MBT and SBTs are (A. S. Kalaji, Hierons, & Swift, 2009; A. S. Kalaji et al., 2011) proposed a recent feasibility detection approach based on GA, (Ali, Iqbal, et al., 2011; Ali et al., 2013) developed heuristic based constraint solver for test data generation, and (Hemmati, Arcuri, & Briand, 2010) proposed GA similarity based test case selection. For software product lines testing, work (Ensan, Bagheri, & Ga, 2012) applied SBTs. They proposed GAs to automatically generate test suites from features models of software product lines (SPL). Now, there is an increasing interest of using SBTs for SPL engineering as reported in the recent survey (Harman et al., 2014).

For more information to the reader, review studies have been conducted on SBST by: Anand et al. (2013) who conducted survey on test case generation approaches based on the expert knowledge of each specific approach, in which they presented the current state-of-the-art and future challenges of SBTs separately, as a part of the survey. Comprehensive

studies in (Ali, Briand, et al., 2010; McMinn, 2004) have reviewed generally SBTs for test case generation and found that most studies on SBST focused on code-based testing. Haraman et al. (2015) presented the achievement, challenges and open problems in SBST domain and concluded that there is a rapid growth of interest in SBST. The increasing interest on SBST for practitioners is due to the results are comparable to human competence (de Souza, Maia, de Freitas, & Coutinho, 2010), which are real-world results. For example, EvoSuite SBST tool (Fraser & Arcuri, 2011) has been successfully utilized to automatically generate test case for open source projects, randomly selected from open source repositories (Fraser & Arcuri, 2013a). SBST is presently mature enough that it has been used for industrial application rather than laboratory study, for example at Daimler (Buehler & Wegener, 2003; Vos et al., 2012), and Microsoft (Tillmann & Halleux, 2014). In addition to, the recent survey study (Harman, Jia, & Zhang, 2015), who is a pioneer ¹ in search-based testing domain, presented the trend of using SBTs for test data generation and concluded that the interest of this research field is continuously increased in recent years.

2.1.5 State-based Testing

One of the possible input models for MBT is state machine model beside class diagrams, activity models and use case models. State Machine models are widely utilized to model the behavior of the most critical and complex system components that exhibit state-driven behavior (L. C. Briand, Labiche, & Wang, 2004). Furthermore, the state machine models are widely utilized to model a great number of today's embedded systems (Ali et al., 2013; Vos et al., 2012; Lindlar et al., 2010). Object-oriented methodologies recommend modeling components with state models for the purpose of test automation (L. C. Briand,

¹He is the one who established the term search-based software engineering in 2001 and published a lot of research in search-based testing

Society, Penta, & Labiche, 2004). State-based testing aims to generate and execute test cases from state machine models. Two challenges in applying state-based testing which are generating abstract test cases and generating test data for executing the abstract test cases. For the first challenge of abstract test cases generation, four steps should be applied which are 1) construct test model 2) specify the target coverage criteria 3) generate all paths in the test model with respect to coverage criteria 4) convert feasible paths into concrete test cases. For the second challenge of test data generation, test data must be generated to fire guards associated with transitions, which typically require parameter values.

Several state-based tools and approaches have been proposed for UML state-machine models with OCL constraints (Weiß leder & Schlingloff, 2008; Lefticaru & Ipate, 2008b; Friske & Schlingloff, 2007; Sarma & Mall, 2009; L. Briand et al., 2010), but non of them can be configured and extended to various context. For instance, practical constraints are able to evolve, such as the test-script language a company works with. This drawback motivated researchers to proposed extensible and configurable model-driven approach (model transformations) to generate concrete test cases (Ali, Hemmati, et al., 2010) and test oracle (Lamancha et al., 2013) from UML models. The cost and effectiveness of this approach on industrial applications is provided in (Holt, Briand, & Torkar, 2014).

In the context of state-based testing using UML state-machine models, several test strategies (based on coverage criteria) are presented in the literature to achieve the first challenge, such as all round-trip paths (RTP) (Binder, 2000), all transitions (AT), all transitions pairs (ATP), M-length signature, and exhaustive coverage (Offutt, Liu, Abdurazik, & Ammann, 2003). Recent evaluation studies concluded that RTP is cost-effective and a compromise between the weak AT and the more expensive ATP criteria (Mouchawrab, Briand, Labiche, & Penta, 2011; Holt et al., 2014). In RTP strategy, a test tree also known as a transition tree (consisting of nodes and edges corresponding to states and transitions

in a state machine) is constructed by depth first traversal of the state machine. A node in the transition tree is a terminal node if the node already exists anywhere in the tree that has been constructed so far or is a final state in the state machine. Now, by traversing all paths in the transition tree, all round trip paths and all simple paths are covered (the paths in the state machine that begins with the initial state and ends with the final state). Another stopping criterion for the transition tree construction is proposed in (Mouchawrab et al., 2011), where a node is terminal if (i) it is a final state of the state machine or (ii) it is a node that already exists on the path that leads to the node. This stopping criterion makes the all round-trip strategy more demanding. This strategy has been experimentally evaluated to be more cost-effective than the all transitions and all transition pairs criteria (Mouchawrab et al., 2011).

For second challenge in the UML context, test data can be generated using random search from the possible set of values, or using more sophisticated techniques such as constraint solvers, or search-based techniques (SBTs) (for example using Genetic Algorithms for test data generation). The latest endeavor is to deploy SBTs to generate optimal test data that satisfies every transitions and constraints in the state model. This recently has become a field of interest as reported in (Ali et al., 2013; Utting et al., 2011). Studies on applying SBTs for UML state machine models show significant success rate performance as compared to other test data generation techniques. For example, studies concluded that SBTs outperform random search and constraint solver when testing complex embedded real-time systems (Ali et al., 2013, 2015). Another study declared that generating test data using model checker is more expensive than using heuristic techniques (Wenzel et al., 2008). Applying SBTs for generating test data for complex systems is non-trivial task because a proper fitness function should be carefully developed which takes time as reported in (Vos et al., 2012). Furthermore, constraints defined on UML state machines, such as

state invariants, guards, and pre/post conditions of triggers, should be evaluated during the execution of the generated test cases. As shown by many studies, using constraints or invariants is a very effective way to detect faults, e.g., state invariants serving as oracles in state-based testing (L. C. Briand, Society, et al., 2004; Holt et al., 2014).

2.2 Search-based Techniques for Model-based Testing: Systematic Review

This section presents systematic literature review (SLR) of the applications of SBTs for MBT².

2.2.1 Systematic Review Methodology

The methodology by which this SLR study was conducted is based on the guidelines proposed by Kitchenham in (B. Kitchenham, 2004; B. A. Kitchenham et al., 2002). The guidelines organize the steps of conducting a SLR into three stages, planning, conducting, and reporting, as shown in Figure 2.5. The aim of the first stage (planning) is to develop the review protocol, which encompasses: identifying research questions, search strategy, inclusion/exclusion criteria, data collection, and methods of synthesis. In the second stage (conducting), the focus is on executing the review protocol. The last stage (reporting) concerns how to elaborate on the final report.

2.2.1.1 Research Questions:

Defining the research questions is a potential step in defining the review protocol. Three questions were derived to embody this study sub-objectives and form the basis of this SLR:

RQ 1 : How can a basic classification framework be devised based on the current research on SBTs for MBT?

²This section is part of the published article (Saeed, Hamid, & Mustafa, 2016).

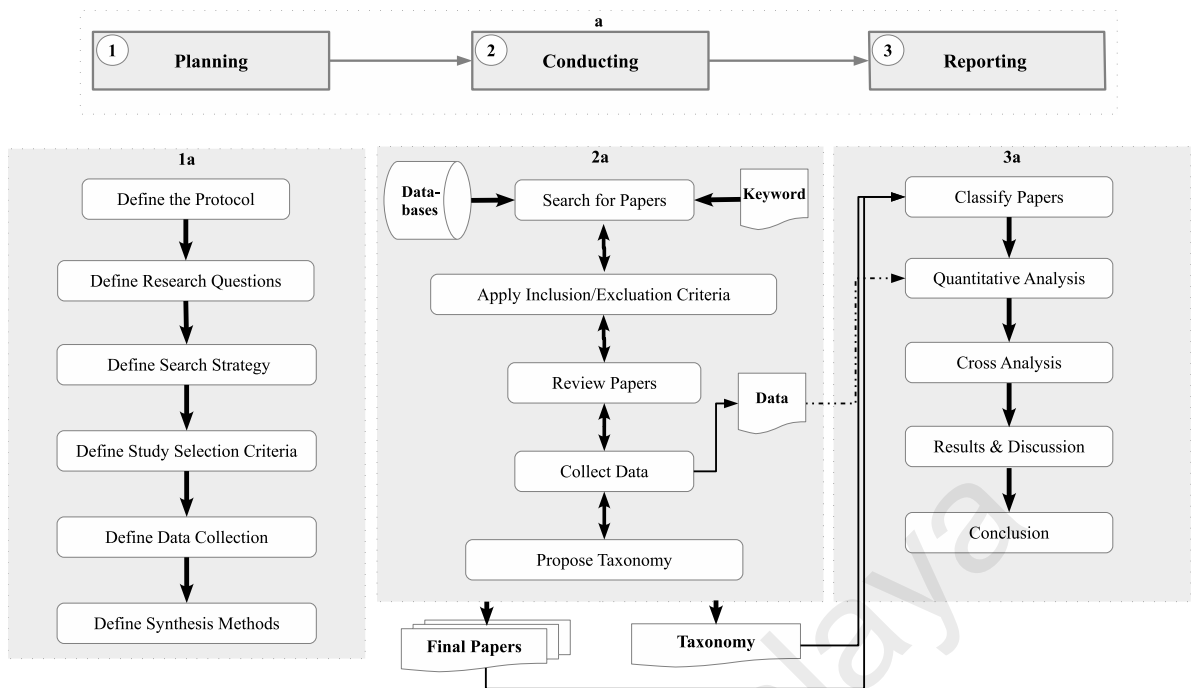


Figure 2.5: Systematic Literature Review Methodology

RQ 2 : What is the current state of SBTs for the MBT research area?

RQ 3 : What can be concluded from the current results that will help to preside future directions?

The aim of RQ 1 is to characterize and address the current research space. This question was addressed by deriving four fundamental questions portraying the devised approaches in this research area:

RQ 1.1 : What is the testing purpose?

RQ 1.2 : How are the problems in SBTs for MBT being addressed?

RQ 1.3 : What techniques have been proposed to solve the problems?

RQ 1.4 : How have these techniques been evaluated?

RQ 2 aims to give a comprehensive overview of the current state-of- the-art of this domain.

Three sub-questions are derived:

RQ 2.1 : What is the overview of the current research with respect to the taxonomy?

2.2.1.2 Search process

The search process of this SLR study started with selecting the data sources and the search queries. The selection of suitable source databases and keywords plays a vital role in the completeness of the collected data. The data sources used for finding the published papers in journals and conference proceedings between 2001 to 2013 are IEEE Xplore, Springer, Google Scholar, ACM, ScienceDirect, and Wiley Interscience. These well-known data sources are the widely accepted literature search engines and databases. 2001 was selected as the starting year for the search. To the best of the author's knowledge, the first study on SBST (Miller & Spooner, 1976) is further probably to be the first study on SBSE. Although, the original stretching of SBSE is back to the 1970s, it was formally become as a field of research in its own right in 2001 (Harman & Jones, 2001). In addition to, this field only accomplished further widespread acceptance and uptake several years later (Freitas & Souza, 2011; Harman, 2007; Harman et al., 2015; Ahmed, Zamli, & Lim, 2012). The last year is 2013 because the search process and data collection were in 2014. The systematic method was followed which presented in (Ali, Briand, et al., 2010) to derive the search queries. This systematic method includes the following: specify the main search keywords based on the study questions, find substitute keywords and synonyms for the main keywords, and generate a search query by combining the main keywords with the Boolean AND operator or by combining the alternative keywords and synonyms with the Boolean OR operator.

Based on the objective of this study, the major keywords are model-based testing, test case generation and search-based techniques. Alternative keywords were found: for *model-based testing* (i.e., functional testing, and black-box testing, state-based testing, and specification-based testing), for *test case generation* (i.e., test data/scenario/ sequence

/generation/optimization), and for search-based techniques (i.e., metaheuristic techniques, optimization algorithms, and AI techniques). *Search-based testing and evolutionary testing* are two well-known keywords in the scope of this study. These keywords were also combined, refined, and extended through the search. After the keywords were finalized, the keywords were run through the search engines of the databases. The final list of the used keywords is presented in the appendix Table A.1.

2.2.1.3 Study Selection

The study selection step ensures the completeness of the selected papers. After all the search keywords were run through the selected data sources, 546 papers were obtained in total from 2001 to 2013, as illustrated in Figure 2.6. According to this study focus and reducing selection bias, suitable inclusion/exclusion criteria were subjectively investigated that guaranteed selecting credible and relevant papers. Two steps were applied in order to apply inclusion/exclusion criteria and to opt for the most relevant papers to the research questions. In the first step, the papers have been divided into two sets. Each set has been reviewed by one researcher, including the titles, abstracts, and keywords of the obtained papers according to the following inclusion criterion:

- Study addresses the use of SBTs for MBT.

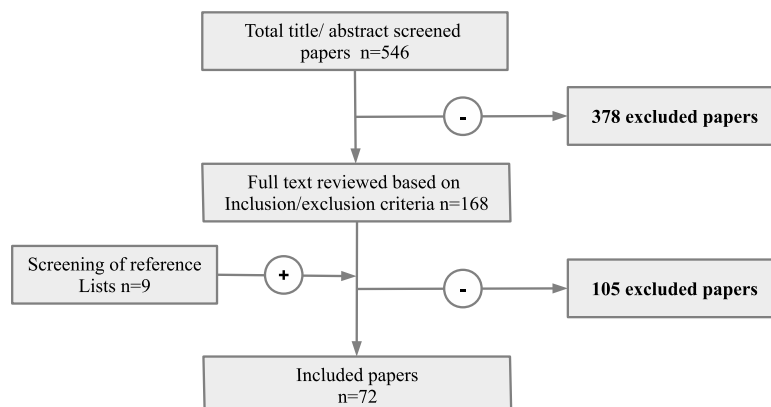


Figure 2.6: The output of search process and study selection steps

After the first reviewing, if the researcher was confident about a paper, the paper was then included for the second selection step. After applying inclusion criterion and removing the duplicate papers, only 168 papers were included in the second selection step, as shown in Figure 2.6. The search was further extended to the reference lists in the 168 papers to obtain more relevant papers. Specifically, the title part was checked in the references and abstracts. The paper was then added in the list, if the title and abstract discussed both MBT and SBTs or their alternatives, and it was not in the papers list as well. 9 new papers were found from checking the references. In the second step, two researchers reviewed the entire text of the papers separately to check the content with respect to the exclusion criteria. The results of the two researchers were compared. In cases there is a difference on the two results, then the paper was discussed with the all researchers and the agreement of including or excluding was made through majority consent.

In this study, the goal is to analyze experimental papers only because this study proposes a taxonomy which is devised based on technical details. Therefore, the theoretical work cannot be analyzed based on this taxonomy. The proposed taxonomy is a basic framework to compare and analyze the experimental papers. However, to include theoretical works, while this work is based on technical work will create conflicts of parameters and understanding to the readers. The following are the exclusion criteria:

- Paper is not peer reviewed, including tutorials, editorial material, posters, technical reports.
- Paper does not present technical details and results; the goal is to analyze experimental papers.
- Paper presents only conceptual framework or concepts; the goal is to analyze experimental papers.

- Paper presents an empirical study that presented in other paper.

Figure 2.6 also shows that 72 papers were obtained after applying the inclusion/exclusion criteria on the 177 papers. Specifically, after applying the first three exclusion criteria, 74 papers were obtained. However, applying the last exclusion criterion led to excluding two conference papers (Kruse, Wegener, & Wappler, 2009; Zhan & Clark, 2004) because its empirical study was presented in other journal articles (Kruse, Wegener, & Wappler, 2010; Zhan & Clark, 2008). Each of the selected 72 papers was comprehensively analyzed, independently, based on the taxonomy attributes (discussed more in section 2.2.2) in order to be confident that the paper was a worthy contribution to this SLR study. Subsequently, these 72 papers were used as the basis of this review.

2.2.1.4 Data extraction

The information extracted from each study must meet the eligibility criteria and address the answers of the research questions. Therefore, two sets of information were gathered from each paper. The first set encompassed general information about the paper, such as the title, authors names, year, source, document type, and a summary of the study. The second set involved the information regarding the taxonomy sub-categories of each main category and the research questions as the following:

- For problem sub-categories: contribution type either tool, method or framework, model type, application domain, modeling language, dimensionality, adequacy criteria, constraints, quality attribute, and test level.
- For solution sub-categories: fitness function, type of SBTs, constraint handling, landscape visualization, seeding type, stopping criteria, model transformation, tuning and other optimization process improvement.

- For evaluation sub-categories: cost measurement, effectiveness measurement, dataset scale, comparison baseline, and overall evaluation.
- For purpose sub-categories: test purposes.

A sample of papers was selected and read by all researchers and the relevant data extracted to assess the consistency of the extracted data. The researchers discussed the extracted data to ensure a common understanding of all data items being extracted. The final set of the selected paper were assigned to one researcher to read and extract the data. The extracted data was reviewed by the other researchers in order to mitigate data collection errors. All ambiguities were clarified by discussion among the researchers. The data is available in <https://github.com/aneesaaljibli2013/SBTs-for-MBT>.

2.2.1.5 Data Analysis

The collected data is tabulated based on each main categories and their sub-categories in the taxonomy. Quantitative analysis (meta-analysis) was applied for each main category (purpose, problem, solution and evaluation). Deeper analysis was then performed on the sub-categories in order to well understand the domain. The extended analysis of the data across both problem and solution categories was established by conducting cross analysis. This collected data was represented and analyzed in Microsoft Excel.

2.2.2 Taxonomy

This section to address RQ 1. Defining a highly detailed taxonomy for SLR influences the depth of knowledge collected for each paper and the quality of the SLR as well. In this study, the taxonomy schema was iteratively updated during data extraction and eligibility evaluation by adding new categories and merging or splitting existing categories. First, the first level of the taxonomy hierarchy was derived which focuses on the four fundamental

questions (RQ1 sub-questions), which characterizing the existing applications (refer to section 2.2.1.1). Each of these sub-questions was discussed in detail to get deeply knowledge from each research, and determine the implied taxonomy scheme. Figure 2.7 depicts the devised taxonomy and shows that the classification taxonomy was based on prominent common characteristics of the existing works involving four intrinsic categories: problems, solutions, evaluation, and purpose. In particular, sub-categories were derived to deeply address all the information related to each main category. Each sub-category has a number of possible features that characterize the application of SBTs for MBT. The characteristics of each category will be described below.

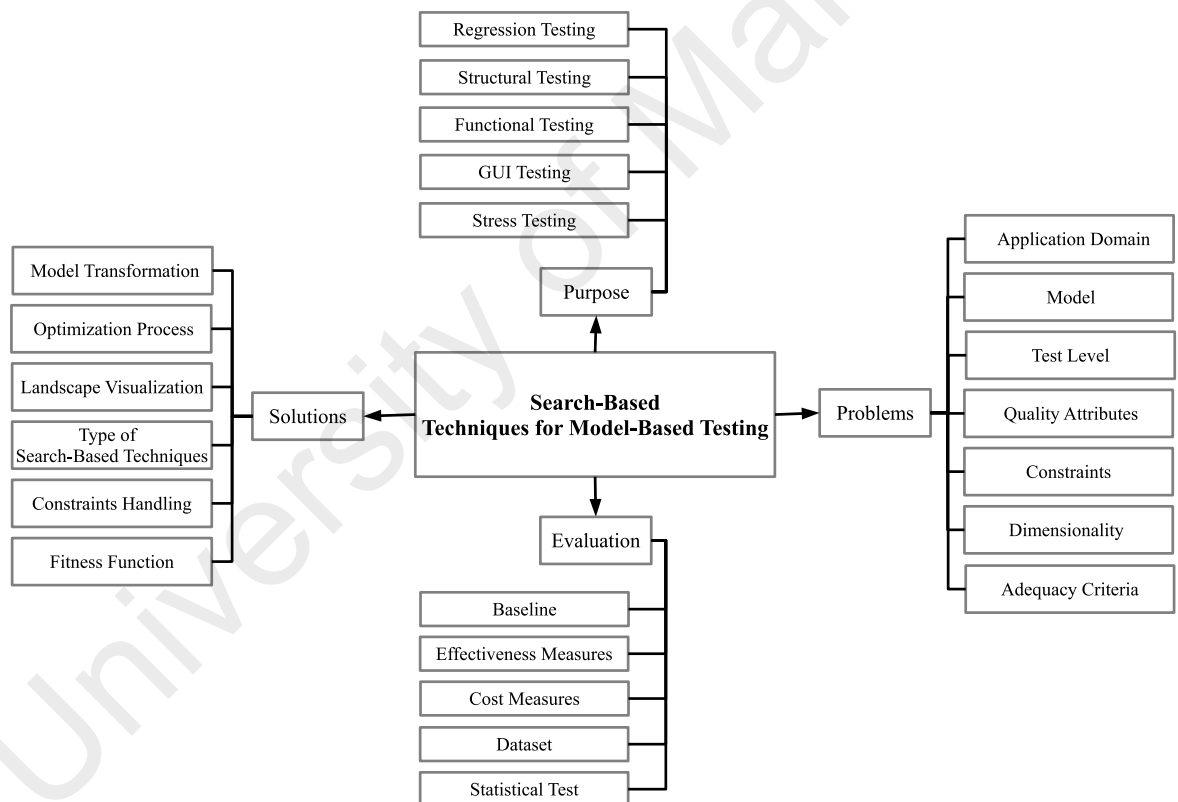


Figure 2.7: Taxonomy of classifying the applications of SBTs for MBT

2.2.2.1 Purpose

The purpose of the testing process plays a role in generating the targeted test cases and opting for the adequate technique. The generated test cases must fulfill the testing

objective. From the selected papers in this study, "structural testing", "functional testing", "GUI testing", "stress testing", and "regression testing" are considered as the *purpose* (objective). The purpose does not include the full range of purposes because it is only based on the selected papers. For structural testing, the target of generating test cases is to cover the internal structure of the system source code or model. For functional testing, the generated test cases are constructed from the specification of system behavior to detect the faults in the system functionality. The specification of the system behavior is usually represented as models. GUI testing refers to generate test cases consisting of sequences of GUI input events. GUI testing focuses on detecting faults related to the GUI and its code. Study (Ammann & Offutt, 2008) reported that GUI testing falls into categories: usability testing and functional testing. In addition to, study (Banerjee, Nguyen, Garousi, & Memon, 2013) classified GUI testing into functional GUI testing and non-functional GUI testing. Therefore GUI was considered as separate purpose from functional testing. Stress testing concerns on performance degradation and consequent system failures which commonly emerge in stressed conditions. For instance, stressed conditions can be realized when various users are concurrently accessing a system or when large amounts of data are being transferred through a network link (Garousi, 2010). Regression testing is a standard part of the maintenance phase in the development. It aims to test the software after changes and to ensure the updated software still processes the functionality it had before the refinement (Ammann & Offutt, 2008). Purpose category combines aspects that are different and not necessarily mutually-exclusive.

2.2.2.2 Problems

Generally, the existing works in optimizing test case generation from models has attempted to attain particular optimization objectives in a certain context. For instance, an optimization goal is to maximize the fault detection and state coverage of system models with

given functional constraints at reasonable cost. An example context is to consider UML state machines models of embedded systems for functional testing. Therefore, the context of the problem are determined by three sub-categories: 1) *application domain* (the type of targeted systems), 2) *models* (the type of targeted models and modeling language), and 3) *test level* (the desired level in the system). The four sub-categories concerning the optimization goal are 4) *dimensionality* (the dimensionality of the optimization problem), 5) *adequacy criteria* (the optimization target), 6) *quality attributes* (the quality attributes that are regarded in achieving testing), and 7) *constraints* (the targeted system properties).

First, the *application domain* refers to the targeted domains in applying SBTs for MBT. In this study, these are categorized into two domains, the general domain and specific domain. Specific domain includes studies which proposed SBTs for specific type of software system models. Specific domain sub-categories comprise "network-based application", "web application", "embedded system", "agent-based system", "database application", "symbiotic systems", and "product-lines software". The general domain sub-category refers to when the specific type of the targeted applications domain is not specified in the paper.

Second, category *models* imputes the used input artifacts. In MBT, the used artifacts are models that originate from the targeted application. The used models were drilled in detail. Therefore, they are distinguished into two prominent aspects of the *model* sub-category; the first aspect considers the targeted *model types*, which describe the desired behavior of SUT or system environment. Both of them involve various types of modeling diagrams, such as class diagrams, sequence diagrams, deployment diagrams, and interaction overview diagrams. The second aspect is the utilized *modeling language* for promoting the design models. For example: UML, and Simulink/MATLAB.

Third, the nature of the testing step embraces that different levels of testing accompany

software development activities, where in, the prominent engaging *testing levels* are "system testing", "unit testing", and "integration testing". These three levels are considered as the testing levels on (Utting et al., 2011). System testing assesses whether the software system meets its specification with respect to the architecture design phase of development or not. Unit testing evaluates the units produced by the implementation phase. Integration testing is designed to evaluate the accurate communication between modules in a given subsystem produced by subsystem design phase. The main resemblance between these levels is that they correspond to parts of the targeted software that must be tested in a specific order. Alternately, the intrinsic distinction between them is the information of each level deduced from the distinct development activity.

Fourth, the *dimensionality* sub-category reflects if the SBTs address either single-objective or multi-objective. The dimensionality of the SBTs is called a "single-objective" optimization if it covers one optimization goal; in contrast, it is called a "multi-objective" optimization if it contains conflicting optimization goals either aggregated into a single fitness function or more. From this study context, an example of single-objective is maximizing state coverage, whereas an example of multi-objective is maximizing state coverage while reducing the number of generated test cases.

Fifth, the majority of the optimization goals in this study focus are related to *adequacy criteria* sub-category, such as maximizing transition coverage. These adequacy criteria intend to discover either implicit faults or explicit faults in the test models. The sub-category of *adequacy criteria* has five possible values. "Model-flow" coverage criteria are related to the structure of the model, such as transition-based coverage, state-based coverage, path-based coverage, and scenario-coverage criteria. "Data" coverage criteria involve how to choose a set of test values from a large data space, including one-value, all-values, boundary-values, N-way values, and pairwise values. "Requirement-based"

coverage criteria depend on the association links between software requirements and model elements, for example: creating tests that cover the model elements (e.g., transitions) related to particular requirements. "Script-flow" coverage criteria refer to the scripts or constraints related to the function behavior, such as an interface (function), branch, condition, statement, atomic-condition, and modified-condition/decision. "Fault-based" criteria are the most related to the test goal, which is to detect faults. The most well-known fault-based criterion is mutation analysis.

Sixth, the goal of conducting software testing in the software development cycle is to ensure the quality of the software system. The subjective nature of the software quality concept leads to defining the quality via system attributes (called quality attributes). To categorize the *quality attributes*, accepted definitions and taxonomies (AviZienis, Laprie, Randell, & Landwehr, 2004; Aleti, Buhnova, Grunske, Koziolok, & Meedeniya, 2013) were followed. Based on the selected papers in this study, five attributes are considered as the following *quality attributes*: "performance", "cost", "robustness", "reliability", and "safety". For example, testing the performance and robustness of the software is a setting with two quality attributes (performance, and robustness) to be optimized.

Finally, software quality attributes and constraints are closed concepts in optimizing MBT. Therefore, They are distinguished by considering *constraints* as a separate category. The *constraint* sub-category refers to the investigated constraints on the software system properties that must be considered, embracing "structural", "functional", "timing", "sources", "integrity", and "performance". The system models are described both the system implementation and the execution environment supporting the system. There are constraint bounded execution environment and constraint for the system implementation. For example, a state machine model of real time embedded software system contains constraints on some system functions (guards), and an environmental model of real time

embedded software systems contains time constraint on a large numbers of interacting components.

An example to distinguish between quality attribute and constraint categories, increasing the reliability and involving the functional and timing constrains is a setting with one quality attribute(reliability) and two constraints (functional and timing).

To see a full structural view of the aforementioned *problem* category, refer to Figure 2.8.

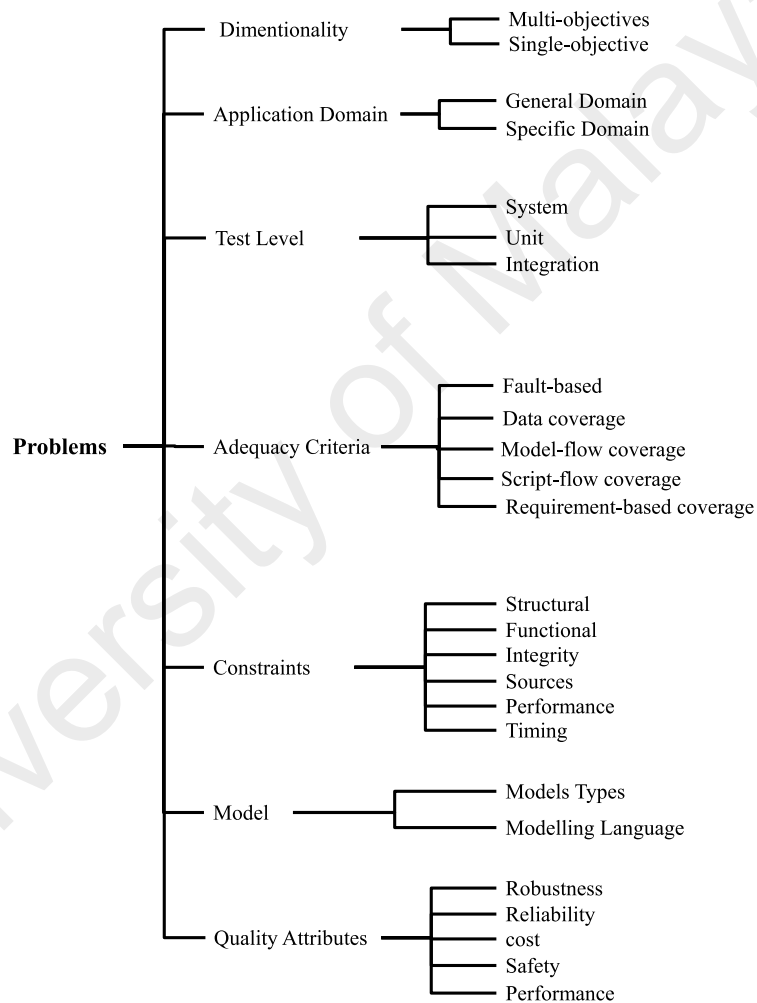


Figure 2.8: Full structural view of problem category

2.2.2.3 Solutions

The existing studies investigated SBTs for MBT to achieve the optimization goal (generating optimal test cases with maximizing the adequacy criteria). These techniques provide

the solutions in terms of how the optimization steps are applied. In this section, the *solution* category distinguishes these optimization steps into six sub-categories: 1) The model transformation refers to the process input that depicts the model to optimize. 2) The fitness function describes the used objective functions of the optimization process that covers the quality evaluation procedures. 3) The SBTs depicts the type of the used SBTs. 4) The constraint handling refers to the method that is used to handle system constraints. 5) The landscape visualization refers to how the fitness landscape is visualized. 6) The optimization process comprises the improvements of the optimization steps to enhance the SBTs performance.

First, MBT takes some representation of the software models as an input. A model is most often a combination of text and drawings, and is a specification and description of the modeled system and its environment. Model transformations are used to add detail to and refine model representations. In MBT community, the researcher applied model transformation to transform a model from one form to another such as from a model to another model (e.g UML-state machines to UML flattened state machines), or a model to text (e.g model to executable code). To employ SBTs for MBT, the input models should be also transformed to be suitable for use by SBTs. The transformed model is maybe the model or the text describing models. The transformed model has to be encoded into the optimization search space that illustrates the fitness function and the adequacy criteria. Therefore, the applications of SBTs for MBT are distinguished according to the transformation approaches applied to the model in the reviewed papers. The *model transformation* sub-category has these possible values: "model to model" , "model to text", and "not transformation".

Second, developing a proper *fitness function* is the most significant step to evaluate the quality of a candidate solution in the search space based on the adequacy criteria; thus,

it needs proper considerations to be taken for developing suitable fitness functions. Based on the reviewed papers, the sub-category *fitness function* is defined that differentiates the used objective functions into two types: a "general" fitness function adopted for code coverage based on branch distance and/or approach level and a "specific" fitness function proposed based on the features of the targeted applications and adequacy criteria. This differentiation between these proposed fitness functions in spited subcategories may help the reader on selecting which type will be more appropriate for a certain type of software models, software testing purpose as well.

Third, SBTs are used when the near optimal solutions are enough for solving the search problem. The *type of search-based techniques* sub-category refers to the used SBTs in order to generate the test data from the model. Deciding which SBTs will be chosen is a substantial tradeoff because of the complexity of transferring MBT as a search problem. SBTs are divided into three types based on the search strategy. First, a "global" SBTs use a global search to find the global optimal solution effectively and to overcome the problem of becoming stuck in local optima. The main strength of global SBTs is their exploration capability. They generally take a global picture of the search space in the beginning of the search, and typically, they iteratively applied simple and problem-dependent operations to derive various new solutions successively focusing the search on auspicious regions of the search space. Global SBTs are good in identifying promising areas of the search space (Blum, Puchinger, Raidl, & Roli, 2011). Global SBTs are widely applied in the test case generation field, such as evolutionary algorithms (EAs), genetic algorithms (GAs), particle swarm optimization (PSO) and genetic symbiosis algorithms (GSAs). Second, a "local" SBTs use a local search to obtain the optimal solution efficiently. Local SBTs start from a current solution and try to update it by modifying some of its components (Bianchi, Dorigo, Gambardella, & Gutjahr, 2008). The advantage of local

SBTs is their rather fast intensification capability, that is, the capability of fast finding better solutions in the locality of specified starting solutions (Blum et al., 2011). Hill climbing (HC), and simulating annealing (SA) are the most common examples of local SBTs. Local SBTs are more efficient and simpler (e.g HC), but they are more likely to become stuck in local optima (Blum et al., 2011). The optimization goal is achieved better via a global SBTs than a local SBTs, but the computational cost of local SBTs is more reasonable than the global search (Harman & McMinn, 2010). Third, "hybrid" techniques are created by combining either two SBTs, such as GA with HC, or one SBTs with other non-optimization techniques, such as GA with model checking.

Fourth, the *constraint handling* sub-category refers to the utilized approaches to handle constraints. Studies (Coello, 2002; Aleti et al., 2013) surveyed four possible approaches for constraint handling. The "penalty" approach adds a penalty parameter to the fitness function in order to address the constraint optimization problem as a series of un-constraint problems. This penalty parameter reflects the constraints violations. The "prohibit" approach refers to dismissing the solutions that violate a constraint. The "repair" approach aims to fix any constraint violations before the evaluation step. The "general" approach indicates the other constraint handling approaches.

Fifth, visualization is one of the most basic tools for studying search spaces (Kim & Moon, 2002). Harman (Harman, 2007) mentioned that landscape visualization is an open challenge in SBSE. Fitness landscape visualization may provide valuable insights on which search technique will work best for the problem in hand (Varshney & Mehrotra, 2013). In this study, the focus is on whether researchers paid attention to landscape visualization or not. *Landscape visualization* points to the used approach to visualize search space (fitness landscape). The researchers in search-based optimization community commonly applied the search space visualization, in which a measure of landscape height is the

fitness evaluation values and each candidate in the search space takes up a location on the horizontal axis. Two *landscape visualization* approaches are used: a 2D plane or 3D plane. 3D plane (three-dimensional space) is used when there are only two decision variables (for both x,y axes) and the height on the z axis depicts the goal of the optimization (e.g minimize the fitness function). 2D plane (two-dimensional space) is applied when the search problem defines by n decision variables. All n decision variables from a search space are possibly mapped onto a flat plane (x,y axes). In this study, the possible values of the *landscape visualization* are as "2D visualization" , "3D visualization", and "no visualization".

Finally, any optimization technique includes various steps to achieve the optimization goal. The experts in the SBTs community proposed some keys that contribute to enhance the performance of SBTs. The *optimization process* can be enhanced by improving their parameters, configuration or process. The typical main improvements that affect these steps are the chromosome encoding, first population, alternative population, and stopping criteria. Based on the reviewed papers, only three common improvements were found that are applied on: "stopping criteria" (when the search stops), "seeding" (generating the first population) and "tuning" (adjusting the algorithms parameter settings). The first two are configuration choices, while parameter tuning is a process that is often applied outside of the algorithm itself. Furthermore, the widely used technique in this research domain is EA, and specific improvements were found for its algorithm steps of "mutation", and "crossover" (recombining operators to generate an alternative population). Also choosing different "chromosome length" has been suggested to improve EA. The first three improvements are common among all SBTs therefore each of them were considered as separate value of optimization process. The last three are related to only EA algorithms thus these three last improvements were distinguished as the "others" value.

Figure 2.9 illustrates a full structural view of the aforementioned *solution* category.

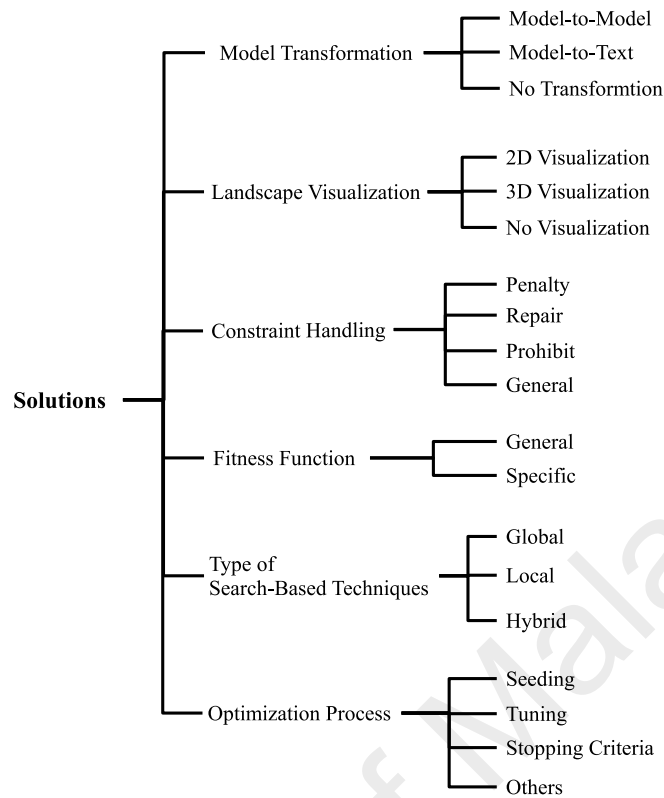


Figure 2.9: Full structural view of solution category

2.2.2.4 Evaluation

The evaluation step in the SBTs research domain aims to assess the effectiveness, cost, and accuracy of the proposed technique. In this study context, this evaluation measures the strength of the proposed technique in generating test cases that attain specific adequacy criteria at a credible cost. The evaluation approaches, explored in the reviewed papers, comprise affirmation with simple examples, benchmarks (experiments), or industrial case studies. Moreover, the empirical studies presented in the reviewed papers were converged.

Empirical studies specifically present the existence of persuasive evidence in order to extend the effectiveness, cost, and scalability of the SBTs for MBT. Empirical studies are ordinarily conducted by experiments, or case studies. According to the focus in this comprehensive SLR study, any type of empirical evaluation, applied in the field of

SBTs for MBT, was considered. In addition to, the empirical evaluation items were reckoned from guidelines (Ali, Briand, et al., 2010) which related to the evaluation part and applied in the existing work, summarized as: select targeted SUT from application domain called (*dataset* sub-category), the used techniques for comparison (*baseline* sub-category), how the effectiveness is evaluated (*effectiveness measures* sub-category), how the cost is measured (*cost measures* sub-category), and the used statistical analysis for comparing cost-effectiveness measures (*statistical test* sub-category).

The used dataset is one of the main elements that affects the SBTs performance evaluation. Therefore, the papers were further classified based on the type of the used dataset. The possible types of the used dataset are one simple example, an industrial case study, or a dataset from a common open source (mentioned in the taxonomy as an experiment). More complexity on the systems used as datasets indicates the quality of the evaluation results and reflects the real performance of the technique.

The comparison baseline is a substantial factor that affects the evaluation of the effectiveness of the SBTs performance. The comparison baseline includes: literature comparison, internal comparison, external comparison, and not presented. Literature comparison refers to the techniques presented in other works from the literature. Internal comparison uses the same technique with specific improvements as the comparison baseline. External comparison includes other SBTs or non-SBTs as baselines. The possible techniques for external comparison are local SBTs as HC, global SBTs as GA, and non-SBTs as random search. Applying external comparison baselines will prove that the problem is either simple enough to solve by simple techniques, such as local search and non-SBTs, or complex enough that it needs more complicated techniques, such as global search. The papers that did not present any type of comparison baseline are referred to as "not-presented." These papers actually presented the performance of the SBTs for MBT

without discussing the justification of why the SBTs were necessary to address the MBT problem and how much better the SBTs were compared to other existing techniques in solving the MBT problem.

The main goal of conducting the empirical study is to assess the effectiveness of the techniques. In this study context, four essential effectiveness measures are considered: coverage-based, fault-based, time-based, and fitness values. The coverage-based includes control-flow (such as state, path, transition, condition, and condition decision) and data (such as all value, one value, and boundary value) sub-measures. The fault-based sub-category compresses mutation analysis (where the mutation score and number of killed mutants are the measures), and the fault detection rate. Time-based sub-category is related to the execution time of the test cases. The fitness value sub-category does not fit into the aforementioned categories but is still related to the quality (referred to as the fitness value) of the generated test cases (solutions). These fitness value measure was applied when the exact optimal targeted solution is not found and the closed solution to the targeted solution is found.

The cost measures refer to the cost of the test case generation process or the cost of executing the generated test cases. According to this study analysis, five sub-measures were found in the literature, including the number of iterations, the number of fitness evaluations, the number of individuals, the time of the test case generation, and the number of the generated test suite (test suite size).

Statistical tests refers to the papers reported the results of statistical tests comparing SBTs and the baselines and showed the statistical significance of the differences between the techniques performance. Statistical tests are often categorized as parametric and nonparametric (Wohlin et al., 2012). Parametric tests are suitable when the sample follows a specific distribution, such as t-test. Alternatively, nonparametric statistical tests

are utilized when no appropriate assumptions can be made about the sample distributions e.g., Mann Whitney U-test.

Figure 2.10 illustrates a full structural view of the aforementioned *evaluation* category.

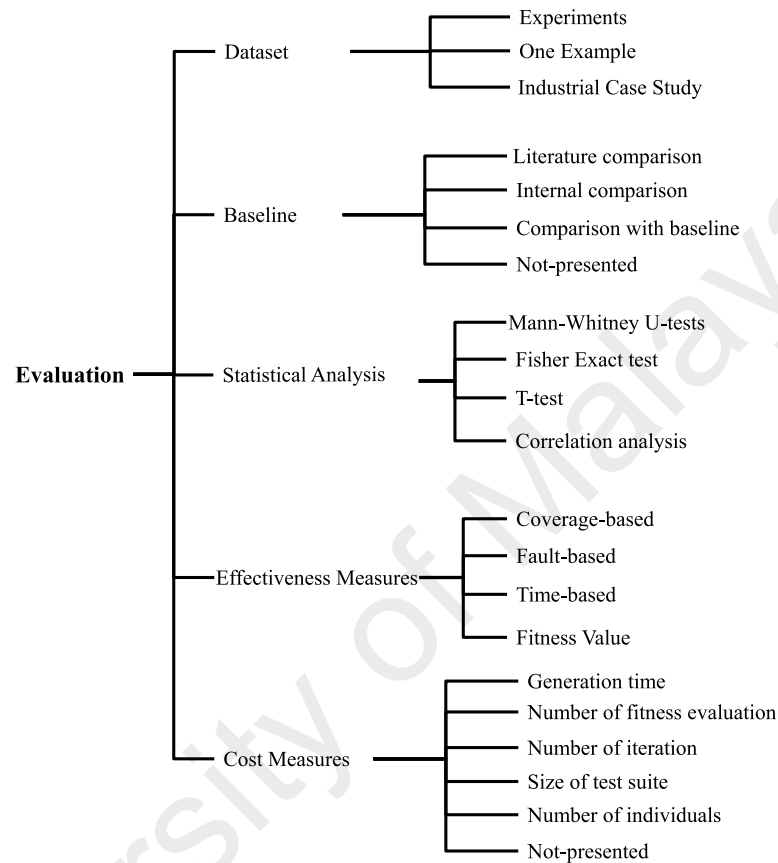


Figure 2.10: Full structural view of evaluation category

2.2.3 State-of-the-Art

This section to address RQ 2. The overview of the current state-of-the-art of the applications of SBTs for MBT guides the reader to the suitable approaches that are of attention and are provided in Tables A.2, and A.3 (appendix A). These tables summarize the approaches of the different sub-categories and the references of the papers that present these approaches. The general goal of applying SBTs for MBT is to conduct a certain testing purpose; thus, these tables are structured based on the *purpose* category to make it more readable. The structure of the tables is as follows: each column provides the purpose of

testing with the numbers of papers in parentheses, each row in the tables refers to the results of a certain sub-category in the taxonomy, and each cell lists the papers that outlined the sub-categories values grouped by the purpose of testing. Three papers (Tasharofi, Ansari, & Sirjani, 2006; Baresel, Pohlheim, & Sadeghipour, 2003; Farooq & This, 2011) appear in two columns because they address two purposes of testing (functional and structural testing). Two papers of regression testing have been also focused on other purposes, including one GUI testing (Alsmadi, Alkhateeb, Maghayreh, Samarah, & Doush, 2010), and one structural testing (Shelburg, Kessentini, & Tauritz, 2013). Furthermore, papers outline multiple values for some sub-categories, and thus they appear many times in the same cell, such as: (A. S. Kalaji et al., 2011; Yano, Martins, & de Sousa, 2011) provided two fitness functions, (one general and one specific), and (N. Li, Li, Offutt, & Va, 2012) proposed a new method based on SBTs. Consequently, the total number in the tables is more than the number of reviewed papers (72).

The state-of-the-art of this study's focus is discussed according to the proposed taxonomy. The taxonomy structure has four main categories, which are investigated to answer the four basic questions and to analyze intensively the selected papers. The observation concluded from each sub-category of the main categories in the taxonomy will be presented in this section.

2.2.3.1 Purpose Category

Figure 2.11 illustrates that both structural and functional testing purposes gained interest more than the others. From the selected papers, 4 percent of the papers focused on both structural and functional testing (Farooq & This, 2011; Tasharofi et al., 2006; Baresel et al., 2003). SBTs were proposed (Shelburg et al., 2013) for both regression testing and structural testing, while study (Alsmadi et al., 2010) presented SBTs for GUI and regression testing. Functional testing garnered more attention than structural testing; this

is because the main focus of MBT is on the functional behavior of the system (Utting & Legeard, 2010). Utting and Legeard (Utting & Legeard, 2010) define MBT as the automation of the design of black-box tests (functional testing). Other testing purposes need further research to realize the effectiveness of MBT for them.

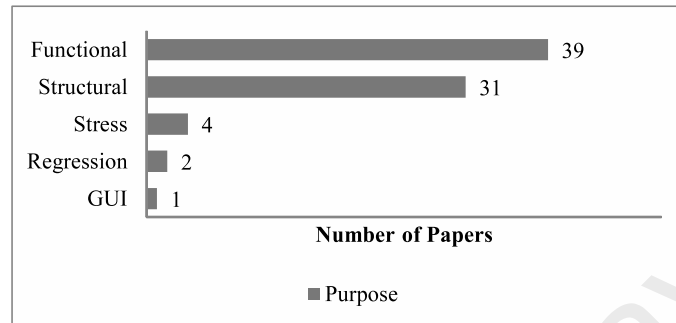


Figure 2.11: Distribution of papers based on the purpose category.

2.2.3.2 Problem Category

This section agitates the summary of the quantitative analysis of the results of each sub-category in the problem category presented in Table A.2 (appendix A).

The classification of the papers is discussed here based on the application domain sub-category. Figure 2.12 shows that the majority of the applications of SBTs for MBT concerned on specific application domains, of which, 64 percent of the overall reviewed papers are focused on the specific domains. Specifically, a significant number of the papers have converged on embedded systems (66 percent of the papers focused on specific domains), while a small number of the papers applied for other specific domain (34 percent). The remaining general domain refers to the used dataset, which has not been clearly assigned from a specific domain, comprising 36 percent of all this literature. The main focus in MBT is for specific domain applications is due to each specific domain has different characteristics and needs specific SBTs to cover all its characteristics. One cause of this significant attention on embedded systems is that a great number of today's products are based on the deployment of embedded system (Kruse et al., 2009) and

embedded systems are a major challenge to test engineers (Hänsel et al., 2011).

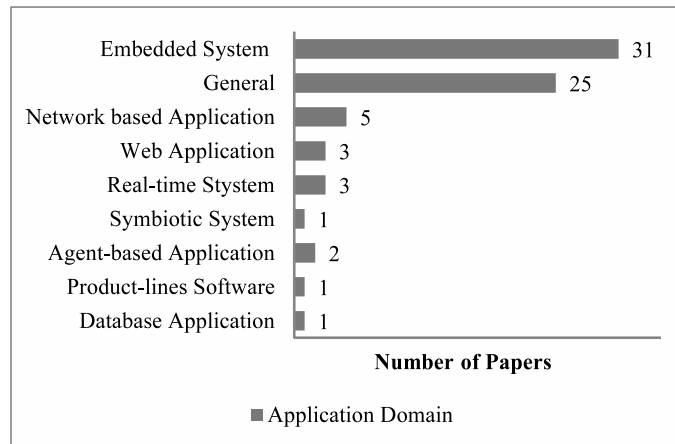


Figure 2.12: Distribution of papers based on the application domain sub-category.

With respect to the model category, two aspects are discussed here. First, Figure 2.13 illustrates that a large number of the papers (48 percent) focused on state-based models (state machines, EFSM, Simulink and FSM). From the literature, state-based models are widely utilized to model embedded and control systems (Ali et al., 2013; Vos et al., 2012; Lindlar et al., 2010), and this result aligns with the results in Figure 2.12. Second, Figure 2.14 shows that 42 percent of all the reviewed papers have not obviously presented the utilized modeling language, while 58 percent (42 papers) listed the used modeling language. 74 percent of them used both UML and MATLAB as a modeling language. UML and MATLAB are well-established modeling languages and widely used in industry. Object Constraint Language (OCL) is used for writing scripts with UML in many studies (Ali et al., 2013; Ali, Iqbal, et al., 2011; Arcuri, Iqbal, & Briand, 2010; Iqbal et al., 2012b).

It can be observed from Figure 2.15 that the number of the papers concentrated on system level (76 percent) is significantly more than the other levels. Only 24 percent of all the papers studied the other three levels, including 20 percent for unit, and 4 percent for integration. Studies of the regression testing focused on both test case generation and

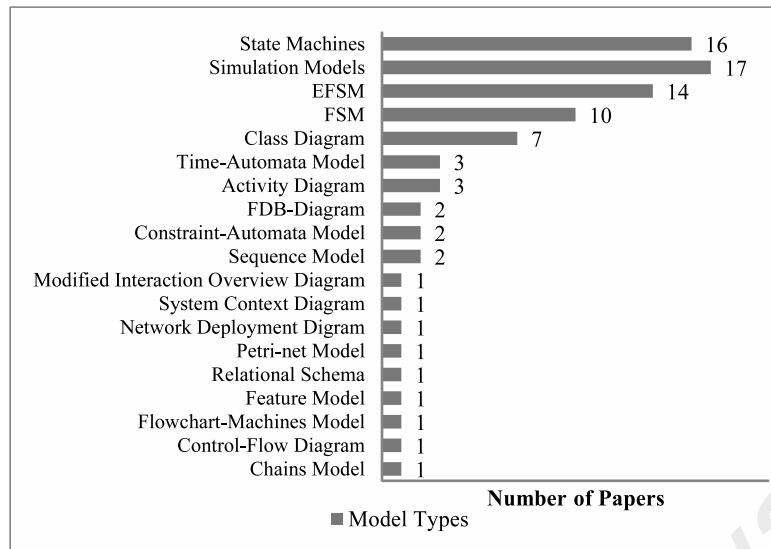


Figure 2.13: Distribution of papers based on the model type sub-category.

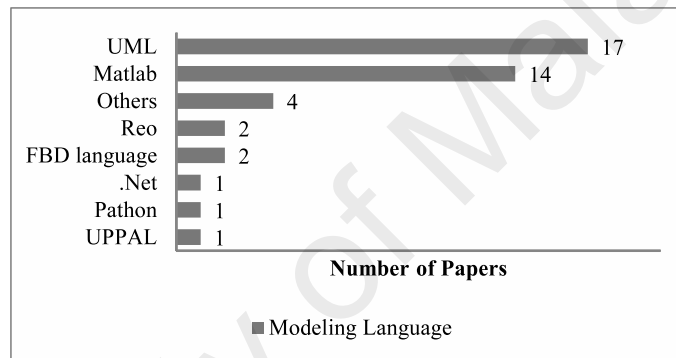


Figure 2.14: Distribution of papers based on the modeling language sub-category.

selection (Shelburg et al., 2013; Alsmadi et al., 2010).

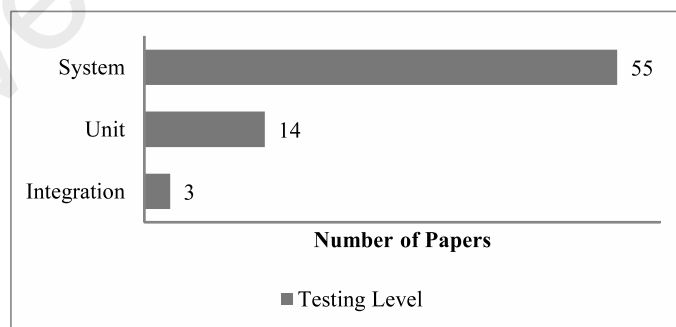


Figure 2.15: Distribution of papers based on the testing level sub-category.

The papers are differently distributed between single-objective (86 percent), and multi-objective (14 percent), as illustrated in Figure 2.16. This indicates to the less attention in handling more than one objective in SBTs for MBT, which contributes to

amelioration of the optimization performance. In this study context, the single-objective studies focused on maximizing adequacy criteria such as (K. Derderian, Hierons, & Harman, 2006; Wegener & Kruse, 2009; Windisch & Al Moubayed, 2009; Baresel et al., 2003; Núñez et al., 2012; Buehler & Wegener, 2003), while the multi-objective focused on maximizing the adequacy criteria with either detecting feasible paths (A. S. Kalaji et al., 2011), reducing number of test cases (Shelburg et al., 2013; Alsmadi et al., 2010; Farooq & Lam, 2009; Farooq & This, 2011), or reducing length of test case (Yano, Martins, & de Sousa, 2011; Yano, Martins, & Sousa, 2011; Yano, Martins, & de Sousa, 2010). Compared to other applications in the optimization field, the recent solicitude on enhancing SBTs is based on the multi-objective perspective.

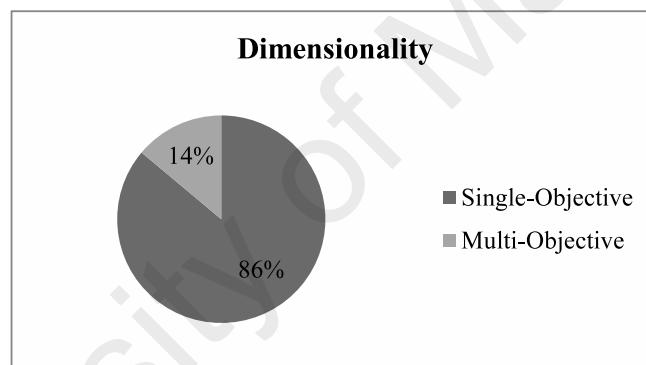


Figure 2.16: Distribution of papers based on the dimensionality sub-category.

The majority of the approaches considered the fault-based adequacy criteria (38 percent), followed by the model-flow criteria (36 percent), as shown in Figure 2.17. This high percentage of both adequacy criteria may be because the main focus is to detect the faults in the system function or the model structure. Moreover, fewer papers (26 percent) considered the constraints or requirements during the test case generation. The cause of this few research studies is the test case generation from models with constraints and requirement is non-trivial task (Ali et al., 2013; Ali, Iqbal, et al., 2011), which need to be satisfied with a view to the system to be endorsed.

Figure 2.18 presents that the performance attribute (67 percent) is the main focus

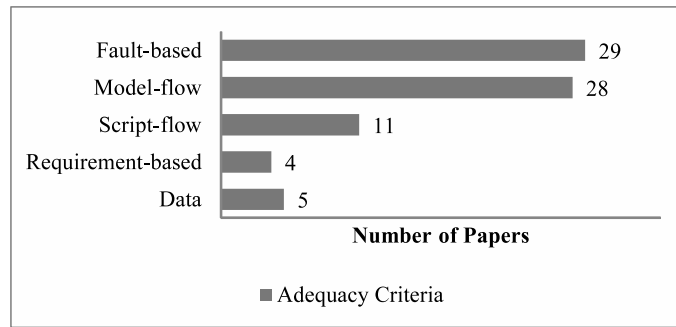


Figure 2.17: Distribution of papers based on the adequacy criteria sub-category.

of the researchers in this SLR. In addition, specific terms of the good performance have been exhibited, comprising safety (8 percent), robustness (6 percent), and reliability (7 percent). However, fewer papers considered the cost of the generated test data (12 percent). One reason for the significant attention paid to the performance is that the discrimination between the correct and the incorrect performance of the software system implementations is one of the intents of the software testing phase (Núñez et al., 2012). Although the cost of generating test cases is an important factor in reducing the cost of the entire software testing and development as well, it needs more research to focus on.

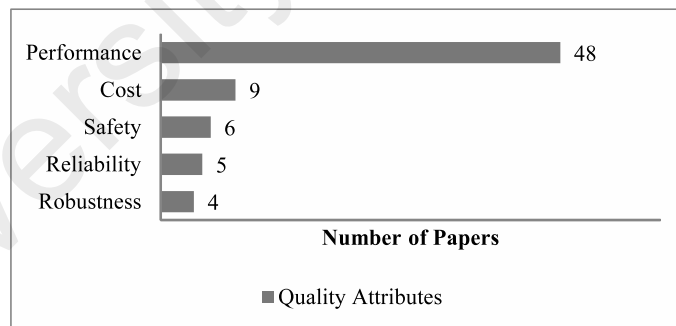


Figure 2.18: Distribution of papers based on the quality attributes sub-category.

Figure 2.19 shows that a considered number of papers applied SBTs for MBT without regarding the system constraints (33 percent). This is because constraint satisfaction in SBTs for MBT is a critical part that adds more intricacy to the search space (Wilmes & Windisch, 2010). The risk of ignoring constraints during the test case generation is that the entire quality of the software testing process will be negatively affected because the

targeted test cases will not generated. From the papers that considered the constraints, the main attention was on the constraints related to the system functions (56 percent). This is an interesting result because the main focus of the test case generation is to evaluate the performance of the system functions. The other crucial constraints are related to the system model structure (13 percent) and the timing (17 percent). Other popular constraints gained less attention. There are various papers that focused on more than one constraint. Papers (Arcuri et al., 2010; Iqbal et al., 2012b) studied both timing and functional constraints. Additionally, study (Tasharofi et al., 2006) addressed constraints related to both function and structural of the system models. Both time and resources constraints were handled in (Nilsson et al., 2006).

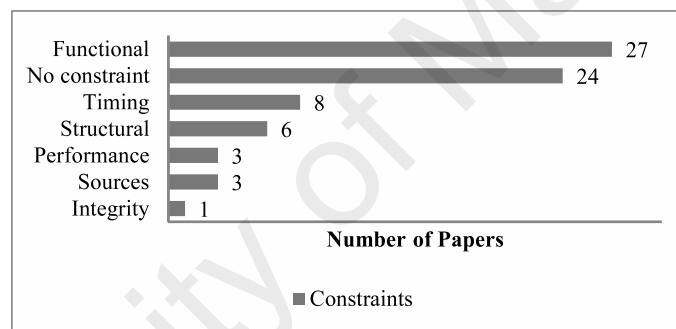


Figure 2.19: Distribution of papers based on the constraints sub-category.

2.2.3.3 Solutions Category

In this section, the main observation is characterized for each sub-category in the solution category from Table A.3(appendix A).

Figure 2.20 shows that a considerable number of the papers did not apply any transformation on the input model (29 percent), while the approaches (71 percent) that applied a model transformation directly either transformed the model to another model (29 percent) or transformed the model to text form (42 percent). Examples of the studies that used a model-to-model transformation are sequence model to control flow diagrams (Garousi, 2010), UML 1.4 class model to UML 2.0 class diagram (Shelburg et al., 2013),

stateflow model to instrument model (Windisch, 2010), UML activity model to colored petri net model (Farooq & This, 2011), UML-state machines model to activity diagram (Mani, Garousi, & Far, 2010), UML-state machines to EFSM (Shirole, 2011), time-automata into FDB (Enoiu, Doganay, Bohlin, Sundmark, & Pettersson, 2013), UML-state machines to UML flattened state machines, (Lindlar & Windisch, 2010; Iqbal et al., 2012a, 2012b; Ali et al., 2013; Ali, Briand, Arcuri, & Walawege, 2011; Ali, Iqbal, et al., 2011; Nilsson et al., 2006; Hemmati et al., 2010). Model-to-text transformations include model to disjunctive normal form (Lefticaru & Ipate, 2008a; Hänsel et al., 2011; Lefticaru & Ipate, 2007; Wilmes & Windisch, 2010; Lefticaru & Ipate, 2008b; Ipate & Lefticaru, 2008), EFSm into hardware description language (Guglielmo, Guglielmo, Fummi, & Pravadelli, 2011), model to auto code (Kruse et al., 2010; Zhan & Clark, 2008; Arcuri et al., 2010; Vos et al., 2012; Yano et al., 2010; Lindlar et al., 2010; Enoiu et al., 2013; Yano, Martins, & de Sousa, 2011; Doganay, Bohlin, & Sellin, 2013; Oh, Harman, & Yoo, 2011; Yano, Martins, & Sousa, 2011; Wong, Ooi, Hau, Marsono, & Shaikh-husin, 2013), model to BCD form (Shirole & Kumar, 2010), model to alphabet form (J. Li, Bao, Zhao, Ma, & Dong, 2009), and model to binary decision diagram (Ensan et al., 2012). The studies (Ali, Briand, et al., 2011; Ali, Iqbal, et al., 2011; Ali et al., 2013; Enoiu et al., 2013; Hemmati et al., 2010; Iqbal et al., 2012b) applied both transformation types.

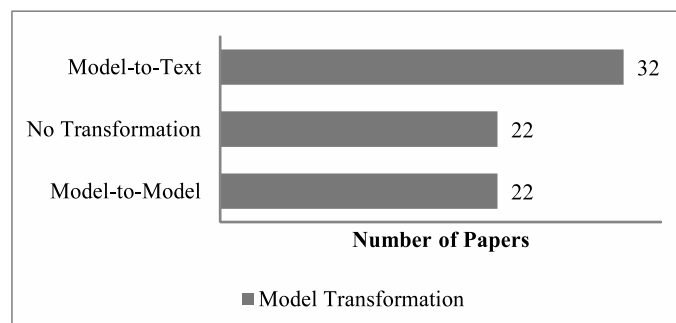


Figure 2.20: Distribution of papers based on the model transformation sub-category.

It has been noticed from Figure 2.21 that most approaches (93 percent) clearly

described the proposed fitness function, in which 69 percent of them proposed a specific fitness function, while the remaining papers applied the general function. Most of the reviewed papers were conducted for specific applications (refer to Figure 2.12); as a result, the fitness function was specifically formulated to be suitable for the specific characteristics of these applications. This is a justification of why most of the existing work is based on a particular fitness function. Lefticaru and Ipate (Lefticaru & Ipate, 2008a) compared both of them and concluded that a general function produces results comparable to those produced by specific fitness functions. Both studies (A. S. Kalaji et al., 2011; Yano, Martins, & de Sousa, 2011) recommended that using the two types of fitness functions in the same study promotes successful results.

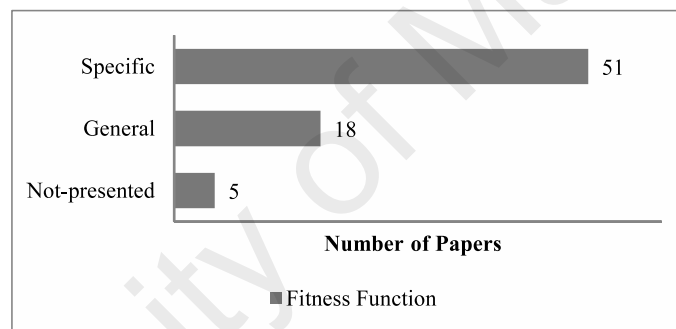


Figure 2.21: Distribution of papers based on the fitness function sub-category.

Figure 2.22 shows that a high number of the papers used global search techniques, comprising 77 percent, compared with local search techniques, which gained only 17 percent. The reason of this high percentage is that EAs, GAs, and their extensions are regarded as robust algorithms (Sharma, Sabharwal, & Sibal, 2013; Timo Mantere, 2005) and in some way superior than other as shown in (Harman, 2007). Harman (Harman, 2007) mentioned the historical reasons for this largely used rather than as a result of any strong theoretical indications. In addition to the complexity of the search space, the computationally expensive fitness functions is the main reason to use more complex SBTs (global SBTs) to optimize MBT. From the literature, the available papers that compared

between global and local SBTs are (Lefticaru & Ipate, 2008a, 2008b). The third type is hybrid techniques, which are recommended by many experts (Anand et al., 2013; Harman, 2007). However, there are few attempts (6 percent) to utilize SBTs with other testing techniques, such as EA with adaptive random testing (Iqbal et al., 2012a), GA with time partition testing (Lindlar et al., 2010), greedy search with constraint solver (Guglielmo et al., 2011), and HC with model checking (Enoiu et al., 2013). Researchers in the search-based community widely investigated hybrid techniques by combining two SBTs, whilst this type of SBTs did not gain interest in the MBT domain. In conclusion, there is a room for more research to develop more efficient hybrid techniques.

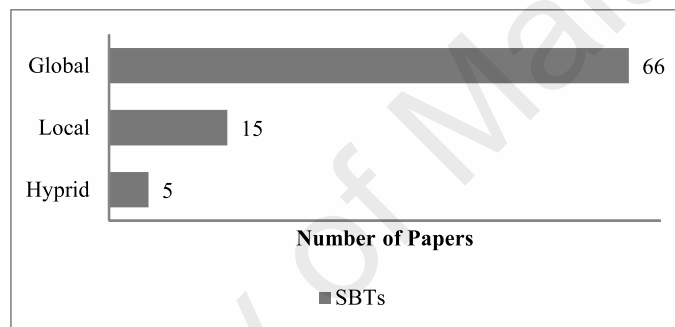


Figure 2.22: Distribution of papers based on the Type of Search-Based Techniques sub-category.

Among the papers that considered constraints, 18 percent did not present a constraint handling approach as described in Figure 2.23, this may be because of handling rigid constraint is non trivial task (Ali et al., 2013) and increases the chances of the search becoming stuck in local optima (McMinn, 2004). On other hand, the rest used general (25 percent), penalty (19 percent), repair (3 percent), and prohibit (1 percent) approaches. In the general approach, the researchers proposed different constraint satisfaction techniques such as heuristic-based constraint solver (Ali et al., 2013; Ali, Iqbal, et al., 2011), logic based constraint solver (Guglielmo et al., 2011), and constraint satisfaction (Lefticaru & Ipate, 2007, 2008a; Shirole, 2011). Additionally, a considerable number of papers (33 percent) did not consider constraint and constraint handling as well (refer to section

2.2.3.2: Figure 2.19).

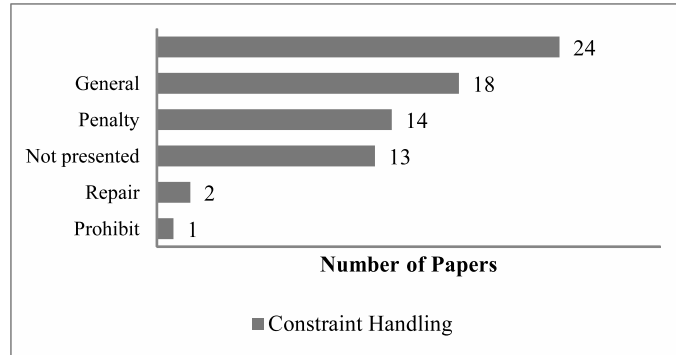


Figure 2.23: Distribution of papers based on the constraint handling sub-category.

Although, it is prevalent in the search-based optimization community to visualize the landscape, it has been observed that from Figure 2.24 most of the papers did not visualize the landscape (80 percent). Only 20 percent of the overall papers similarly presented the landscape visualization by using 2D (10 percent) or 3D (10 percent) planes. Study of Lefticaru and Ipate (Lefticaru & Ipate, 2008a) is the only available study from the literature which analyze the landscape of different fitness functions for MBT. This is an indication that there is a research space for conducting studies and applying all the concepts recommended by the SBTs community.

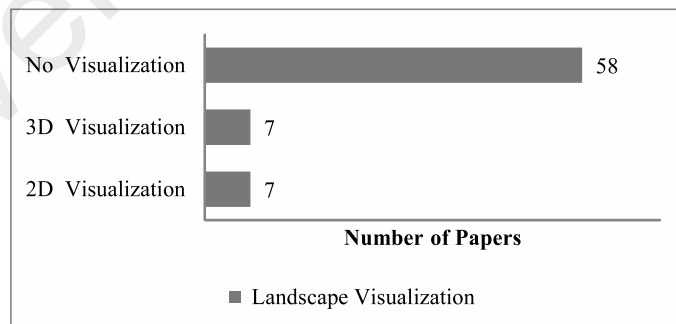


Figure 2.24: Distribution of papers based on the landscape visualization sub-category.

In the SBTs for MBT domain, the enhancing keys did not gain popularity compared to the applications of SBTs for code-based testing. Table 2.1 depicts the detailed results of each key point. From the complete set of papers, only one study (1 percent) focused

on seeding using other SBTs to enhance the performance, while the majority of the studies used random techniques (85 percent) or static data (5 percent) for generating first population. This lack of using SBTs as a seeding step indicates that more attention needs to be paid to seeding, as it showed successful performance in other domains (Grosan & Abraham, 2007). Concerning stopping criteria, Table 2.1 shows that none of the reviewed papers applied dynamic stopping criteria. Most of the attention was paid to employing static stopping criteria such as a maximum number of iterations (39 percent) or/and maximum fitness evaluations (35 percent). The other stopping criteria were given less attentions. Many of these studies utilized more than one stopping criteria (Marchetto & Tonella, 2010; Núñez et al., 2012; Kapfhammer, McMinn, & Wright, 2013; Lindlar et al., 2010; Ipate & Lefticaru, 2008; Ali et al., 2013; Lefticaru & Ipate, 2008b; Ensan et al., 2012; Vos et al., 2012; Alsmadi et al., 2010; Wilmes & Windisch, 2010; N. Li et al., 2012; Lefticaru & Ipate, 2007; A. S. Kalaji et al., 2011; Ali, Iqbal, et al., 2011; Zhan & Clark, 2008; Garousi, 2010; Lefticaru & Ipate, 2008a). Using dynamic stopping criteria can reduce the number of fitness evaluations and computation time as well (Hermadi, Lokan, & Sarker, 2014), which decreases the cost of the entire test case generation. The tuning concept attracted attention of only 3 percent of the papers, and the same level of attention was paid to novel other optimization steps.

Table 2.1: The detailed result of the optimization process sub-category.

Stopping Criteria							
Number of iteration	Reaching static fitness values	Running time	Maximum fitness evaluation	Finding optimal solution	Detecting fault	Others	Not presented
35	2	1	31	5	2	6	7
Seeding				Tuning	Others		
Random	Static Data	Heuristic Technique	Not presented		Crossover	Mutation	Chromosome
63	4	1	6	2	2	3	1

2.2.3.4 Evaluation Category

The results for each evaluation sub-category will be explained in this section.

As shown in Table 2.2, only 11 papers used one simple example to show the performance of the devised techniques, and one of them utilized one simple example as motivation and other complicated systems (Mani et al., 2010). 58 percent of the reviewed papers reported an empirical study using experiments, while the rest used industrial case studies. The attention of experiments is comparable with industrial studies, studies (Ali et al., 2013; Vos et al., 2012) presented that the applications of SBTs for MBT is still limited in industry. Three of these papers used both an industrial case study and experiments (Arcuri et al., 2010; Iqbal et al., 2012a; Ali et al., 2013). These high percentages indicate that the application of SBTs for MBT is important in both academic and industrial fields. From all these used dataset, studies that chose complex systems are (Núñez et al., 2012; Baresel et al., 2003; Marchetto & Tonella, 2010; Hemmati et al., 2010; Ali et al., 2013; Bühler & Wegener, 2008a; Vos et al., 2012; Arcuri et al., 2010; Ali, Briand, et al., 2011; Ali, Iqbal, et al., 2011).

Table 2.2: Used dataset for evaluation

Dataset for evaluation	Number of papers
Experiments	44
Industrial case study	24
One example	8

Table 2.3 shows the percentage of the used baselines. There was considerable percentage of the papers (16 percent) that did not present any comparison, while only 14 percent applied internal comparison by using the same techniques with some improvements or different settings. The external comparison based on random search gained most of the research attention with 35 percent of the overall papers, and the other techniques were similarly distributed between papers. The result of these studies presented that SBTs outperformed random search such as (Ali et al., 2013; A. S. Kalaji et al., 2011;

Kapfhammer et al., 2013; Guo et al., 2004). This is an evidence to show that MBT is a complex problem and needs to be solved by complex techniques, such as SBTs. Study (Yano, Martins, & Sousa, 2011) compared its results with Kalaji et al (2011) and Wong et al. (2013) used MOST tool (Yano, Martins, & de Sousa, 2011) as the baseline.

Table 2.3: The distribution of papers based on the baseline sub-category.

Literature comparison	Not presented	Internal comparison	Comparison with baseline				
			Random	GA	EA	BFS	Fourier series
2	12	11	27	6	3	2	1
			Constraint solving	PSO	SA	HC	Model checker
			3	2	5	1	2

From the reviewed papers as shown in Table 2.4, 55 percent of the papers utilized some type of coverage criteria, mostly with control-flow criteria (45 percent), followed by data coverage criteria (10 percent). Fault-based criteria (fault detection) gained 27 percent from all instances, in which mutation analysis is used in these papers in order to record the mutation score or the rate of the mutants killed. 5 papers used fault-based measures with others measures. There were 10 papers (14 percent) that applied fitness values of the solution as an effectiveness measure, while only 4 percent used time-based measure (execution time). Both coverage-based and fault-based measures, comparatively, received attention in this literature domain. This is because most of the studies on SBTs for MBT utilize functional and structural testing.

Table 2.4: The distribution of papers based on the effectiveness measures sub-category.

Coverage-based		Fault-based	Time-based	Fitness value
Control-flow	Data			
33	7	22	3	10

Table 2.5 summarizes the cost measures results. From the overall papers, 17 percent did not apply any cost measures. The remaining 83 percent of the papers were divided between the others cost measures. The only measure for test case execution (size of the

test suite cost) attracted low research attention. There is a clear lake in considering both the objective of the test case generation problem and the generation of a good test suite with low cost for executing as well.

Table 2.5: The distribution of papers based on the cost measurements sub-category.

Cost measures	Number of papers
Generation time	23
Number of fitness evaluation	16
Number of iteration	14
Not-presented	11
Size of test suite	7
Number of individuals	1

From the statistical test perspective as shown in Table 2.6, only 8 papers from the literature evaluated the effectiveness measure statistically (Ali, Iqbal, et al., 2011; Ali et al., 2013; Arcuri et al., 2010; Wilmes & Windisch, 2010; Windisch, 2010; A. S. Kalaji et al., 2011; Iqbal et al., 2012b; Zhao, Harman, & Li, 2010). Specifically, These studies presented different statistical tests such as both Mann-Whitney U-test and Fisher Exact test in (Iqbal et al., 2012b, 2012a), Mann-Whitney U-tests and t-tests (Arcuri et al., 2010), Fisher Exact test and t-test in (Ali et al., 2013), correlation analysis (A. S. Kalaji et al., 2011; K. A. Derderian, 2006; Zhao et al., 2010), and Mann-Whitney U-test (Wilmes & Windisch, 2010; Windisch, 2010). This low attention to the statistical evaluation is an obstacle to generalize that SBTs are sufficient for MBT and outperform the others techniques. More studies should be performed to provide vigorous evidence about the applicability of SBTs for MBT.

Table 2.6: The distribution of papers based on the statistical test sub-category.

Statistical test	Number of papers
Not-presented	59
Mann-Whitney U-test	5
Fisher Exact test	3
T-test	2
Correlation analysis	3

2.2.3.5 Cross Analysis between Problem and Solution Categories

Deeper analysis of the observations from sub-categories is worthwhile in order to well understand the domain. Therefore, the extended analysis of the data across both problem and solution categories is established by conducting cross analysis. Below is the discussion of the cross analysis based on the observations from Table 2.7, and Table 2.8.

Table 2.7 depicts the cross analysis between the major sub-category in the problem (adequacy criteria) and the related sub-categories in the solution categories. Because of the high complexity of optimizing the test case generation and the ability of global SBTs to successfully achieve the maximum coverage (Harman & McMinn, 2010; Ali, Briand, et al., 2010), the global SBTs are the most common techniques applied by the state-of-the-art research (presented in Figure 2.22). The majority of the papers (80 percent out of 30 papers) utilized global SBTs for the fault-based criteria, followed by 73 percent of the papers that applied global SBTs for the model-flow coverage criteria. Hybrid techniques have a lower proportion of the papers that applied the fault-based (7 percent) and the script-based (18 percent) approaches. As the notice from the gaps in Table 2.7, hybrid techniques are not devised to optimize the specific adequacy criteria. For instance, there are no papers optimizing model-flow, data-flow, and requirement-based criteria.

Table 2.7 also illustrates specific patterns concerning the fitness functions that are developed in the SBTs versus the adequacy criteria. There is a strong relation between adequacy criteria and the fitness function, wherein the fitness function is developed to cover as many as possible of the adequacy criteria. Because the process of implementing the adequacy criteria in the fitness function is different from one criterion type to another, the papers that developed a specific type of fitness function are more frequent. For instance, the specific fitness functions are mostly implemented the fault-based (83 percent), model-flow (38 percent), and requirement-based (80 percent) criteria. Moreover, specific fitness

Table 2.7: The cross analysis between the adequacy criteria sub-category with the fitness function, the Type of SBTs, the landscape visualization, and the optimization process sub-categories.

Adequacy Criteria	Fitness Function			SBTs		
	General	Specific	Not-presented	Global	Local	Hybrid
Model-flow	11	15	3	22	6	1
Data-flow	-	5	-	5	-	-
Script-flow	5	6	-	8	1	2
Requirement-based	0	4	1	5	-	-
Fault-based	6	25	1	24	4	2

Adequacy Criteria	Landscape Visualization			Optimization Process				
	2D Visualization	3D Visualization	No Visualization	Seeding	Tuning	Mutation	Crossover	Chromosome
Model-flow	5	1	23	1	1	2	2	1
Data-flow	1	-	4	-	-	1	-	-
Script-flow	-	1	10	-	-	-	-	-
Requirement-based	1	1	3	-	-	-	-	-
Fault-based	1	5	24	-	1	-	-	-

functions are utilized in all the papers that focused on data criteria but only half of the papers that concentrated on script-flow criteria. The general fitness function is used in a considerable number of the papers; 38 percent of the papers considered model-flow, and 45 percent of the papers focused on script-flow. It has been also noticed from the table that few number of the papers that did not mention any details about the fitness function.

Due to the high complexity of the search space of MBT (Ali et al., 2013; Ali, Briand, et al., 2010) and the landscape visualization is an open challenge in SBSE (Harman, 2007). Most of the papers did not visualize the search space as shown in Table 2.7, including 60 percent of the requirement-based papers, 80 percent of the fault-based papers, 79 percent of the model-flow papers, 91 percent of the script-flow papers, and 67 percent of the data papers. However, 3D visualization is frequent more than 2D in fault-based (17 percent), and script-flow (9 percent), while 2D is more applied in data (20 percent), and model-flow (17 percent).

There are clear gaps shown from Table 2.7 for improving the optimization process in requirement-based, fault-based, script-flow, and data. However, model-flow pay little attention to improving the optimization process with mutation (4 percent), seeding (3 percent), tuning (3 percent), crossover (7 percent), and chromosomes (3 percent).

Table 2.8 illustrates cross analysis of the various constraints and the constraint handling methods. The majority of the research studies handle constraints by using a general approach, such as 50 percent of papers with functional constraints, 50 percent of papers with timing constraints, and 100 percent of papers with integrity constraints. The second widely used constraint handling method is the penalty function, including a considerable percentage of papers with structural constraints (57 percent), with functional constraints (20 percent), and with source constraints (67 percent). A significant number of papers that did not mention any details about the constraint handling techniques. From the constraint handling perspective, the rate of papers that applied a penalty function as the constraint handling is less than the general techniques for all constraints and is equal to the proportion of the papers that did not present any constraint handling. However, prohibition and repairing constraint handling methods are not frequent in the SBTs for MBT domain. This may be because applying these techniques increases the complexity of the optimization process because they need more knowledge to obtain practical results.

Table 2.8: The cross analysis between the constraint and the constraint handling sub-categories.

Constraint	Constraint Handling				
	Penalty	Repair	Prohibit	General	Not presented
Structural	4	-	-	-	3
Functional	6	2	1	15	6
Timing	2	-	-	4	2
Performance	-	-	-	-	3
Sources	2	-	-	1	-
Integrity	-	-	-	1	-

2.2.4 Challenges

This section to address RQ3. According to the results and the analysis of the literature, it is obvious that SBTs for MBT have received considerable attention over the last decade and that worthy improvements have been proposed. The results also expose a number of conclusions that can help to preside future directions. The observations are presented as

various thought-provoking future research directions in the following subsections:

Complete empirical studies: From the reviewed studies, studies (Vos et al., 2012; Ali et al., 2013) are mentioned that the applications of SBTs for MBT of complex industrial software systems are limited. Based on the results in Table 2.2, industrial case studies gain less attention compared with experiments. Furthermore, from the result of statistical analysis in Table 2.6 and Table 2.3, only few papers that used other SBTs as baseline and 11 percent of the overall papers applied statistical analysis to show the significant performance of SBTs for MBT. As a result, more sufficient complete empirical studies must be performed to show a powerful evidence about the suitability of applying different SBTs for MBT for various testing purposes. In line with the idea of evidence-based software engineering (Dyba, , B.A. Kitchenham, & rgensen, 2005), this needs for further systematic analysis of systems that tested with and without the test cases generated by SBTs in order to increase the permeation of SBTs in industry. This systematic analysis should also include detailed successful economic benefits of applying SBTs for MBT.

Systematic guidelines for selecting suitable SBTs: Generally, there is a lack of systematic guidelines for selecting a suitable technique from the wide number of available SBTs for a specific MBT problem. Despite the considerable level of interest in SBTs for MBT, to date, there is not any theoretical and empirical analysis that characterized the complexity and the size of the given software system models for which different types of SBTs are predicted to be effective. Providing systematic guidelines of using SBTs for MBT is a necessity to help researchers in this domain. From the proposed taxonomies (Figure 2.7 to 2.10), the model type, application domain, constraint, type of SBTs, and fitness function can be the base of the systematic guidelines. Specifically, assuming problems of nontrivial size, the complexity

of the problem is the most significant factor that requires to be taken into account. Model type is the core of defining the problem. The constraints considered with a specific software system (application domain) is one of the components that defines the complexity of the optimization problem. For optimization, the coverage criteria and the computational cost are the aspects that one is interested in. There is a wide range of optimization algorithms available, which can be grouped into three main classes: global, local and hybrid techniques. The majority of the problems in MBT optimization can be solved by using global SBTs as shown in the result of Figure 2.22. Based on this investigation, a dataset can be established from the primary experiments and the systematic analysis of the systems; then, one can apply machine learning techniques to infer the relationships between different characteristics of the technique, given problem, and fitness function. From these relationships, systematic guidelines can be demonstrated.

Optimization process: The optimization process can be improved in many ways as discussed in 2.2.3.3. Based on the result summarized in Table 2.1, few studies applied some aspects of improving the SBTs performance for MBT such as tuning the technique parameter (Vos et al., 2012; Yano, Martins, & de Sousa, 2011), and seeding the proposed technique by using other SBTs output (K. Derderian et al., 2009). There are other aspects which did not gain interest to improve the performance of SBTs for MBT such as proposing the fitness function evolves itself, and applying dynamic stopping. Furthermore, a novel search-based optimization technique similar to that of (Fraser & Arcuri, 2013b) is a necessity, and it must be generalized with respect to the cost of executing the generated test cases and the high fault revealing power in the generated test cases.

Hybrid techniques: The hybrid SBTs can solve well any search problems that have unpredictable fitness landscapes, in which better perception of the search fitness landscapes may propose combining the best sides of the existing SBTs. Researchers successfully applied these hybrid SBTs in engineering areas, for example, PSO with GA (Kao & Zahara, 2008). In the test case generation context, researchers investigated other forms of hybrid search by applying SBTs either with dynamic symbolic execution, model checking, or other non-SBTs. To the best of the author's knowledge, there are few works on proposing hybrid techniques for MBT (as shown in Figure 2.22) by combining existing non-SBTs with SBTs (Iqbal et al., 2012a; Lindlar et al., 2010; Enoiu et al., 2013); furthermore, there are no attempts to combine two SBTs. In complex real systems, the result search space is very complex and the landscape contains several local optima and plateaux. Specially rigid constraints increase the chances of the search becoming stuck in local optima (McMinn, 2004). Consequently, further research is required for unified strong hybrid techniques, by combining two SBTs, to overcome SBTs limitations, such as becoming stuck in local optima, and to improve the fault revealing power in the generated test cases.

Model transformation: Based on the result of Figure 2.20, model transformation is one of the most important concerns in MBT. These model transformations can be regarded as a new form of testability transformation (at the model level). However, applying the aspects of the testability transformation in the model transformation did not gain interest in this literature. Therefore, a trusted testability transformation approach, as presented in (Friske & Schlingloff, 2007), which can be applied at the model level with respect to the limited characteristics and constraints of the system, is a necessity.

2.3 State-based Testing: Review

This section reviews the state-based testing.

2.3.1 Taxonomy

In this section, the test case generation from state machine models is comprehensively studied by analyzing the two challenges in the state-based testing. The taxonomy was devised based on analyzing 61 collected studies between 2005 until 2015 from five well-known data sources Springer, IEEE Explore, ACM, Science Direct, and Google Scholar. Figure 2.3.1 depicts the devised taxonomy and shows that the classification of the taxonomy is based on addressing the two challenges in the existing works involving two intrinsic categories: path generation and data generation. In particular, sub-categories were derived to deeply address all the information related to each challenge will be described. Each sub-category has a number of possible approaches that handle the sub-challenges. Next subsections discuss the characteristics of each category.

2.3.1.1 Path Generation

Four steps should be considered to address the path generation challenge which are: specify coverage criteria based on the testing purpose, construct test model from source model, generate all the paths in the test model, and detecting the infeasible paths from the all generated paths. Below is the description of each step.

Coverage Criteria: The first step is to specify a coverage criteria based on the test purpose. Coverage criteria are the measures used to describe the degree to which the source models of a SUT is covered in the generated test cases. Several coverage criteria were proposed and well studied in the literature for state-based testing, which are all transitions (AT), all transitions pairs (ATP), all round-trip paths (RTP), and full predicate coverage (FP), M-length path, and piecewise coverage. The last

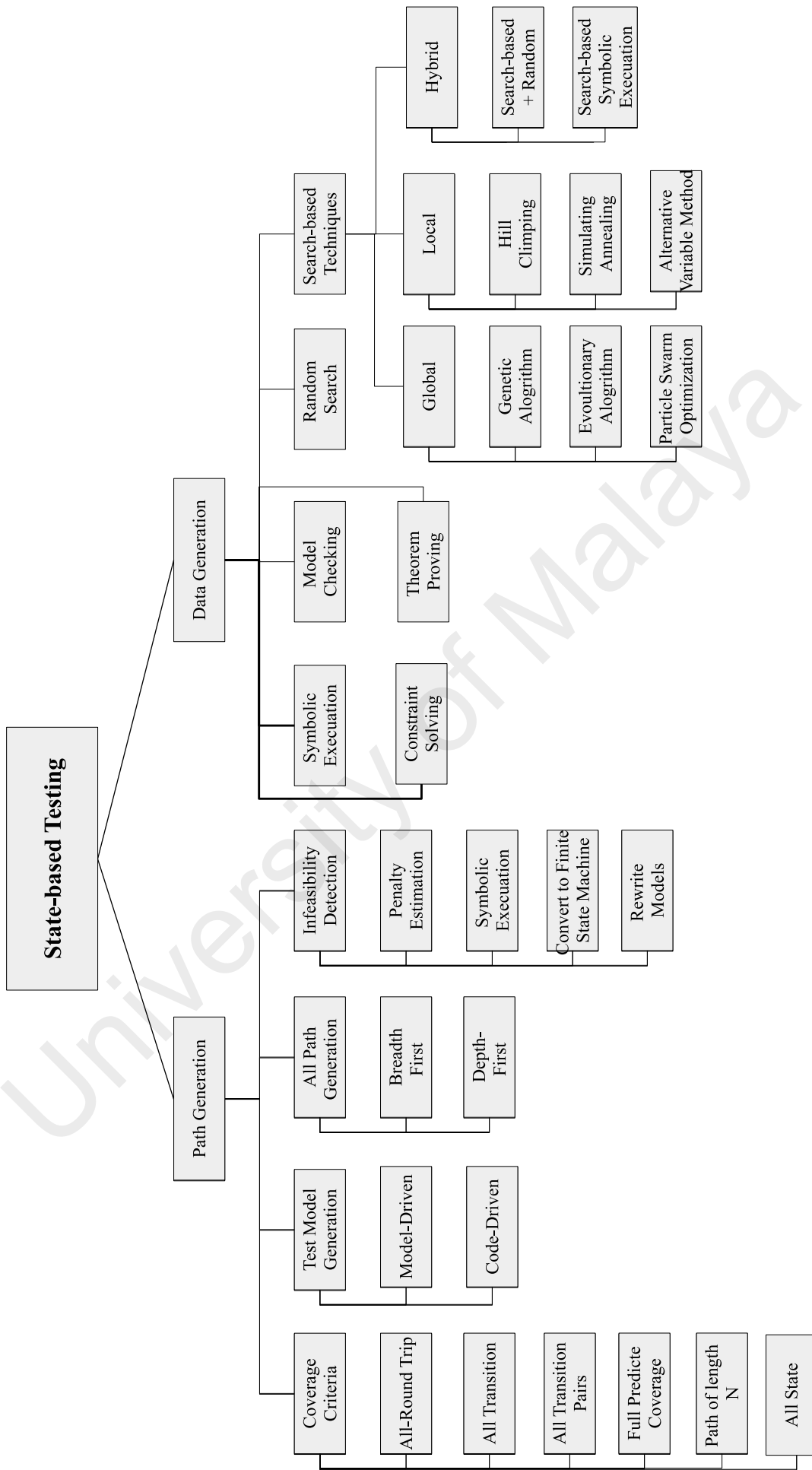


Figure 2.25: The Taxonomy of State-based Testing.

criterion is related on detecting the behavior faults in the SUT while the others focus on covering the structural elements of the model. The definition of these criteria based on studies (Binder, 2000; Offutt et al., 2003).

1. AT coverage is based on exercising every specified transition in a state machine at least once (Binder, 2000). Applying this criterion is to ensure that all states, events and actions are exercised, and is considered as being the minimum amount of coverage that one should achieve when testing software (Binder, 2000).
2. ATP coverage focuses on generating test cases that contain all pairs of adjacent transitions. For each pair of adjacent transitions from state i to j and from state j to k in the state machine, the test cases contain a test that traverses the pair of transitions in sequence.
3. RTP requires that all paths in a state machine model that starts and finishes with the same state must be covered. To cover all such paths, a test tree (consisting of nodes and edges corresponding to states and transitions in a state machine) is constructed and then traversed by breadth- or depth-first traversal strategy. In the test tree, a node is a stopped node either if the node previously exists anywhere in a tree that has been built so far or is a final state in the state machine model. By traversing all paths in the transition tree, all round-trip paths and all simple paths are covered. According to Binder (Binder, 2000), applying this criterion will find incorrect or missing transitions, incorrect or missing transition outputs and actions, missing states, and will detect some of the corrupt states.
4. FP coverage (FP) is based on ensuring that each clause in each predicate on

guarded transitions is tested independently (Offutt et al., 2003). M-length paths coverage concentrates on covering all possible sequences of transitions of length m from the initial state.

5. Piecewise coverage focuses on exercising each state, each event, or each action at least once. This criteria is not related to cover the structure of the behavioral model of SUT but it relates to detect the behavior faults of SUT due to applying this criterion may produce test case with missing some parts such as covering all states and missing some events (Binder, 2000).

Test Model Generation The second step is to construct a test model as transition tree model from source models. Two approaches are found in the literature. 1) The first approach concentrates on generating test model directly from source models using model-driven architecture concept, called model-driven approach. This model-driven approach proposed based on model-to model transformation in the literature for different purposes such as for transition tree (Ali, Hemmati, et al., 2010; L. Briand et al., 2010), system state graph (Sarma & Mall, 2009), LOTOS specification (Chimisliu & Wotawa, 2013a), and Satisfiability Modulo Theories (Cantenot et al., 2014) . Using the model-driven based on transformation, the abstract test cases are automatically generated using models extracted from software models through serial of model transformations. Model transformations process contains a source model, a target model, and a set of transformation rules that characterize how the elements from the source model are matched into elements of the target model. Therefore, the advantage of the model-driven approach that it is easily extensible and configurable for different context factors such as input models, test models, coverage criteria, test data generation strategies, and test scripting languages. One of the challenges of applying this approach is that additional effort is needed to develop

the transformation rules. Specifically, technical skills in the model transformation languages are required and are not popular in the software testing community. 2) The second approach is by using the models in form of code as a test model by either generate the auto code of the model using auto-code generation tools or by write the model in a code form.

All Path Generation : The third step is to generate all the paths in the test model using traversal strategy. The test model contains all the paths that generate based on applying an coverage criterion. Two traversal strategies have been applied in the literature: depth-first and breadth-first traversal. Depth-first traversing starts at the root and explores as far as possible along each branch before backtracking. Breadth-first traversal starts at the tree root and explores the neighbor nodes first, before moving to the next level neighbors. Using any traversal strategy does not affect the generated paths (Ali, Hemmati, et al., 2010).

Infeasible Path Detection : The final step is to detect the infeasible paths from the generated paths. The infeasible path contains conflict guards of the transitions such as a path contains guards $a > 10$ and $a < 10$. Detecting infeasible paths is important because not all the generated paths from previous step are feasible and generating data to trigger the infeasible paths is time-consuming. The existing infeasible paths detection method are 1) penalty estimation (K. Derderian et al., 2009; Shirole, 2011; A. S. Kalaji et al., 2011), 2) symbolic execution (Zhang, Chen, & Wang, 2004; Zhang, 2005; Galeotti, Fraser, & Arcuri, 2013; Delahaye, Botella, & Gotlieb, 2015), 3) convert to FSM (LI, LI, LI, & ZHANG, 2009), and 4) rewrite source models (Duale & Uyar, 2004). Penalty estimation depends on estimating the feasibility of the path by proposing integer values of each operation. This approach may restrict to

the estimated penalty values of the operations, which do not cover all the data types and operations. Symbolic execution is used to generate feasible paths and then data to trigger the paths. The symbolic execution is limited to the fundamental problems discussed in the next section 2.3.1.2. Converting an EFSM to a FSM by removing the guards and conditions so that FSM-based testing will be applied. This approach can suffer from the path feasibility problem since a path in the FSM formed by abstracting an EFSM may not correspond to an feasible path in the original EFSM. Also converting models can lead to the state explosion problem: the size of the resultant state space is exponential in the number of variables. Rewriting an EFSM to construct another form of EFSM that not suffers from infeasible paths. There is no general algorithm for solving this converting and the automated approaches impose significant restrictions on the EFSMs such as requiring linear actions and guards.

2.3.1.2 Data Generation

Data generation category refers to the used technique in order to generate the test data that triggers all the transitions and satisfies all the constraints in the state model. The category includes six techniques: symbolic execution (Galeotti et al., 2013; Delahaye et al., 2015; Prelguskas & Bareisa, 2012), model checker (Swarup Mohalik, Ambar A. Gadkari, Anand Yeolekar & Ramesh, 2014; Hamon et al., 2004), theorem proven (Cantenot et al., 2014; Brucker & Wolff, 2013), constraint solving (Zhang et al., 2004; Vishal et al., 2012), random search (Huang et al., 2015; Iqbal et al., 2012a; Arcuri et al., 2010), and SBTs (Ali et al., 2013, 2015; Blanco et al., 2009; Iqbal et al., 2012a).

Symbolic execution is a software analysis technique that analyzes a softwares code or model to automatically generate test data for the software. A large body of work exists that demonstrates the techniques usefulness in a wide range of software engineering

problems, including test data generation. However, the technique suffers from at least three fundamental problems that limit its effectiveness on real world software. 1) Path explosion, it is difficult to symbolically execute a significantly large sub-set of all program paths because most real world software have an extremely large number of paths, and symbolic execution of each program path can incur high computational overhead. Thus, in reasonable time, only a small subset of all paths can be symbolically executed. The goal of discovering a large number of feasible program paths is further jeopardized because the typical ratio of the number of infeasible paths to the number of feasible paths is high . 2) The inability to compute precise path constraints leads to path divergence. Path divergence is the path that the program takes for the generated test data diverges from the path for which test data is generated. Because of the path divergence problem, a symbolic execution system either may fail to discover a significant number of feasible program paths or, if the user is required to provide models, will be less automated. 3) Solving the general class of constraints is undecidable. Thus, it is possible that the computed path constraints become too complex (e.g., constraints involving nonlinear operations such as multiplication and division and mathematical functions such as sin and log), and thus, cannot be solved using available constraint solvers. The disability to solve path constraints reduces the number of distinct feasible paths a symbolic execution system can discover (Anand et al., 2013).

Model checking is a technology for verifying or falsifying properties of a system. For certain classes of properties, model checkers can yield counterexamples when a property is not satisfied. The general idea of test case generation with model checkers is to first formulate test case specifications as reachability properties, for instance, eventually, a certain state is reached, or a certain transition fires. A model checker then, by searching for counterexamples for the negation of the property, yields traces that reach the given

state or that eventually make the transition fire. Other variants use mutations of models or properties to generate test suites (Utting et al., 2011). The main problems of this technique are the space explosion and run out of memory when the input models are complex (Hamon et al., 2004).

Theorem proving is used for the generation of tests, particularly with provers that support the generation of witness traces or counterexamples. One variant is similar to the use of model checkers where a theorem prover replaces the model checker. Most often, however, theorem provers are used to check the satisfiability of formulas that directly occur as guards of transitions in state-based models. A theorem prover can compute assignments for the variables that occur in the guards and that, in turn, give rise to values of the respective input and output signals. A sequence of such sets of signals then becomes the test case (Utting et al., 2011). The problems of theorem prover is undecidable for non-trivial domains of inputs and need to write the models in specific formula such as encoded formalism (Brucker & Wolff, 2013).

Constraint solving is useful for selecting data values from complex data domains. It is also often used in conjunction with other methods such as symbolic execution, graph search algorithms, model-checking or theorem proving where specific relationships between variables in guards or conditions are expressed as constraints and efficiently solved by dedicated constraint solvers (Utting et al., 2011).

Random testing (RT) is one of the most fundamental and most popular testing methods. It is simple in concept, easy to implement, and can be used on its own or as a component of many other testing methods. It may be the only practically feasible technique if the specifications are incomplete and the source code is unavailable. Because of the simplicity of this technique, it is not efficient when applied for complex systems as reported in (Ali et al., 2013).

SBTs are generic algorithms that use the near optimal solutions are enough for solving the search problem. SBTs are divided into three types based on the search strategy global, local and hybrid SBTs. The full description of these types is discussed in section 2.2.3.3.

According to this thesis focus and the analysis of the taxonomy, next sub-section 2.3.2 reviews the studies in state-based testing that focused on UML state machine models with OCL constraints, according to the proposed taxonomy. Furthermore, because of the efficiency of the penalty estimation approach in detecting infeasible paths in EFSM context, the following sub-section 2.3.3 analyzes the studies that concentrated on penalty estimation for infeasible path detection in state-based testing domain.

2.3.2 UML State-based Testing

This section reviews the studies of generating test cases from UML state machine models with OCL constraints based on the proposed state-based testing taxonomy (in terms of path and data generation). Tables 2.9 and 2.10 summarize the studies that focused on generating test paths from state machine models and data to satisfy OCL constraints. In the Table 2.9, the columns refers to the following: test model refers to the approach used to construct test model from source models, constraint type refers to the covered constraint (either linear or non linear), data type refers to the used data types in the constraints, infeasibility detection refers to is the study provide detection method or not, and coverage criteria refers to the applied coverage criteria in the path generation process. In the Table 2.10, in the second column, the used technique is mentioned for generating data. The third column presents the size of the generated data based on the constraints variables, followed by if an approach translates OCL into another formalism before solving in the fourth column. The fifth column presents if the study supports three-valued logic, followed by other information related to the supported subset of OCL in the last column.

For the path generation, many approaches as shown in Table 2.9 have been proposed

in the literature to improve the coverage in the generated paths such as using model instrument (Friske & Schlingloff, 2007), data flow dependency (L. Briand et al., 2010), and data, control, and communication dependency (Chimisliu & Wotawa, 2013a, 2013b). In other perspective, Weißleder S. and Sokenou D. developed a tool for generating test cases based on boundary-based coverage (Weißleder & Schlingloff, 2008). Ali et al. (2010) improved the configuration and extendibility of the existing state-based testing tools using model transformation. Their tool has been used in many other studies (Ali, Iqbal, et al., 2011; Ali et al., 2013, 2015; Iqbal et al., 2012b; Holt et al., 2014).

From Table 2.9, only (Friske & Schlingloff, 2007) used auto generated code from the source UML models to be as the test model, while the authors of the study (Weißleder & Schlingloff, 2008) did not present any details about the test model construction. The rest studies used model-driven approach to construct test model as the following: Sarma M. and Mall R. (Sarma & Mall, 2009) proposed a model-driven approach based on constructing system state graph from UML use case, sequence and state machine models according to AT coverage to be as the test model. Study (Ali, Hemmati, et al., 2010) applied model transformation to transform the UML source models into a transition tree using ATL language based on RTP coverage criterion and then transformation from transition tree model into code using MOFScript language. However, study (L. Briand et al., 2010) applied the ATL transformation to construct transition tree based on data flow analysis and round trip coverage criterion. The authors of study (Chimisliu & Wotawa, 2013a) flattened the UML models to become simple models and transformed the models into LOTOS specification. Recent study (Cantenot et al., 2014) transformed the UML state machine models into Satisfiability Modulo Theories (SMT) instance and solved it by using SMT solver. From the Table, non of the studies proposed infeasible path detection for UML state-based testing with OCL constraints because OCL contains sophisticated

construct that should be considered.

With respect to test data generation, four studies considered solve OCL constraints to generate data by translating OCL into another formalism such as HOL (Brucker, Krieger, & Longuet, 2011), first-order formula (Cantenot et al., 2014), and DNF (Weiß leder & Schlingloff, 2008). Translation is an additional overhead and the other studies (Ali et al., 2013, 2015; Iqbal et al., 2012a) avoided such overhead by directly solving OCL constraints to generate test data. Moreover, it is not always possible to translate all OCL features and associated models into a target formalism. Furthermore, translation may result in combinatorial explosion. For example, study (Cabot, Claris, & Riera, 2008) concluded that the transformation of OCL to a SAT formula or a CSP instance can easily lead to combinatorial explosion as the complexity of the model and constraints increases. One example factor that could easily lead to a combinatorial explosion when converting an OCL constraint into an instance of SAT formula is when the number of variables and their ranges are large. Conversion to a SAT formula requires that a constraint must be encoded into Boolean formulas at the bit-level and, as the number of variables increases in the constraint, chances of a combinatorial explosion therefore increase. For industrial scale systems, this is a major limitation because the models and constraints are generally quite complex. Furthermore, the four studies used various types of techniques for test data generation after translating OCL constraints into other formalisms, including SMT solving (Cantenot et al., 2014), partition testing (B.-L. Li, Li, Qing, & Chen, 2007; Weiß leder & Schlingloff, 2008), and theorem proving (Brucker et al., 2011). In contrast, the other studies proposed SBTs for solving OCL constraints to generate test data.

The OCL supports three-valued logic which means each constraint in addition to being true or false may evaluate to undefined. The undefined value is commonly used to indicate an error in the evaluation of an expression such as divide by zero. Only two

Table 2.9: Summary of studies focused on path generation from UML state machine with OCL constraints.

Study	Test Model	Constraint	Data Type	Infeasibility Detection	Coverage Criteria
Friske & Schlingloff (2007)	Code-based	Linear	Primitive	-	Modified Condition /Decision Coverage, All States, All Events and All Transitions
WeiSSleder & Schlingloff (2008)	-	OCL	-	-	Boundary Coverage
Sarma & Mall (2009)	Model-driven	All	-	-	All Transitions and All States
Briand et al. (2010)	Model-driven	All	OCL	-	Round-Trip Path
Ali et al. (2010)	Model-driven	All	All	-	All
Chimisliu & Wotawa (2013)	Model-driven	-	Primitive	-	All Transition
Cantenot et al. (2014)	Model-driven	All	All	-	Specific Criteria

of them (B.-L. Li et al., 2007; Weiß leder & Schlingloff, 2008) did not handle the OCL three-valued logic. Furthermore, from the Table, all the studies solved one constraint by a time. However, there are common variables among the constraints in the generated test cases. Solving each constraint in the test case separately may generate more than value for the same variable and may are conflict values. Moreover, from Table 2.10, three studies did not handle some OCL features such as Collections (Weiß leder & Schlingloff, 2008), Enumerations (Brucker et al., 2011), special subset (OCLInState, OCLState) (Cantenot et al., 2014), and three-valued logic (B.-L. Li et al., 2007; Weiß leder & Schlingloff, 2008)). On other hand, approaches in (Ali et al., 2013, 2015; Cantenot et al., 2014) support a much more comprehensive subset of OCL.

2.3.3 Penalty-based Infeasible Path Detection

This section focuses on reviewing the studies focused on solving the problem of detecting infeasible paths in the state-based testing using penalty estimation approach. Infeasible path contains conflicted guards of the path transitions, while feasible path includes tran-

Table 2.10: Summary of studies focused on data generation from UML state machine with OCL constraints.

Study	Approach	Data Size	Translation to Formalism	Three valued Logic	Other
Li et al. (2007)	Partitioning Analysis	One	Yes	No	Not-discussed
WeiSSleder & Schlingloff (2007)	Partitioning Analysis	One	Yes	No	Not support collections and enumerations
Iqbal et al. (2012)	Hybrid SBTs	One	No	Yes	Support all OCL data type
Ali et al. (2013)	SBTs	One	No	Yes	Support all OCL data types
Brucker et al.(2011)	Interactive Theorem Proven	One	Yes	No	Support only OCL 1.4
Cantenot et al. (2014)	SMT Solver	One	Yes	Yes	Not support specific subset such as oclIn-State
Ali et al. (2015)	SBTs	One	No	Yes	Support all OCL data types

sitions with non conflicted guards. Figure 2.26 shows an example of path generation steps to differentiate between the generated paths into feasible and infeasible. From the figure, test model (transition tree) is constructed from UML state machine model based on RTP criterion. By applying depth-first traversal strategy, two path are obtained. The first path is feasible which contains one constraint $in_TheftFlashing = 1$ and does not conflicted by others . However, path 2 is infeasible because the path includes two constraints $in_TheftFlashing = 1$ and $in_TheftFlashing = 0$ and no generated data for $in_TheftFlashing$ variable is enable to satisfy these two constraints at the same time, so these two constraints are called conflict constraints.

The penalty estimation proposed in the literature as one constraint handling method (Coello, 2002; Aleti et al., 2013). This approach has been used for infeasible path detection recently in EFSM context. This section analyses the studies applied penalty estimation for detecting infeasible paths from EFSM models as summarized in Table 2.11. Derderian K. et. al (K. Derderian et al., 2009) proposed a fitness function to estimate the feasibility

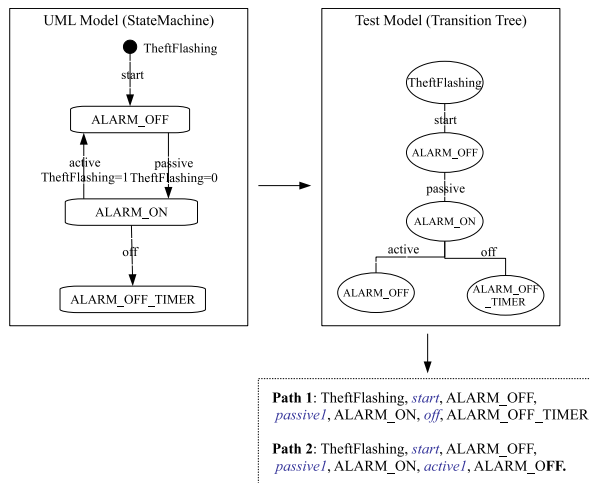


Figure 2.26: An example of the path generation from UML state machine models

of the path based on ranking the penalty values of transitions conditions. Lower fitness metric value means the probability of path feasibility is more. The approach in (A. Kalaji, Hierons, & Swift, 2009) proposed GA with extending the transition feasibility metric presented in (K. Derderian et al., 2009) by categorizing the state transitions into two types: affecting and affected-by. Moreover, they assigned penalty value to a certain path based on the assignment type, condition's guard type and the operator. In (Núñez et al., 2012), the authors extend the penalty estimating approach as presented in (K. Derderian et al., 2009) by considering not only the transition guards conditions (constraints) but also time temporal constraint. In (A. S. Kalaji et al., 2011), the authors extended their previous feasibility metrics (A. Kalaji et al., 2009) by using data flow analysis (static dependency analysis). In other study (Yang et al., 2011), transition feasibility metric in (A. Kalaji et al., 2009) was extended by distinguishing the guards condition into hard guards and easy guards. In UML context, study (Shirole, 2011) considered UML state machine model and converted UML models in to EFSM models. An Extended flow graph was constructed from EFSM to specify the control and data flow in a UML state machine models. The GA was utilized to specify feasible paths, in which fitness value for each path in a graph was calculated based on number of cycles and number of vertices in path. The feasibility was

determined by checking the value of field variables and set the parameter values to satisfy guard condition genetically. All these approaches in finding infeasible paths rely on the penalty values which are limited to be generalized due to the penalty values for integer data type with its basic relations (<, >, <=, >=, =, <>). In the UML context, the models should be transformed into EFSM models which not suitable for the UML models that contain complex guards conditions as in OCL.

Table 2.11: Summary of Infeasible path detection studies.

Study	Constraint Type	Data Type	relations	Detection Method
Derderian et al. (2009)	Linear	Integer	Basic	Penalty
Kalaji et al. (2009)	Linear	Integer	Basic	Penalty
Kalaji et al. (2011)	Linear	Integer	Basic	Penalty + dependency analysis
Yang et al. (2011)	Linear	Integer	Basic	Penalty + static analysis
Shirole (2011)	Linear	Integer	Basic	Penalty + static analysis
Nunez et al. (2012)	Linear + Time	Integer	Basic	Penalty

2.4 Research Gaps

From the existing research in the fields of state-based testing and search-based test data generation, the following research gaps were found that need further investigations.

2.4.1 UML state-based testing

The existing tools and methods of generating test cases from UML state-machine models do not support infeasible path detection. The existing methods generate all the paths and consider each path as test case, in which some of these test cases are infeasible and can not be executed. To generate executable test case, only the feasible paths are converted to be test cases. There is a need for developing infeasible paths detection to consider all the sophisticated constructs in the OCL.

2.4.2 Infeasible path detection

The infeasible paths have been studied in the literature in the EFSM context, in which only the integer data type is studied with its basic operations. Furthermore, The linear

constraint, which has one clause with one simple operation of integer variable has been only studied in the literature. Therefore, applying the existing penalty estimation detection methods are not efficient for the OCL constraints which are mostly non-linear and contain operations of different data types. Moreover, the OCL constraint may complex and contains more than one clauses. A detection method that considers non-linear constraints and complex data types and their operations is needed.

2.4.3 Search-based test data generation in MBT

The existing data generation methods in the UML context with OCL constraints evaluate only one constraints by a time. Evaluating and generating data to satisfy only one constraint by a time is not practical in the testing because some constraints depend on the same variables and generating data for a specific variable based on evaluating one constraints leads to generate various conflict values for the same variable. Optimizing the whole constraints at the same time is needed to generate efficient data of each variable in the whole constraints.

From these research gaps, there is a necessity to analyze experimentally the existing methods that generate test case and test data for UML state machine models with OCL constraints. This analysis is presented in next chapter.

CHAPTER 3: PERFORMANCE ANALYSIS FOR THE INFEASIBLE DETECTION METHOD AND SEARCH-BASED TEST DATA GENERATOR

The previous section presented the literature reviews of SBTs for MBT and state-based testing. This chapter discusses the empirically evaluation of the cost and effectiveness of existing model-driven path generator and search-based test data generator within the context of industrial systems ¹. The empirical case study method was adopted which used in (B. A. Kitchenham et al., 2002; Perry, Sim, & Easterbrook, 2004). To conduct the evaluation, an extensible model-driven approach based on RTP criterion was developed for automating the path generation from UML state machines with infeasible path detection method, and non-heuristic, global and local SBTs for test data generation for solving OCL constraints (Ali et al., 2013, 2015) were applied. The case study was conducted within the context of using two industrial public case studies which are taken from (Peleska et al., 2011). Seven common metrics were used to evaluate the cost and effectiveness.

The remainder of this chapter is visualized in Figure 3.1. Section 3.1 presents the followed case study method and section 3.2 describes the execution of the case study. Section 3.3 presents and discuss the results. Section 3.4 highlights the threats to validity. The chapter is concluded in section 3.5.

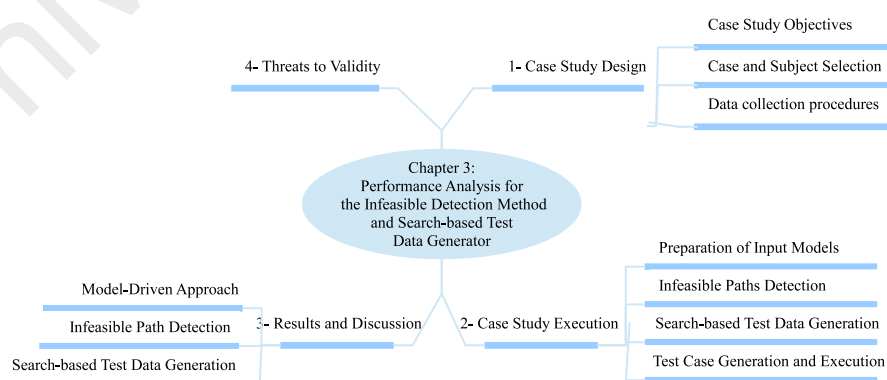


Figure 3.1: Semantic presentation of chapter 3 outline.

¹This section is part of the published article (Saeed, Hamid, & Sani, 2016).

3.1 Case Study Design

This analysis adopts the Empirical Case Study method in software engineering as defined in (B. A. Kitchenham et al., 2002; Perry et al., 2004). This method includes three steps: specifying objectives, case study selection with data collection, and case study design for executing and evaluating.

3.1.1 Case Study Objectives

The main purpose of this study is to evaluate the performance of SBTs and model-driven approach in terms of cost and effectiveness. The cost and effectiveness criteria were chosen due to their common use in test case generation domain (Holt et al., 2014). These criteria were also selected to give an evidence of applicability of using SBTs and MBT in industrial context. The goal/question metrics (GQM) template was followed in (Victor, 1992) to derive proper questions that embody the study objective:

RQ 1 : What is the cost and effectiveness of generating abstract test case using model-driven approach?

RQ 2 : What is the effectiveness of the infeasible paths detection method?

RQ 3 : What is the cost and effectiveness of generating test data using SBTs?

3.1.2 Case and Subject Selection

This study focuses on real-world complex industrial systems to ensure relevancy of this study results. The used case studies were developed by Daimler and European Vital Computer, turn indicator system (TIS) and ceiling speed monitoring (CSM) (Peleska et al., 2011). Daimler is a German automotive corporation and one of the largest car manufacturers in the world, while European Vital Computer is the main on-board controller for trains conforming to the European Train Control System specification.

The main factor for selecting these case studies is that they described concurrent real-time behavior of automotive applications and directly derived from industrial applications. These two case studies are publicly available as the benchmark of real time embedded systems (Peleska et al., 2011). Their models include UML state machines, class diagrams, and C language for constraint.

3.1.2.1 Case Study 1: Turn Indicator System (TIS)

Turn Indicator System (TIS) was developed by Daimler to control the turning of vehicle lights. The specification of TIS systems covers all functionalities in Mercedes Benz vehicles, comprising turn indication, varieties of emergency flashing, crash flashing, theft flashing and open/close flashing, as well as configuration-dependent variants. TIS specification is currently utilized by Daimler for testing the functionality of the turn indicator lights by automatically generating test suite, specified test data and test steps. Daimler publicly disseminated this specification to be as a real-world benchmark supporting MBT research community in 2011.

The systems input are signals and these signals exchanged between controllers which can be spotted by the testing environment. Moreover, this environment can activate and observe analogue and discrete communication between software under test (SUT) and peripherals, such as switches, buttons, indicator lights and several dashboard indications. The system output are captured as signals where part of them observable by end user. The SUT contains four main components:

1. Flashing component

- a) Normal and Emergency Flashing controls left/right turn indication, emergency flashing and the dependencies between both functions.
- b) Open Close Flashing controls the indicator-related reactions to the locking and

- unlocking of vehicles with the central locking system.
- c) Crash Flashing controls indications triggered by the crash impact controller.
 - d) Theft Flashing controls reactions triggered by the theft alarm system.
2. Conflict resolving component which resolves conflicts between indication-related commands.
3. Lights controller components
- a) Duration sub-component defines the periods for switching lights on and off through one flashing duration. These durations rely on the status of the ignition switch and the function to be executed.
 - b) Light sub-component assigns which lamps and dashboard indications should cooperate in the cycles of the flashing.
 - c) Message Handling sub-component transfers the duration and identification of affected lamps and indicators on a bus as message and coincides the flash cycles by re-transferring of the message at the starting of each flashing cycle.
4. Sub-component light control contains all output control functions; each function prevailing the flashing cycles of a single lamp or dashboard indicator.

The full description of the system is provided in (Peleska et al., 2011).

3.1.2.2 Case Study 2: Ceiling Speed Monitoring (CSM)

The European Train Control System (ETCS) is a signaling, control and train protection system designed to replace the incompatible safety systems currently used by European railways. ETCS depends on the onboard controller which is the European Vital Computer (EVC). The functionality and basic architectural features of EVC are depicted in the universal specification of ETCS system reported in (European Railway Agency, 2015).

Speed and distance monitoring is covered in one of the functional category of the EVC is to ensure *"the supervision of the speed of the train versus its position, in order to assure that the train remains within the given speed and distance limits."* (UNISIG, 2012). The brakes are triggered by the monitoring functions in case of violations of speed limit, when actual and allowed speeds to the train engine driver are displayed. Speed and distance monitoring contains three sub-functions ceiling speed monitoring (CSM), target speed monitoring, and release speed monitoring (UNISIG, 2012), and only one out of these three functions is active at a point in time. The part used in this study is the CSM functionality. CSM observes the maximum speed allowed regarding to the current most restrictive speed profile. The CSM is active when the train target is not approached such as train station, level crossing, or any other point that must be reached with predefined speed.

3.1.3 Data collection procedures

There are seven metrics used to measure the cost and effectiveness of model-driven approach and SBTs as used in (L. C. Briand, Society, et al., 2004; Holt et al., 2014). These metrics are common in test case generation domain (Holt et al., 2014). The GQM template in (Victor, 1992) is used to derive proper metrics that embody this study objective and questions:

The cost was measured in terms of the following metrics:

- Time of Preparation: The time taken on the transition tree generation, generating the transition tree paths, and building the test cases.
- Time of Generation: The time spent on generating data for the test case.
- Time of Execution: The time spent on executing test cases.

- Size of Test-suite: The number of generated test cases in terms of feasible and infeasible paths.

These metrics are independent from each other.

Effectiveness was measured by:

- State and transition coverage: The number of covered states and transitions in the generated test suites.
- Detection rate: The number of detected infeasible paths in the generated test suites.
- Success rate: The number of times the search-based technique was successfully obtaining a solution out of the total number of runs. In this study, success rate in solving the constraints in each feasible test case.

Timing data is collected by running the experiment on a Windows 7 64-bit operating system, machine with an Intel(R) Core(TM) i5 CPU @ 3.4 GHz processor, and with 4 GB memory. The time is measured in seconds.

3.2 Case Study Execution

This section describes the main activities in evaluating model-driven approach and SBTs for UML models, including the preparation of input models, developing various test data generation techniques, and the implementation of model-driven approach for test cases generation and execution.

3.2.1 Preparation of Input Models

The two case studies were modeled using Papyrus. This software was used due to it is an eclipse plugin and to be compatible with the other used software are eclipse plugins. One state machine was flattened in the TIS case study manually because the scope is that generating test cases not the flattening part of the model-driven approach. The summary of

the CSM and TIS case studies are shown in Table 3.1. Note that the original functionality has not been affected by the remodeling. The constraints in the both case studies were

Table 3.1: Case studies description.

State-machine feature	CSM case study	TIS case study
Maximum level of hierarchy	-	-
Number of State Machine models	3	39
Number of flattened submachine	-	4
Number of simple states	13	118
Number of transitions	17 (13 guarded)	164 (119 guarded)

written in C++ language. Therefore, the constraints were rewritten in OCL. The number of guards in CSM case study are 13 while guards in TIS case study are 119. Each guard contains conjuncted clauses which varies from one to four. The characteristics of the constraints are summarized in Table 3.2, including as well the details on the distribution of numbers of clauses. The data type of the variables used in the constraints are primitive types (real, integer, and boolean) and enumerations.

Table 3.2: Description of OCL constraints.

Conjuncted Clauses	Frequency (CSM)	Frequency (TIS)
4	-	15
3	1	15
2	2	38
1	9	51

3.2.2 Infeasible Paths Detection

The infeasible paths detection has been studied in EFSM context as reviewed in chapter 2 section 2.3.3. The recent detection method was proposed with SBTs by Kalaji et al. (2011). The method is based on penalty estimation with data flow dependency analysis in order to detect infeasible paths from EFSM models. Only the integer data type was covered with its related basic operations. The following is the Pseudo code of the infeasible path detection of Kalaji as illustrated in below:

Algorithm 1: Pseudocode of infeasible path detection based on penalty estimation and dependency analysis by Kalaji et al. (2011)

1:Input : path pf length n, EFSM relation matrix case constraints
2:Output: non negative integer value
3:Steps :
4:Set penalty of the guards that do not include context variable v
5:For all transitions in the guard
6:Check the dependency between transitions t_i, t_j
7:For all context variables in the t_i, t_j
8:Check the dependency context variable
9:Check the dependency type
10:Set the penalty value based on the dependency type
11:End For
12:End

3.2.3 Search-based Test Data Generation

Applying SBTs for generating data for UML state machines models with OCL constraints is non-trivial tasks due to OCL contains sophisticated constructs facilitating the definition of constraints. GA and SA SBTs were selected because they are the most commonly used global and local search algorithms in SBSE (Ali, Briand, et al., 2010). This section provides information about using GA and SA to generate test data for solving OCL constraints.

3.2.3.1 Fitness Function Definition

The fitness function proposed in (Ali et al., 2013) was applied, which is inspired from code-based testing in (McMinn, 2004). The fitness function $F(\text{TestCase})$ is based measuring the distance (so-called branch distance $d()$) as shown in equation (3.1). The equation normalizes the distance value. The $d()$ returns 0 if the constraint has a solution or a positive value, which heuristically estimates how far the constraint was from being evaluated to true. The details of how calculating the branch distance $d()$ can be found in (Ali et al., 2013).

$$F(\text{TestCase}) = 1 - 1.05^{-d()} \quad (3.1)$$

3.2.3.2 Representation of Problem

Each generated abstract test case has number of transition guards and state invariants represented by OCL constraints. One OCL constraint (P) represents as a set of Boolean clauses (C_1, C_2, \dots, C_N) linked by Boolean operations (*and, or, implies, xor, not*). Each Boolean clause C_i is defined over a number of variables ($V_{Ci1}; V_{Ci2}; \dots; V_{Cij}$). The constraint P needs to be solved by generating data for its variables using a SBT, which can be symbolized as a set of variables: $\bigcup_{x=1}^n \bigcup_{y=1}^{m_x} (C_{xy})$ where n is the clauses number in a constraint and m_x is the variables number included in the x th clause. Note that m_x may be different across clauses.

3.2.4 Test Case Generation and Execution

Model-driven approach (Ali, Hemmati, et al., 2010) was implemented that apply model-to-model and model-to-text transformations (Figure 3.2). Using two main steps: First step, the input models are transformed into transition tree model based on RTP criterion using ATL model-to-model transformation language. The transformation ATL rules take the source UML state machine models and two meta models: transition tree metamodel and UML 3.0 metamodel. The two metamodels are ecore files, in which the transition tree metamodel is taken from (Ali, Hemmati, et al., 2010) and modeled using Eclipse Modeling Framework (EMF) (Figure 3.3). The UML 3.0 metamodel is provided by EMF. The output of this step is a generated test model (transition tree) in XML file, where each state and its association (state invariants) in the state machine model is a node in the transition tree, and the transitions and its associations (event, guard, and an effect) is an edge in the test tree model. The second step, the test model is transformed into executable test cases using MOFScript language. The step includes traversing the test model (the transition tree) to get all paths in the transition tree, and transforming each one path into one Java based

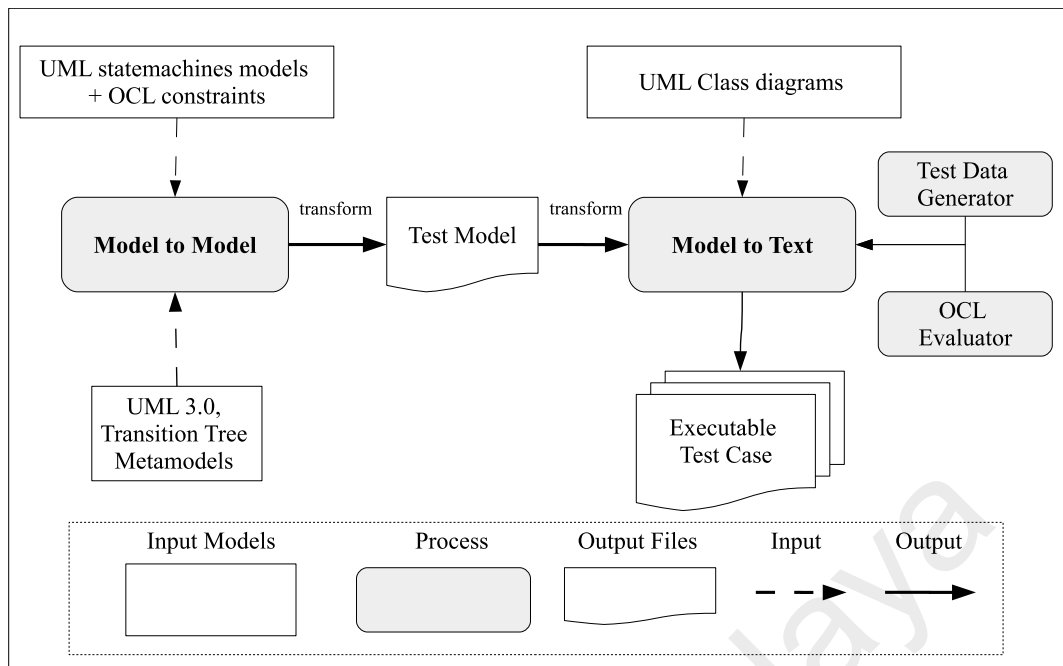


Figure 3.2: The model transformation approach architecture used for model-based testing.

abstract test case. In this step requires MOFscript rules and the two metamodels: the transition tree models and UML 3.0. The output is a set of Java files (test cases) and each Java file represents one abstract test case. The generated test cases are then checked by the infeasible paths detection method in order to detect real infeasible test cases caused by unsatisfied guard conditions. To make the abstract feasible test cases executable, test data for each of them is generated using test data generator.

The search-based test data generator aforementioned in section 3.2.3 is based on three techniques, 1) non- heuristic SBTs (random), 2) global SBTs (GA) and 3) local SBTs (SA). The configuration of the SBTs used in this research is shown in Table 3.3. The OCL evaluator, EyeOCL Software, was used to evaluate the generated data by satisfying the OCL constraints . To use OCL evaluator, the class and object diagrams were constructed from the SUT class diagram called the OCLWrapper. To automate the build, execution, and time data collection, a batch file was created to contain all the generated test cases, the OCLWrapper, and the test data generator. In the execution, EyeOCL was used at runtime

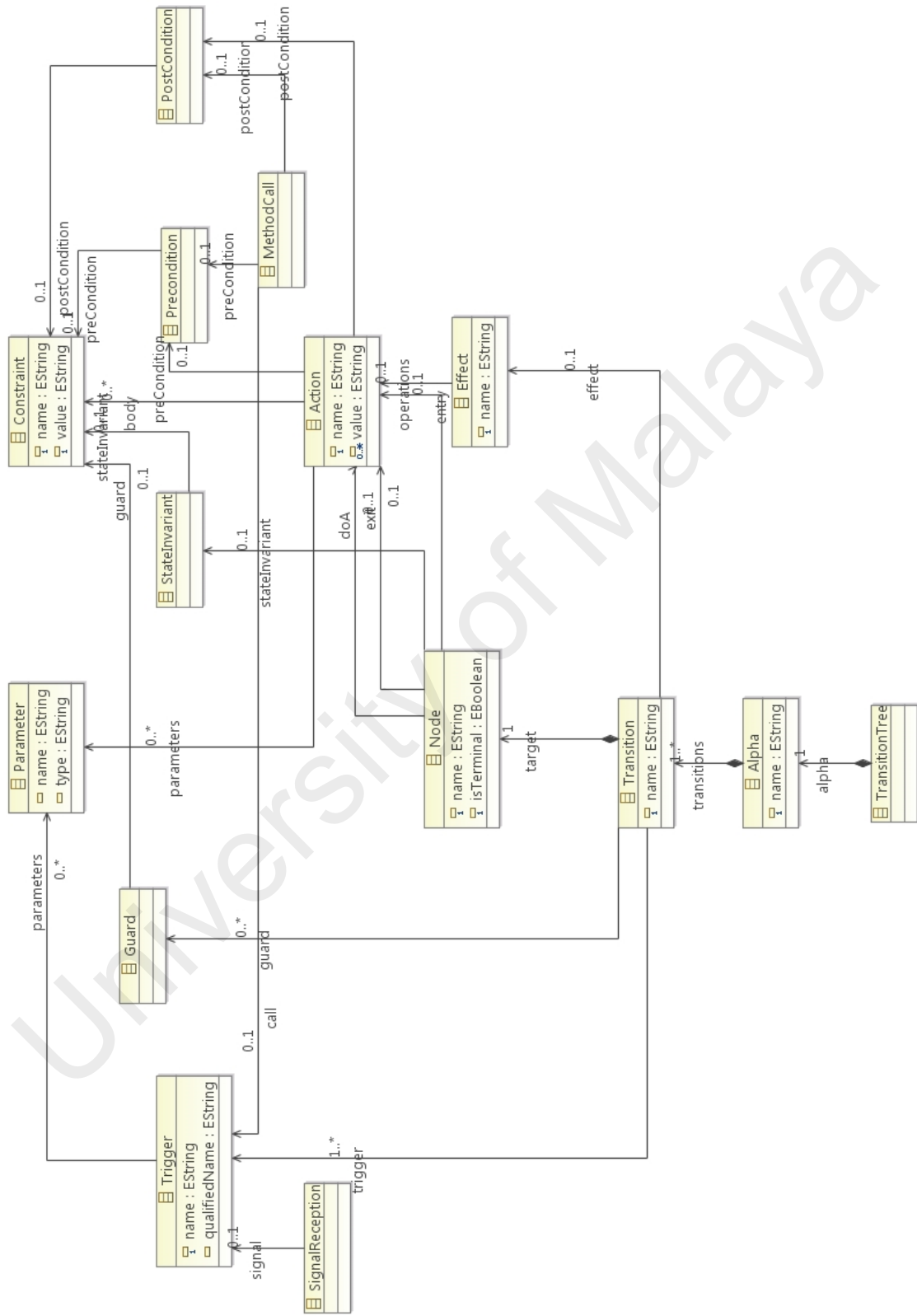


Figure 3.3: The transition tree (test model) metamodel.

to check the constraints in the test case script and then each test case is invoked on a new instance of the SUT.

Table 3.3: Configuration of search-based techniques.

Parameter	Value
Fitness evaluations	2000 times
Number of iterations	1000
Crossover rate	0.75
Population size	1000
Mutation	$1/n$, where n is the number of variables

3.3 Results and Discussion

This section discusses the findings based on the study objectives presented in section 3.1.1. The cost and effectiveness of model-driven approach in generating test cases from the two industrial case studies in terms of feasible and infeasible test cases, state and transition coverage, and preparation time are presented in section 3.3.1. The detection rate of the infeasible path detection method is described in section 3.3.2. The effectiveness and cost of SBTs in generating test data for each feasible abstract test case in each case study in terms of success rate, and generation time are discussed in section 3.3.3. In the same section, statistical t-test has been applied to analyze the results statistically to show the significant differences between the three techniques, as well as, the result of the t-test on the success rate and the result of the t-test on the generation time of the three SBTs.

3.3.1 Model-Driven approach

The result of this section is to address the problems stated in RQ1 (What is the cost and effectiveness of generating abstract test case using model-driven approach?). The evaluation metric related to the effectiveness of model-driven approach are the state and transition coverage. The metrics related to the cost are the preparation time, the execution time and the size of the test cases. The size of the test cases were measured by the number

of all generated abstract test cases and how many of them are feasible and infeasible. Table 3.4 shows significant state and transition coverage of model-driven approach for generating test cases, in which 100 percent coverage have been obtained with minimum cost (preparation time). Although the model-driven approach has taken minimum cost, the number of generated infeasible test cases is high in both case studies. The reason for this is that the model-driven approach does not support any infeasible paths detection. From the generated data by test data generator, the execution time for executing the test cases of TIS is more due to its large size as compared to CSM case study.

Table 3.4: Result of model-driven approach.

Evaluation Metrics	CSM Case Study	TIS Case Study
Generated abstract test cases	26	161
Generated feasible test cases	9	29
Generated infeasible test cases	17	132
State coverage	100	100
Transition coverage	100	100
Preparation time (Seconds)	0.5	1.5
Execution time (Seconds)	97	191

3.3.2 Infeasible Path Detection

The RQ2 (What is the effectiveness of the infeasible paths detection method?) is addressed in this section. The effectiveness metric of infeasible paths detection is detection rate. Figure 3.4 shows the results of the infeasible paths detection method in the two case studies. The infeasible path detection method achieved only 29 and 18 percent detection rate. The reason for the low detection rate is that the Kalaji method is limited to Integer data types and its related basic operations, while case studies contain boolean and enumeration data types.

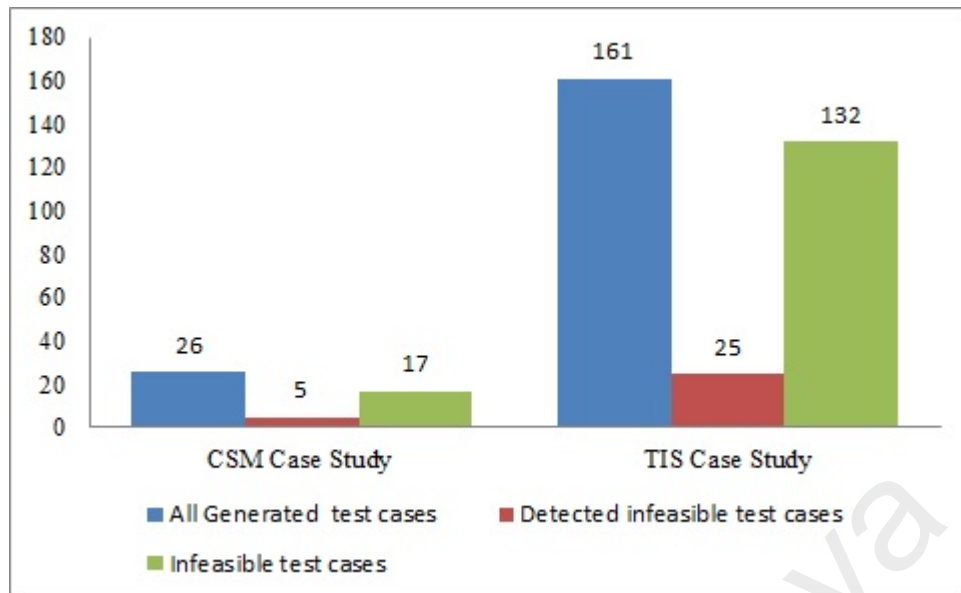


Figure 3.4: The results of the detection method.

3.3.3 Search-based Test Data Generation

This section is to answer RQ3 (What is the cost and effectiveness of generating test data using SBTs?). The evaluation metrics for the cost is the generation time and for the effectiveness is the successful rate. From Table 3.5 and 3.6, SA and GA achieved better success rate than RS in both case studies. SA significantly outperformed the RS and GA in CSM while its performance is roughly similar to GA in TIS. In specific, SA significantly outperformed both RS and GA in all test cases in CSM but it is slightly better than GA in TIS. GA and RS closely achieved the same success rate in CSM. GA greatly outperformed SA with more generation time in test case numbers 11,12, and 22 and with less generation time in test cases 19 and 23. SA achieved 100 percent in the first six test cases in both case studies, and GA obtained 99 percent in test case 12.

From the results, it has been observed that SA performed more efficiently when the number of constraint in the test cases is high and the number of clauses in each constraint is simple (containing 1, 2 or 3 clauses), which is the situation of test cases of CSM case study. The test cases with higher number of constraints are more difficult to be solved

Table 3.5: The results of successful rate and generation time for each GA, SA and RS techniques in case study 1 (CSM).

CSM Case Study		Successful Rate			Generation Time		
Test Case ID	No of constraints	GA	RS	SA	GA	RS	SA
1	1	0.21	0.28	0.7	95.12	10.36	110.52
2	4	0.19	0.2	0.75	64.12	22.56	70.43
3	5	0.31	0.5	0.8	102.35	14.16	71.41
4	4	0.31	0.25	0.83	98.33	9.31	106.96
5	4	0.25	0.4	0.76	101.82	9.97	103.04
6	5	0.24	0.2	0.75	101.26	11.88	103.76
7	5	0.16	0.17	0.82	75.29	12.57	103.02
8	6	0.45	0.1	0.67	1.08	5.78	3.72
9	2	1	0.5	1	1.00	0.001	1.55

by GA and RS as shown in Tables 3.5 and 3.6. Specifically, the success rate inversely proportional to the number of the conjuncted clauses. With respect to the cost, RS is faster compared to the GA and SA, taking few seconds for generating the data. In CSM case study, SA costs more than GA in all the test cases except the test case number 3. However, the time taken by SA is wavering with GA in the TIS case study. In conclusion, GA and SA significantly outperformed RS in success rate with high cost in both case studies, while the performance of SA is superior than GA with more generation time in the CSM case study. It has approximately the same performance and generation time of GA in TIS case study. Generally, the long time for generating data, the better success rate is for solving constraints.

With respect to statistical check, the statistical paired t-test is carried out on the distributions of the success rates and generation time of all the three techniques. Table 3.7 shows the statistical difference based on success rates and Table 3.8 depicts the differences of the cost (generation time). Table 3.7 shows that the p-values are very close to 0 in some distributions compressions in both case studies. This shows that there is a significant statistical difference between the performance of the three techniques. In CSM case study, the performance of SA is statistically superior than the others due to the p-values of

Table 3.6: The results of successful rate and generation time for each GA, SA and RS techniques in case study2 (TIS).

TIS Case Study		Successful Rate			Generation Time		
Test Case ID	No of Constraints	GA	RS	SA	GA	RS	SA
1	1	0.81	0.2	1	0.5	0.001	1.25
2	1	1	0.99	1	0.7	3.81	4.18
3	2	1	0.99	1	1.28	5.85	6.69
4	1	1	0.99	1	0.72	4.57	3.7
5	2	1	0.99	1	1.31	6.57	6.58
6	3	0.73	0.48	1	4.537	4.32	7.91
7	2	0.15	0.14	0.36	107.74	7.99	364.99
8	2	0.42	0.24	0.56	62.85	7.6	66.4
9	2	0.31	0.12	0.35	110.32	7.59	86.57
10	2	0.31	0.12	0.35	98.73	7.75	96.98
11	2	0.92	0.23	0.32	136.37	7.55	112.32
12	2	0.99	0.25	0.35	190.97	7.48	126.28
13	2	0.18	0.27	0.31	124.9	7.2	122.73
14	2	0.21	0.25	0.31	125.65	7.74	122.18
15	2	0.17	0.26	0.31	128.73	7.79	124.9
16	2	0.20	0.24	0.32	128.95	7.24	122.73
17	2	0.18	0.23	0.31	128.15	7.68	124.82
18	2	0.20	0.22	0.26	123.84	7.11	125.06
19	2	0.65	0.99	0.25	33.74	7.04	36.46
20	1	0.72	0.48	0.97	7.67	4.55	5.5
21	1	0.69	0.52	0.98	8.13	4.12	5.56
22	2	0.79	0.25	0.55	13.7	7.53	9.55
23	2	0.79	0	0.56	6.85	4.65	9.56
24	4	0.43	0.24	0.57	125.43	12.39	134.8
25	2	0.16	0.24	0.26	125.71	7.64	130.13
26	2	0.22	0.26	0.30	96.88	7.8	98
27	2	0.22	0.24	0.28	103.76	7.51	100.09
28	2	0.23	0.5	0.32	124.16	4.32	103.59
29	4	0.72	0.02	0.96	9.45	7.61	9.37

SA are close to 0. In TIS case study, the p-value of SA against GA shows that there are no significant difference between the performances of them while p-values of their comparisons with RS show that SA and GA are better than RS. From Table 3.8, the p-values of RS versus the others clearly indicate that RS is faster than others, while SA and GA take approximately the same time for finding solutions in both case studies.

For general overview, Figure 3.5 illustrates the success rate of three SBTs based on the 38 test cases; 9 test cases of CSM case study, and 29 test cases of TIS case study.

Table 3.7: The results of the paired t-test based on successful rate.

Pairs of Techniques	p-value	
	CSM Case study	TIS Case study 2
GA vs SA	9.02743E-05	0.30909013
GA vs RS	0.233288588	0.00585219
SA vs RS	5.64267E-07	0.00263052

Table 3.8: The results of the paired t-test based on generation time.

Pairs of Techniques	p-value	
	CSM Case study	TIS Case study 2
GA vs SA	0.244773935	0.309263749
GA vs RS	0.000819209	6.40878E-07
SA vs RS	0.000752768	1.03843E-05

Median, maximum, minimum, first quartile (Q1), and Third quartile(Q3) values of success rates of the three techniques were considered. The Q1 is defined as the middle number between the smallest number and the median of the data. The Q3 is the median of the upper half of the data set. From the Figure, SA achieved better results than the others with average success rate of 61 percent. GA outperformed RS with average success rate of 49 percent and 36 percent respectively. It can be observed that, all the techniques achieved 100 percent for at least one test case. With the upper limit of 1000 iterations, both RS and GA achieve almost the same median success rates (35 and 31 percent) and SA exceeds better median success rate of 57 percent. It can be also concluded that the lowest success rate of SA is approximately 25, whereas the tendency of the lowest success rates of GA and RS are towards 0. The highest 25 percent of the SA, GA, and RS success rates are respectively 92, 78, and 48 percent, while the lowest 25 percent of their success rates are 31, 21, and 20 percent. Overall, all the values in the figure refers to the preference of SA as compared to the RS and GA.

The explanation of the difference between GA and SA performances is that SA does local search and GA applies global search. Moreover, if the fitness landscape (search space) gives an obvious tendency into the global optima (best solutions in all search

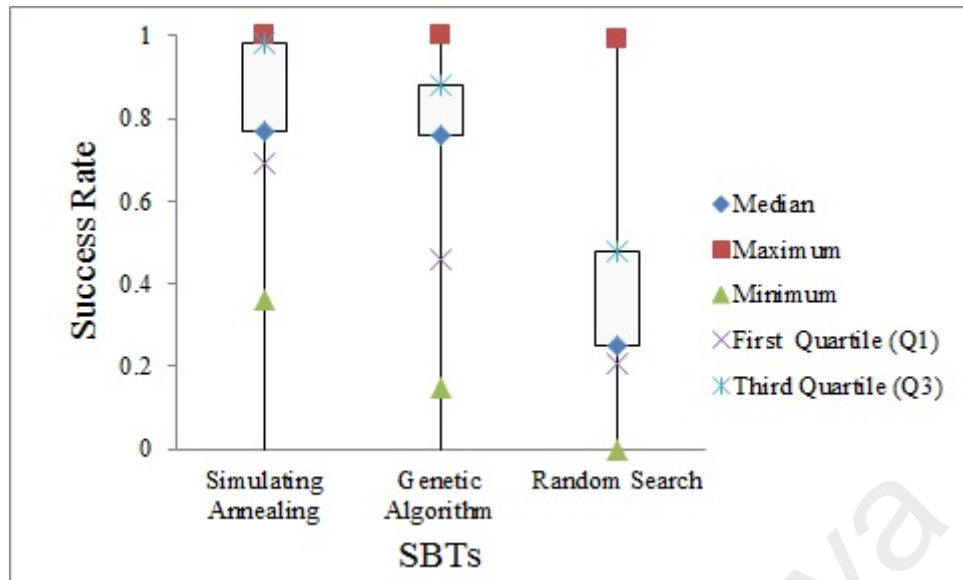


Figure 3.5: Box plot graph of success rate for RS, GA and SA in both case studies.

space), then SA concentrates on one of them. On the other hand, GA explores all the landscape. If there is an obvious tendency into the local optima (best solution in part of the space), then the GA avoid these by exploring the other search space, while SA stuck and has to restart from other point of search space. The search landscape of the case studies consists only a few local optima; a type of landscape in which local search enable to give efficient results.

3.4 Threats to Validity

The developing of the model-driven approach and the test data generation SBTs may not be as conventional as the existing technique. To address this, the same tools and transformation languages were used for implementation. Also, the same library of SBTs (jmetal library) was used and the developed fitness function (distance function) similar to the ones provided in (Ali et al., 2013).

The reliability threat defined in (Runeson & Höst, 2008) that affects the robustness of this study is that an unclear, and not detailed description of data collection may give different results in case of repeating the study experiments. This threat was mitigated by

presenting the design and analysis parts of this study in details.

The conclusion threat that may affect the experiments is the random variation. To mitigate this, the experiments were reiterated 1000 times to decrease the likelihood that the obtained results are caused by chance. In addition, statistical test has been conducted to analyze statistically the results.

The internal threat that may reduce the validity is that the experiments were performed with one configuration setting for parameters of SBTs (SA and GA) . To address this, the default settings were set which are in accordance with common guidelines in the literature. At the same time, similar stopping criterion was applied which is the maximum fitness evaluation for all the SBTs.

3.5 Conclusion

This chapter provides empirical evaluation of the cost and effectiveness of UML state-based testing for generating executable test cases and SBTs for generating test data based on two industrial case studies. The UML state-based testing approach adopted model-driven based on serial model transformations. Three SBTs have been involved in this evaluation, including global SBTs (GA), local SBTs (SA), and non-heuristic (RS). Seven evaluation criteria were considered that measured the cost and effectiveness of both model-driven approach and SBTs, comprising state, and transition coverage criteria, size of test cases, detection rate, success rate, generation time, execution time and preparation time. The results of three SBTs to solve the OCL constraints have been statistically tested using t-test.

The result shows that model-driven approach achieved 100 coverage with minimum cost (time); however, high number of the generated test cases are infeasible. The infeasible path detection method can detect the infeasible paths with simple constraint contains integer data type only. From the results of search-based test data generator, it has been

concluded that GA and SA significantly outperformed RS with high cost in both case studies, while the performance of SA is superior than GA with more cost (generation time) in CSM case study and it has approximately the same performance and generation time of GA in TIS case study. The result of SA aligns to the result of SA presented in (Mansour & Salame, 2004), which reported that SA tends to achieve slightly better than GA in terms of the number of executed paths. Generally, higher success rate is often associated with longer generation time.

Based on the results, further research is needed to improve the existing model-driven approach for UML state machines with OCL by detecting the infeasible test cases and also to improve search-based test data generator by enhancing the fitness function that solving all the constraints at one execution time. In next chapter, the proposed method will be presented.

CHAPTER 4: METHOD FOR GENERATING FEASIBLE EXECUTABLE TEST CASES FROM UML STATE MACHINE MODELS WITH OCL CONSTRAINTS

The previous chapter presented the empirical analysis for the existing model-driven path generator, infeasible path detection method, and search-based test data generator. This chapter presents an in-depth explanation of the proposed method for generating feasible executable test cases from UML state machine models with OCL constraints. The method consists of three major components, 1) generating all paths based on RTP coverage criterion, 2) detecting infeasible paths using proposed static analysis and 3) generating data based on error feedback for satisfying whole constraints at the same time. This study covers all the OCL data types in the infeasible path detection and search-based test data generator. Later, all the rules of handling OCL data types and operations in the infeasible path detection, the method of optimizing the whole constraints, and the fitness function that evolves itself by using error feedback are presented. Significance and novelty of the proposed method are also presented at the end of this chapter.

The remainder of this chapter is illustrated in Figure 4.1. Section 4.1 presents the proposed method and describes its components. Section 4.2 highlights the significance and novelty of the proposed method. The chapter is concluded in section 4.3.

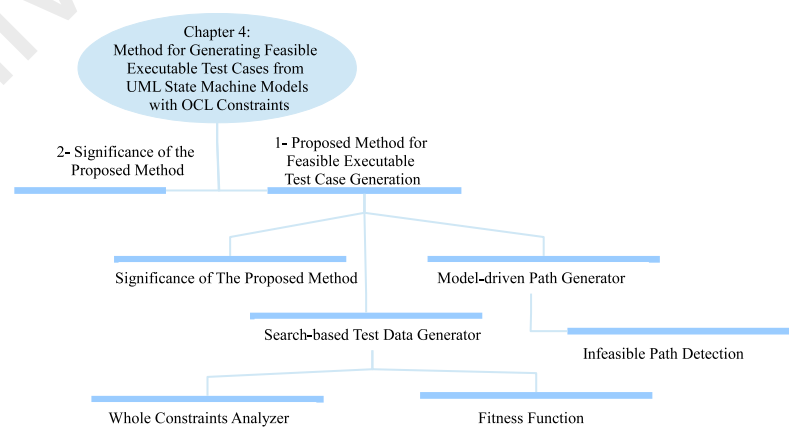


Figure 4.1: Semantic presentation of chapter 4 outline.

4.1 Proposed Method for Feasible Executable Test Case Generation

A method is proposed based on model-driven approach and SBTs for generating feasible executable test cases from UML state machine models with OCL constraints. Fig 4.2 shows the proposed method which consists of two generators with three sub-components to complete the process of generating feasible executable test cases: model-driven path generator for generating feasible abstract test cases and search-based test data generator for generating optimal data satisfies all OCL constraints in each abstract test case. The three sub-components are proposed for 1) infeasible path detection in model-driven path generator, 2) whole constraints analyzer and 3) fitness function based on error feedback in the search-based test data generator.

4.1.1 Model-driven Path Generator

Model-driven path generator is a model-driven approach based on penalty estimation with model-to model and model-to text transformations. This Model-driven path generator consist of two components 1) path generator based on RTP criterion and 2) infeasible path detection. The path generator is similar to model-driven applied in section 3.2.4. The difference is that the infeasible path detection is implemented in the model-to-text transformation. In the first part of the model-driven path generator, the same steps of model-to model transformation in section 3.2.4 are applied in order to generate test model. Second part is different from section 3.2.4 which is transforming the test model into executable test cases, including traversing the test model (the transition tree) to get all paths in the transition tree, and check the feasibility of each path using the proposed infeasible path detection method. The infeasible path detection method is based on a set of static rules based on penalty estimation is proposed (Section 4.1.1.1). These different rules are investigated based on static analysis for different data types of OCL. Each feasible

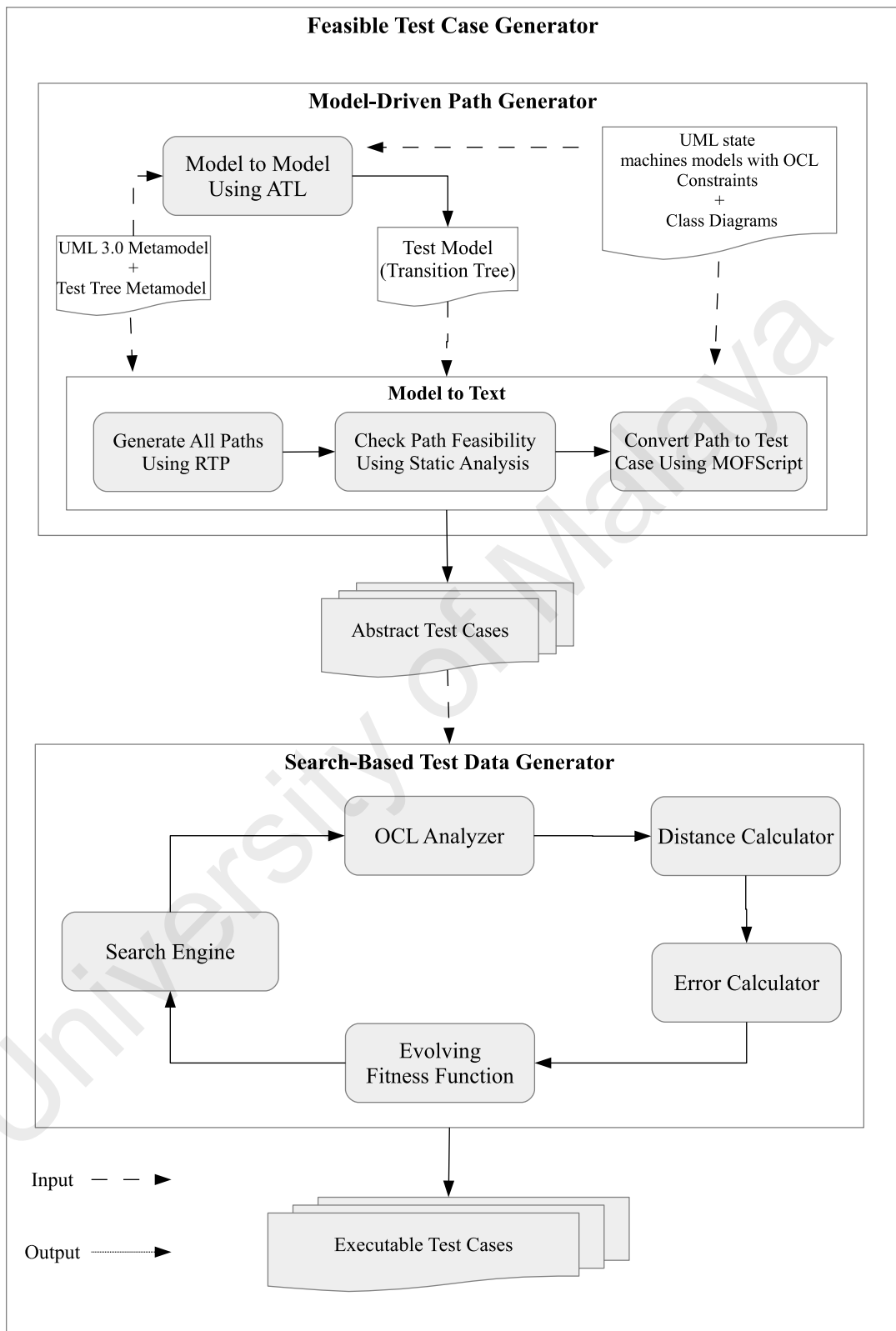


Figure 4.2: The proposed method.

path is then transformed into one test case.

Next subsection discusses the proposed infeasible path detection sub-component.

4.1.1.1 Infeasible Path Detection

To detect infeasible paths that contain conflict OCL constraints, it is necessary to define a set of rules of analyzing the paths using static analysis. Figure 4.3 presents the flow of the proposed infeasible path detection method for various OCL data type. The proposed infeasible path detection method is based on calculating penalty value of each clause in the path constraints.

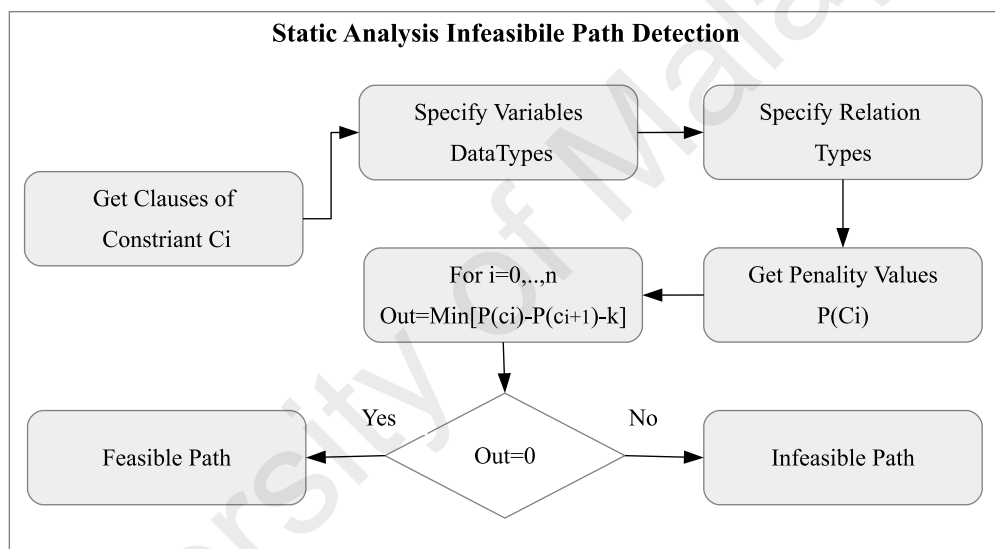


Figure 4.3: The proposed method of infeasible path detection.

Specifically, constraint consists of one or more clauses combined by boolean operator and each clauses contains decision variable and relation and/or constant and/or assigned variable. For example, a constraint $(A > 3 \text{ and } B = C)$ contains two clauses $(A > 3, B = C)$ combined by AND operator. Clause $A > 3$ contains A as decision variable, > as relation and 3 as constant while clause $B = C$ consists of B decision variable, = relation and C assigned variable. To determine the feasibility of a certain path, two cases are considered: 1) if constraint has only one clause or 2) constraint has more clauses with boolean operators. The feasibility metric is calculated based on the penalty value for each clause C_a in a

constraint within the path, and then, the feasibility metric between each two clauses is calculated using equation 4.1.

$$\min[||p(Caji) - p(Caji + 1)|| - k], \quad (4.1)$$

In the equation, the penalty value of each clause $p(Ca)$ is determined based on the its relation and data types. The k value is one of the cases in table 4.1.1.1:

Table 4.1: Values of K

K=1	when same decision variables with same constant or same assigned variables.
K=2	when same decision variables with same constant and both > and =.
K=30	when same decision variables with different constants.
K=30	when different decision variables with different assigned variables.
K=30	when different decision variables with same assigned variables.
K=0	when same decision variables with different constants and =operator.
K=40	other cases

Other values of K can be determined and one should be the difference between two conflicted operations or relations. If the minimum value of the feasibility metric moves toward zero, the path is feasible. If the returned value of the feasibility metric is greater than zero, this means there is a conflict relations in the path. The detailed pseudo code of the proposed infeasible path detection is shown in Algorithm 2.

OCL has various data types, including primitive, collection related, tuples, and enumerations. The penalty values of the relations of these different OCL data types are presented in next subsections.

Primitive Types OCL includes four primitive data types, Integer, Real, String and Boolean. The Integer type represents a set of integer values, the Real type is a set of real numbers, Boolean holds either true or false, and String contains strings over an alphabet. Each of these data types contains undefined special value. This special value is utilized for three purposes: 1) undefined value represents the error of evaluating an expression

Algorithm 2: Pseudo code of detecting infeasible paths

1: Input : test case
2: Output: feasibility rate
3: Steps :
4: For $i=0$; $i <$ no. of constraints
5: For $j=i+1$; $j <$ no. of constraints 6: Get all clauses of constraints i, j
7: While (constraint i has clause h)
8: Determine decision variable, relation, assigned variable, and constant for each clause ca_h
9: Calculate the penalty value of ca_h based on the relation type
10: While (constraint j has clause f)
11: Determine decision variable, relation, assigned variable, and constant for each clause ca_f
12: Get the penalty value of ca_f based on the relation type
13: Get the value of k based on the clauses ca_h and ca_f
14: Calculate $FR[hf] = |p(ca_h) - p(ca_f)| - k$
15: End of while 2
16: End of while 1
17: Get the minimum value in FR
18: if $Min(FR) = 0$
19: then stop
20: else go to step 5
21: Return $Min(FR)$
22: End of For 2
23: End of For 1
24: End

such as division by zero, 2) attributes are assigned to undefined value if their current value are not assigned yet, and 3) attribute assigned to undefined value if the attributes have not any values for specific instance. In the context of this thesis, the first case is relevant and considered.

For Real and Integer data types, the relations are $<$, $>$, $=$, $<=$, $>=$, and $<>$. For these relations, penalty values are assigned for each one as illustrated in Table 4.2.

Table 4.2: The penalty value of the basic relations.

Relation	Penalty value($p(o)$)
$>$	1
\geq	-1
$<$	2
\leq	-2
$=$	3
$<>$	4

For boolean data type, the relations in OCL and their estimated penalty values are presented in Table 4.3.

Table 4.3: The penalty value of the boolean relations.

Boolean Relation	Penalty value(p(o))
A	6
not(A)	7
A and B	$ p(a) - p(b) -k$
A or B	$ p(a)+ p(b) -k$
A implies B	(not A or B)
If A then B else C	(A and B) or (not A and c)
A xor B	(A and not B) or (not A and B)

Next, the penalty values of the complex data types will be discussed which include a collection-related data type, tuples, enumeration, and other miscellaneous.

Collection-Related Data Types The four collection data types in OCL, are Set, Ordered Set, Bag, and Sequence. OCL includes various operation on these types. For determining the feasibility of the paths contain operations of the collection data types, penalty value for each operation based on the returned value by the operation are proposed. In case returned value by the operation is primitive data type (Real, String, Boolean, and Integer), then the feasibility metric is calculated as numeric types (Integer and Real). For example, operation size() returns Integer value. For the other operations, they are categorized into three categories, 1) operations to check the equality of two collections, 2)operations to check the existing of item in a collection, and 3) operations to select subset items from a collection. Table 4.4 presents the penalty value for each operation of them and the discussion of how to calculate the feasibility metric of these operations.

Equality of operations: to compare two collections A1, A2, four cases can be found.

- 1- A1 and A2 are undefined,
- 2- A1 and A2 are not in the same kind.
- 3- A1 and A2 are the same kind and have different number of elements.
- 4-A1 and A2 are the same kind and have same number of elements.

Table 4.4: The penalty value of the operations of collection data type.

Relation	Cases	Penalty value(p(o))
Equality (A1=A2)	A1.ocIsUndefined() and A2.ocIsUndefined()	0
	Not (A1.ocIsKindOf(A2))	0
	(A1.size() != A2.size()) and (A1.ocIsKindOf(A2))	0
	(A1.size() = A2.size()) and (A1.ocIsKindOf(A2))	3
Non-Equality (A1<>A2)	A1.ocIsUndefined() and A2.ocIsUndefined()	0
	Not (A1.ocIsKindOf(A2))	0
	(A1.size() != A2.size()) and (A1.ocIsKindOf(A2))	0
	(A1.size() = A2.size()) and (A1.ocIsKindOf(A2))	4
Checking Existing	includes()	9
	excludes()	10
	isEmpty()	11
	notEmpty()	12
	includAll()	13
	excludAll()	14
	exists()	15
	isUnique()	17
	ForAll ()	19
	Select Subset	select()
reject()		22
collect()		23

Checking Existing: OCL includes various operations to check the existing of the item.

These operations return boolean value. The operations are include(), exclude(), isEmpty(), notEmpty(), includAll(), excludAll() and exist(), and isUnique()

Selection: OCL includes various operations to retrieve subset from the collected data .

The operations that make the path infeasible are select(), reject (), and collect().

Tuple Tuple in OCL is used to grouped various values together. Tuple includes different parts separated by a comma and each part specifies a value. For example: Tuple Name =Marc Bill, Age =50 contains a string Name with Integer Age. The value is accessed by its name such as Tuple Name =Marc Bill, Age =50. Age returns 50. In the feasibility analysis, the penalty values of primitive data types operation are used.

Special Cases: This sub-section describes how to calculate the penalty value of the other data types.

Enumeration: is a data type in OCL, which have a name and a set of literals. Enumeration is an objects which no specific order relation. The equality comparison between enumerations are treated as the collection related type.

Miscellaneous Operations: OCL includes various operations and returns Boolean value.

These operations are `oclIsTypeOf()`, `oclIsKindOf()`, `oclIsNew()`, `oclIsUndefined()`, and `oclIsInvalid()`. These operations are treated as Boolean type.

For the other operations on the complex data types, the penalty value for all of them is set to 40 because they do not conflicted with each other if they found in the same path.

4.1.2 Search-based Test Data Generator

For generating optimal test data for satisfying whole OCL constraints in a test case, a fitness function that evolves itself by using error feedback and a method for analyzing the whole constraints are proposed.

In the proposed search-based test data generator, the data is generated to satisfy whole constraints in each generated abstract test case. The OCL constraints in a test case are analyzed by the proposed OCL analyzer to get the dependency between the constraints clauses and the variables. All the clauses related to one variable are gathered and combined by AND operator to be new constraint associated with certain variable. The new constraints of all variables are passed to the search engine for generating the data based on the proposed fitness function. The fitness function calculates the distance of the new constraint to lead SBTs to generate data to satisfy all the constraints. The fitness function evolves itself based on error feedback in order to improve the performance of SBTs.

Next subsections discuss the two proposed sub-components of the proposed method: whole constraints analyzer, and fitness function.

4.1.2.1 Whole Constraints Analyzer

To generate data to satisfy all constraints in a test case, OCL constraints analyzers is proposed to statically analyze the constraints and find the dependency between them. In specific, all variables in the constraints of a test case are gathered and the dependency of the constraints based on the variables are calculated. For generating data to satisfy the whole constraints in a test case, Figure 4.4 illustrates the proposed method to analyze the constraints and find the dependency. In the proposed method, all the variables in the constraints are firstly gathered and get all the clauses in the constraints that related to each variable. The clauses of each variable are connected by AND operator separately. This connected clauses is considered as new constraint of the variable which needs to be solved by the proposed fitness function. The generated data solves each variable based on the its related clauses. To check the success rate of the generated data, the OCL constraint evaluator is used which replaces the variables by their generated data in the original constraints and check if the result is true or not. The pseudo code is described in Algorithm 3.

Algorithm 3: Pseudo code of Optimizing Whole Constraints

- 1: Input : test case constraints
 - 2: Output: set of constraints
 - 3: Steps :
 - 4: Get all variables in the constraints
 - 5: For each variable v
 - 6: Analyze constraints clauses
 - 7: Get clauses related on variable v
 - 8: Consider new constraint c by connecting the clauses of variable v by AND operator
 - 9: Generate data based on the new constraint C
 - 10: Evaluate the generated data on the original constraints
 - 11: End For
 - 12: End
-

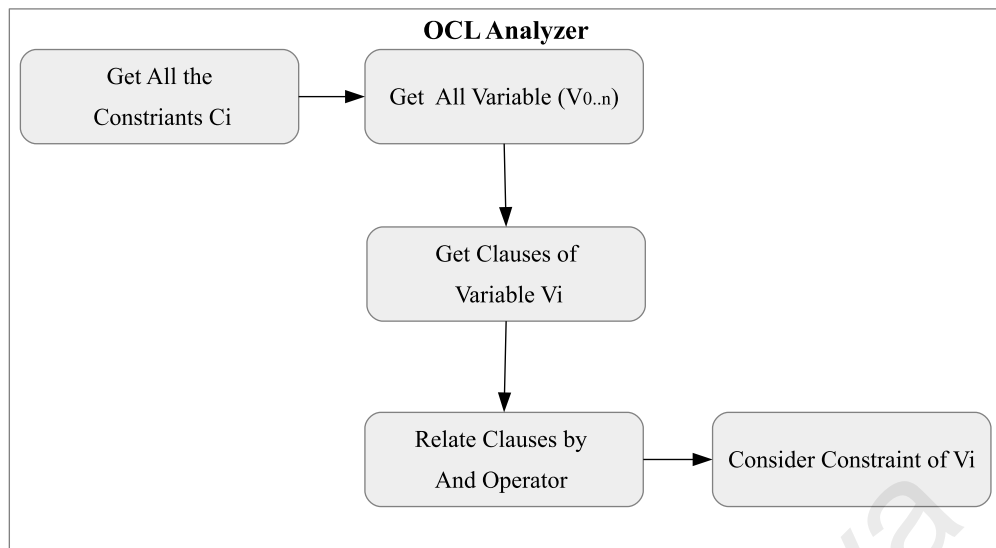


Figure 4.4: The proposed method for optimizing whole constraints.

4.1.2.2 Fitness Function

To guide the search for test data satisfying OCL constraints, it is necessary to define a fitness function of the SBTs. A fitness function indicates how far input data are from satisfying the constraint. For example, let us say values for variable y are needed to satisfy the constraint $y=10$, and suppose three data inputs are given: $y_1 = 1$, $y_2= 12$ and $y_3 =1000$. All inputs do not satisfy constraint, but y_2 is heuristically closer to satisfy $y = 10$ more than others. A search technique utilizes a fitness function to reward input data that is closer to satisfy the target constraint. In this thesis, the fitness function is proposed which inspired from neural network backpropagation algorithm and from work in (Ali et al., 2013, 2015)

In the proposed method, the distance ($d()$) is calculated based on the data type. The function $d()$ returns a value 0 if the constraint is solved, and otherwise a positive value that heuristically estimates how far the constraint was from being evaluated to true. The fitness function calculates the value of the fitness based on the relation or operation in the clause.

The following discusses the distance functions for different types of clauses in OCL.

Primitive Type: For Boolean data type, The distance of a Boolean variable A ($d(A)$) is one of three possible values: $d(A)=0$ if A is true, $0<d(A)<k$ if A is false, and $d(A)=k$ if A is undefined, where k is an arbitrary positive constant. When a Boolean variable A is obtained from an operation call, then the distance is one of these three possible values.

The operations defined in OCL to combine Boolean clauses are or, xor, and, not, if then else, and implies. The branch distances of these operations are adopted from (Ali et al., 2013) and shown in Table 4.5. For example, for two clauses a and b, the branch distance is calculated as follows:

$$d(A \text{ and } B) = \text{numberOfUndefinedClauses} + \text{nor}(d(A) + d(B)) \quad (4.2)$$

Table 4.5: The distance calculation of boolean operations.

Boolean Operation	Distance calculation
A	If A is true then $d(A)=0$ else A is false then $d(A) >0$ and $d(A)<k$ else $d(A)=k$
not(A)	If A is false then $d(A)=0$ else A is true then $d(A) >0$ and $d(A)<k$ else $d(A)=k$
A and B	$\text{numberOfUndefinedClauses} + \text{nor}(d(A)+d(B))$
A or B	$\text{numberOfUndefinedClauses} + \text{nor}(\min(d(A),d(B)))$
A implies B	(not A or B)
If A then B else C	(A and B) or (not A and c)
A xor B	(A and not B) or (not A and B)

For numerical types (Integer and Real), their relational operations that return Booleans are $<$, $>$, $<=$, $>=$, $=$ and $<>$. For these operations, the distance between variables A and B with their weights w_A and w_B is calculated as described in equation 4.3.

$$d(A, B) = \text{abs}(A * w_A - B * w_B) \quad (4.3)$$

The fitness function then evolves the value of the weight w_A and w_B based on input value A and B and the calculating error e as shown in equation 4.4.

$$w_A = w_A + A * e_A, w_B = w_B + B * e_B \quad (4.4)$$

The error is calculated based on the distance between the optimal input data and the generated data as illustrated in equation 4.5. The optimal data in this case is the data that satisfy the constraint.

$$e_A = x_{optimal} - A, e_B = B_{optimal} - B \quad (4.5)$$

The fitness value of the distance is then calculated according to (Ali et al., 2013) as shown in Table 4.6.

Table 4.6: The fitness calculation of the basic numerical relations.

Relation	Fitness calculation(d())
$A > B$	if not A.oclIsUndefined and not B.oclIsUndefined and d()=0 then 0 else if not A.oclIsUndefined and not B.oclIsUndefined and d()<>0 then k* nor(d()+1) else k
$A \geq B$	if not A.oclIsUndefined and not B.oclIsUndefined and d()=0 then 0 else if not A.oclIsUndefined and not B.oclIsUndefined and d()<>0 then k* nor(d()+1) else k
$A < B$	if not A.oclIsUndefined and not B.oclIsUndefined and d()=0 then 0 else if not A.oclIsUndefined and not B.oclIsUndefined and d()<>0 then k* nor(d()+1) else k
$A \leq B$	if not A.oclIsUndefined and not B.oclIsUndefined and d()=0 then 0 else if not A.oclIsUndefined and not B.oclIsUndefined and d()<>0 then k* nor(d()+1) else k
$A=B$	if not A.oclIsUndefined and not B.oclIsUndefined and d()=0 then 0 else if not A.oclIsUndefined and not B.oclIsUndefined and d()<>0 then k* nor(d()) else k
$A <> B$	if not A.oclIsUndefined and not B.oclIsUndefined and d()=0 then 0 else if not A.oclIsUndefined and not B.oclIsUndefined and d()<>0 then k* nor(d()+1) else k

Other operations are defined in OCL on Real and Integer such as +, -, *, /, abs(), div(), mod(), max(), and min(). Since these operations are not utilized to compare

two numerical values in clauses, a distance for them is not defined in this study. For instance, considering x and y are two Integer variables and a constraint c is $x-y+10 > 3$, then the operation $-$ and $+$ are used only to define the constraint c . The overall result of the constraint c will be an Integer and the clause will be considered as a comparison of two Integer values.

For the String type, OCL defines several operations such as $=$, $+$, $\text{size}()$, $\text{concat}()$, $\text{substring}()$, and $\text{to Integer}()$. There are only three operations that return a Boolean: equality operator $=$, inequality $<>$, and $\text{equalsIgnoreCase}()$. In these cases, instead of using k if the comparisons are false, the value from any string matching distance function is returned to evaluate how close any two strings are. The distance function of (Mcminn, Shahbaz, Stevenson, & Court, 2006) is adopted for String type.

Complex Type: includes collection related data types (Set, OrderedSet, Bag, and Sequence), tuples, and enumerations, these types are described in Section 4.1.1.1. OCL defines various operations on collection related data types. These operations return several data types and this section discuss how to calculate the distance for each operation. First type of operations return numerical data types and the return value of an operation is used in an expression, then the distance is calculated in the same way as for primitive types as defined in aforementioned section (primitive types). For example, $\text{size}()$ operation, which returns an Integer. The distance calculation of Integer data type for $\text{size}()$ operation is used. For second type of operations that return Boolean, the distance calculation is adopted from studies (Ali et al., 2013, 2015). These operations include the equality ($=$ and $<>$), checking the existing ($\text{includes}()$, $\text{excludes}()$, $\text{isEmpty}()$ and $\text{notEmpty}()$), iterators ($\text{forAll}()$, $\text{exists}()$, $\text{one}()$ and $\text{isUnique}()$), and selection of subset ($\text{select}()$, $\text{reject}()$ and $\text{collect}()$).

For tuples, there are no operations on tuples in OCL because they are not subtypes of OCLAny. However, when a value in a tuple is accessed and compared, a branch distance is calculated based on the type of the value and the comparison operation used. For enumerations, the equality operations are treated as Boolean data type in calculating distance. Regarding the user defined operations, a branch distance is calculated according to the return types of these operations.

4.2 Significance of the Proposed Method

The proposed method has several significant features that are described as follows.

Feasibility: The most significant feature of the proposed method is that the generated test cases are feasible and all the generated data able to satisfy all the OCL constraints in the test case at least one time. This feature was demonstrated by detecting the infeasible paths in chapter 4 (section 4.1.1.1).

Robustness: One of the significant features of the proposed method is the robustness of the generated test cases which obtains from solving complex OCL constraints on the properties of the systems emulating faulty situations. These constraints were solved at the same time by the proposed search-based test data generator 4.1.2 during test case generation in order to set the system properties in such a way as to trigger faulty situations. This feature was demonstrated by the proposed search-based test data generator in chapter 4.

Extensible: One of the significant feature of the proposed method is the extensibility of the method for various contexts which inherits from applying model transformations. In specific, the proposed method can be extensible to various UML models, various output scripting languages such as C++, test models such as testing flow graph, and coverage criteria such as all transition for different application domains and systems.

These are examples of useful extensions for the proposed method. This feature was demonstrated by using ATL model-to-model transformation and MOFScript model-to-text transformation. These transformations are based on developed rules which need metamodels as inputs. Therefore, to extend to other contexts, new rules and metamodels can be added.

Configurable: The proposed method is easily configurable to satisfy varying requirements such as input model, coverage criteria and scripting language. High configurability enables testers to experiment with various techniques without significant effort and changes in the implementation. This is of practical importance as different test models, coverage criteria, and scripting language helps in targeting different types of faults. This feature is because the method was implemented by Eclipse and MDE and can be extensible for different contexts.

Coverage: Path coverage (includes transitions and states) measures the number of paths traversed during the generation process as compared to the actual number of paths that exist in the subject. High coverage was achieved by the proposed method, where in all aspects of the state models were sufficiently considered in the generation process. Thus, test cases that have a good coverage statistic were generated as presented in the results in chapter 3 and 5 (100 percent state and transition coverage).

Independence: The proposed method generated test cases with no interaction between them, in which there is no test case depends on the output of other test cases. The test cases were run in any order without having any impact on the overall evaluation of the test suite. This is because each path was transformed as one abstract test case (java file) by the MOFScript language as shown in section 5.2.2.

Costless: The proposed method was implemented and developed with open source lan-

guages and softwares. Eclipse with its modeling plugins were used for developing test case generation and open source library for search-based test data generator. This free cost of developing reduced the cost of the entire testing cost.

Easy to use: The proposed method is easy to use because it is configurable and flexible enough to cater to a variety of different contexts. Different systems have various testing needs and the proposed method is able to be adapted to different testing processes.

4.3 Conclusion

This chapter illustrates the proposed method in this research and described its characteristics. The schematic presentation of the proposed method and its major components are depicted. also described the proposed method by explaining infeasible path detection, whole constraints analyzing, and fitness function evolves itself. Moreover, several significant features of the method are highlighted in details. In next chapter, empirical evaluation of the proposed method using three industrial case studies is presented .

CHAPTER 5: EVALUATION OF THE PROPOSED METHOD

Chapter 4 described the proposed method that generate feasible test cases from UML state machine models with OCL constraints. In this chapter, empirically evaluation was conducted for the proposed method within the context of industrial systems. For this purpose, the empirical case study method was adopted which used in (B. A. Kitchenham et al., 2002; Perry et al., 2004) to analyze the cost and effectiveness using three metrics. The case study was conducted within the context of using three industrial public case studies which were taken from (Peleska et al., 2011; Frías, Queralt, & Ramon, 2003). This chapter also discusses the results of the empirically evaluation of the proposed method compared to the existing methods. The effectiveness and cost of the infeasible path detection and search-based test data generator were presented, analyzed and synthesized for the three selected industrial case studies. Finally, the evaluation results were compared with the results of the existing method.

The outline of this chapter remainder is presented in Figure 5.1. Section 5.1 presents the case study method. Section 5.2 describes how the case study is executed. The results of the proposed method are presented and evaluated in section 5.3. The results of comparison between the proposed method's results with the existing methods are presented in section 5.3.3. Overall discussion and statistical testing results are presented in section 5.3.4. Section 5.5 concludes the chapter.

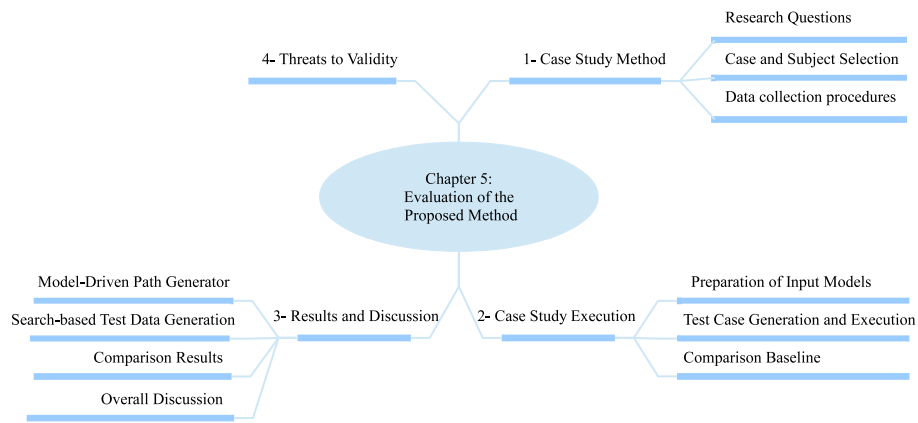


Figure 5.1: Semantic presentation of chapter 5 outline.

5.1 Case Study Method

This evaluation used similar case study method described in section 3.1.

5.1.1 Research Questions for the proposed method

The main first step in the empirical case study evaluation method is to construct research question that fulfilled the evaluation purpose. In this evaluation, the same terms of evaluation criteria (cost and effectiveness) in chapter 3 were considered in this chapter. Chapter 3 has evaluated the performance of model-driven path generator, Kalaji approach and EsOCL solver. However, this chapter focuses on evaluating the performance of the proposed method (infeasible path detection and search-based test data generator), because of the same model-driven path in chapter3 was also used in the proposed method. Therefore, two questions were derived to embody the evaluation objective:

RQ 5.1 : What is the effectiveness of the proposed infeasible paths detection method?

RQ 5.2 : What is the cost and the effectiveness of the proposed search-based test data generator?

5.1.2 Case and Subject Selection for the Proposed Method

For the same reason discussed in section 3.1.2, this evaluation used real-world complex industrial systems to ensure relevancy of the proposed method results. The two of these systems (TIS and CSM) was revisited, which have been described in chapter 3 (section 3.1.2) and include the third system, European car rental (Frías et al., 2003) in this evaluation.

These three case studies are publicly available as the benchmark of real time embedded systems (Peleska et al., 2011; Frías et al., 2003). Their models include UML state machines, class diagrams, and constraints.

5.1.2.1 Case Study 1: Turn Indicator

This case study is described in section 3.1.2 in chapter 3.

5.1.2.2 Case Study 2: Ceiling Speed Monitoring

This case study is described in section 3.1.2 in chapter 3.

5.1.2.3 Case Study 3: Eu-rent

The case study is a widely known as publicly available case study that models a European Car Rental System with over 1000 branches in different countries. The case EU-Rent specifications was initially developed by Model Systems, Ltd (Wilson, 1994) and then extended by Frias et al. (2003). EU-Rent Specifications is considered as a modeling benchmark case study and has been used in multiple studies for evaluation and demonstration (Ali et al., 2015; Cabot & Gogolla, 2012). The specifications of the case study are developed using UML3.0 diagrams and OCL2.0 constraints. In total there are thirty-four OCL constraints in the case study in three state machine models. The constraints include a number of constructs including nested conjunctions, disjunctions, and implies. This allow us to evaluate most of the proposed improved method.

5.1.3 Data collection procedures for the proposed method

This chapter focuses on evaluating the proposed method (infeasible path detection and test data generator). Therefore, four metrics were applied from the seven metrics that presented in section 3.1.3 to measure the cost and effectiveness of the proposed method.

The cost was measured in terms of:

- Time of Generation: The time spent on generating data for the test case.
- Size of Test-suite: The number of generated test cases in terms of feasible and infeasible paths.

Effectiveness was measured by:

- Detection rate: The number of detected infeasible paths in the generated test suites.
- Success rate: The number of times the proposed data generator method is successfully obtaining a solution out of the total number of runs. In this study, success rate in solving all constraints in each feasible test case.

To note that, the procedure is similar to the one presented in chapter 3. Timing data is collected by running the experiment on a Windows 7 64-bit operating system, machine with an Intel(R) Core(TM) i5 CPU @ 3.4 GHz processor, and with 8 GB memory. The time is measured in seconds.

5.2 Case Study Execution for the proposed method

This section describes the main activities in evaluating the proposed method, including the preparation of input models, developing various components of the proposed method, and the implementation of model-driven approach for test cases generation and execution.

This steps are similar to the evaluation executing in section 3.2.

5.2.1 Preparation of Input Models

The state machine models of the three case studies were modeled using UML 3.0 and OCL 2.0 by Papyrus eclipse plugin. It has been mentioned (in section 3.2.1) that one state machine model in the TIS case study was flattened manually and this also has been similarly applied on one state machine model in Eu-rent case study because the flattening part of the model-driven approach was not implemented. The summary of the three case studies are shown in Table 5.1. Note that the functionality as originally modeled has not been affected by the remodeling. The constraints in the all case studies were written in

Table 5.1: Case studies description.

State-machine feature	CSM case study	TIS case study	Eu-rent case study
Number of State Machines	3	39	3
Number of submachine	-	4	5
Number of simple states	13	118	29
Number of transitions	17	164	34
Number of constraints	13	119	34

OCL language. As discussed in section 3.2.1, the number of guards in CSM case study are 13 while guards in TIS case study are 119. The number of clauses in each guard varies from one to four in TIS and CSM systems. The data type of the variables used in the constraints are primitive types (real, integer, and boolean) and enumerations. Eu-rent case study includes 34 complex OCL constraints with 1 to 20 clauses. The constraints contains a number of constructs compressing nested conjunctions, disjunctions, and implies.

5.2.2 Test Case Generation and Execution

The proposed method was implemented using java JDK 1.7, UML 3.0 and Eclipse. For the proposed abstract test case generation (path generator), the model-to model transformation was implemented using ATL language and the model-to text transformation was developed using MOFScript language same as described in section 3.2.4. The difference between the work in chapter 3 and this chapter is that the the implementation of the proposed infeasible

path detection is inside the model-to-text component. The two required metamodels are core files same as files described in section 3.2.4, in which the transition tree metamodel was taken from (Ali, Hemmati, et al., 2010) and modeled using EMF and the UML 3.0 metamodel is provided by EMF. The output of model-to model transformation is (transition tree) in XML file, while the output of model-to text transformation is a set of java files (test cases) and each java file is one feasible abstract test case.

For the proposed search-based test data generator, Jmetal library, java and eclipse were used. Four SBTs were applied with the proposed method to generalize the proposed method with the common SBTs while two SBTs were applied in section 3.2.4, including i) quantum GA (QGA), ii) GA, iii) EA, and v) SA. These SBTs were selected as representative of algorithms belonging to different categories. SA was selected as a representative of local SBTs. GA was selected since it is the most commonly used global search algorithm in SBSE and can be used as a representative of global SBTs. EA is simpler than GA, it was found that it can be more effective for software testing problems (Ali, Briand, et al., 2010). QGA is chosen due to quantum theory with global SBTs is an efficient technique in the problems with large data size like software engineering problems as reported in (Kumari, Srinivas, & Gupta, 2013; Jin & Jin, 2016). The other SBTs have been increasingly being used in SBSE, such as particle swarm optimization, that may be interesting to be applied in the future work. The configuration of the proposed data generator is shown in Table 5.2 similar to chapter 3. From the table, the population size was set to 1000 and the crossover rate is determined as 0.75, with a 1.5 bias for rank selection. A standard one-point crossover was used, and mutation of a variable was done with the standard probability $1/n$, where n is the number of variables. Different settings would lead to different performance of a search algorithm, while standard settings always perform well (Arcuri & Fraser, 2013).

To compare the SBTs in search-based test data generator, OCL evaluator was used as similar to the evaluator used in section 3.2.4 to check if the generated data satisfy the constraint or not. OCL evaluator EyeOCL Software was used. To use OCL evaluator, the class and object diagrams were constructed from the SUT class diagram called OCLWrapper. To automate the build, execution, and time data collection, a batch file was created to contain all the generated test cases, the OCLWrapper, and test data generator. In the execution, EyeOCL was used at runtime to check the constraints in the test case script and then each test case was invoked on a new instance of the SUT.

For the performance evaluation, statistical test was used to assess randomized test strategies. First, the success rate was calculated for each technique, which is defined as the number of times a solution was found out of the total number of runs (1000 iterations). These success rates were then compared using t-test at a significance level of 0.05(p value). In this study context, a value of p less than 0.05 tells that one technique is better than another in finding solutions in lesser number of iterations. A value of 0.05 or greater tells that there is no difference between techniques in finding solutions in lesser number of iterations.

Table 5.2: Configuration of search-based techniques.

Parameter	Value
Fitness evaluations	2000 times
Number of iterations	1000
Crossover rate	0.75
Population size	1000
Mutation	1/n, where n is the number of variables

5.2.3 Comparison Baseline

Based on the guidelines of conducting empirical case study in search-based testing in (Ali, Briand, et al., 2010), choosing a baseline and comparing the result of the proposed method with result of baseline is necessary to show the improvement of the proposed

method compared to the existing techniques. For this purpose, the method that were analyzed in chapter 3 were used as comparison baseline. Specifically, the recent infeasible detection method by Kalaji et al. (2011) was used as a comparison baseline with the proposed infeasible path detection method and the recent EsOCL solver was utilized which proposed by Ali et al. (2013) and improved in (Ali et al., 2015) with four SBTs (GA, SA, EV, and QGA) as a baseline for the proposed search-based test data generator.

5.3 Empirical Evaluation Results of the Proposed Method

The results of the model-driven method in terms of state and transition coverage criteria have been presented in section 3.3.1 for the CSM and TIS case studies. The state and transition coverages are 100 percent for the Eu-Rent case study. Therefore, this section focus on the infeasible path detection and search-based test data generator results. The results are presented with the help of descriptive statistics, tables, and charts. These results were used to evaluate the proposed method with comparison baselines infeasible path detection (Kalaji approach) and search-based test data generation (EsOCL solver).

5.3.1 Infeasible Path Detection

The RQ 5.1 is addressed in this section. The effectiveness metric of infeasible paths detection is detection rate. Figure 5.2 shows the results of the proposed infeasible path detection method in the three case studies. The infeasible path detection method achieved 100 percent detection rate in the CSM and TIS while achieved 90 percent for the Eu-Rent case study. CSM and TIS case studies contains integer, boolean and enumeration data types with 4 maximum constraint clauses while the Eu-Rent case study contains all the primitive data types and complex data types (enumerations, tuples, and Set) with 20 maximum clauses. The proposed infeasible path detection is significantly efficient in detecting infeasible paths that contain different types of data and operations.

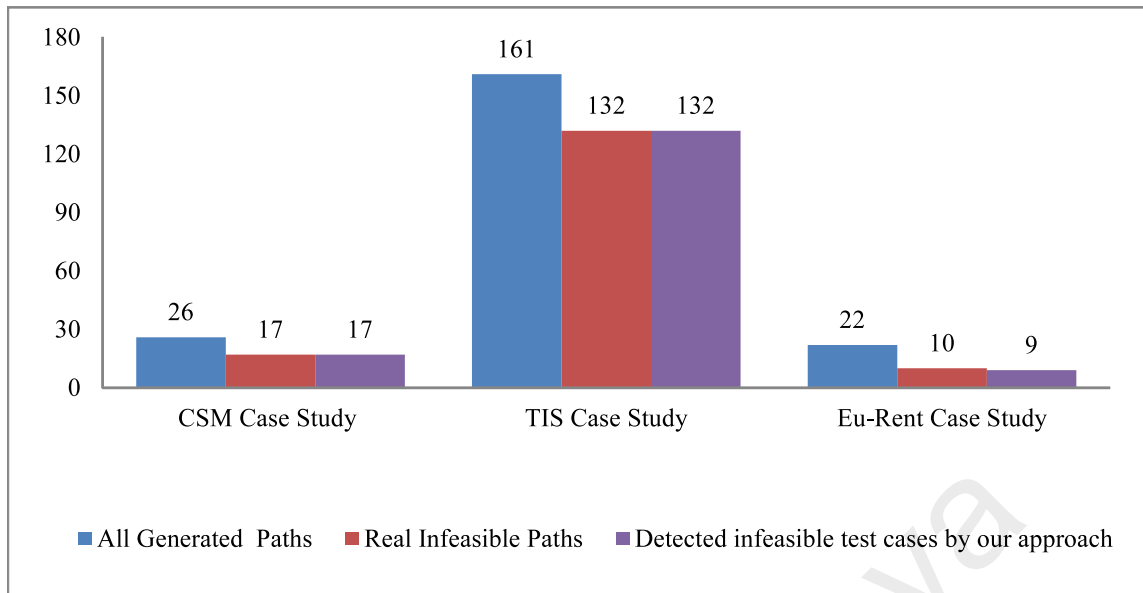


Figure 5.2: The results of the proposed infeasible path detection method.

5.3.2 Search-based Test Data Generator

This section discusses the results of the proposed test data generator within three case study. In term of the effectiveness criterion, Figures 5.3, 5.4, and 5.5 illustrate the results of the proposed test data generator in the CSM, TIS and Eu-Rent case studies respectively using four SBTs. From the three figures, the proposed method is efficient in satisfying all the constraints in each test cases with the four SBTs in the three case studies.

Figure 5.3 shows that the proposed method with SA obtained the best results while the proposed method with QGA obtained the lowest success rate. GA and EA with the proposed test data generator approximately performed the same success rate except in 7 and 8 test cases. GA and QGA obtained roughly same success rate in test case 8. Overall, the proposed method with four SBTs achieved high success rates in the CSM case study.

It has been concluded from Figure 5.4 that 100 percent success rate obtained from the four SBTs in the first six test cases and test cases number 20 and 21 and then the techniques are wavering. The equal or lowest performance achieved by QGA in all the test cases. Generally, GA, SA, and EA achieved approximately same success rate and

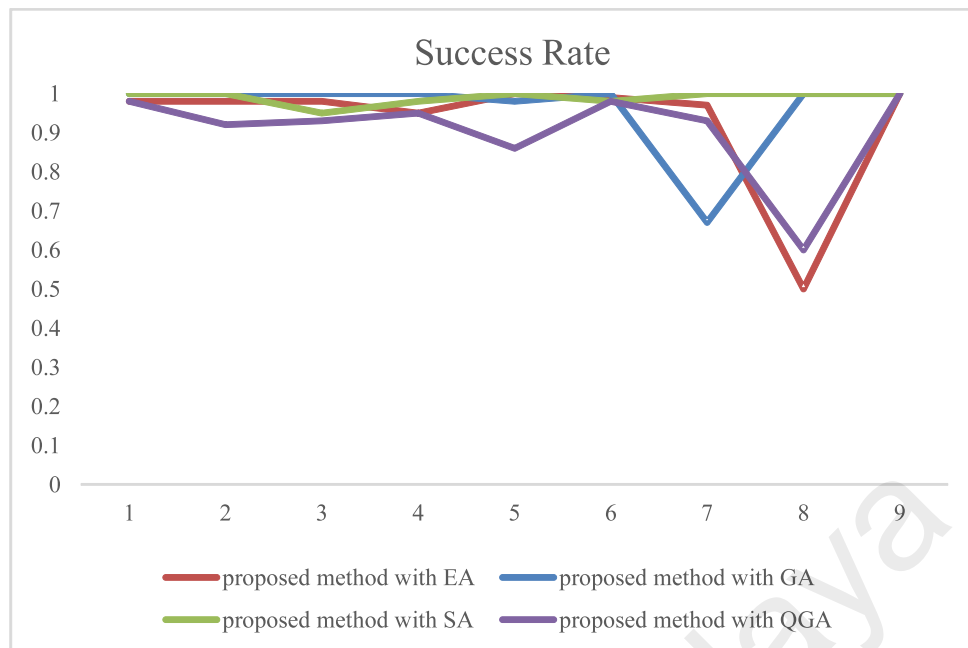


Figure 5.3: The results of success rate of the proposed method with GA, EA, SA, and QGA techniques in case study 1 (CSM).

outperformed QGA in most test cases of TIS case study.

From Figure 5.5, QGA obtained the best success rates in most test cases(3-12) while SA obtained the worst success rates. EA and GA approximately achieved the same success rates in all the test cases. In specific, test case 9 got the worst success rate by all the techniques followed by both test cases 3 and 10 due to all constraints of this test case are very complex which contain from up to 21 clauses. Data was successfully generated to satisfy all the constraints in each test case 4-8. To conclude, QGA outperformed the others techniques in EU-Rent case study.

In term of the cost evaluation criterion, Tables 5.3, 5.4, and 5.5 illustrate the time taken to generate data in the three case studies respectively. From Table 5.3 of CSM, test case 6 took more time to generate data by the four SBTs while less time was taken for test case 9 with four SBTs. The proposed method with SA generated data with more time than others SBTs in test cases 2, 5-7, and 9, while GA generated data with less time in all test case 3. Comparing between SBTs, GA and SA took roughly same generation time in test

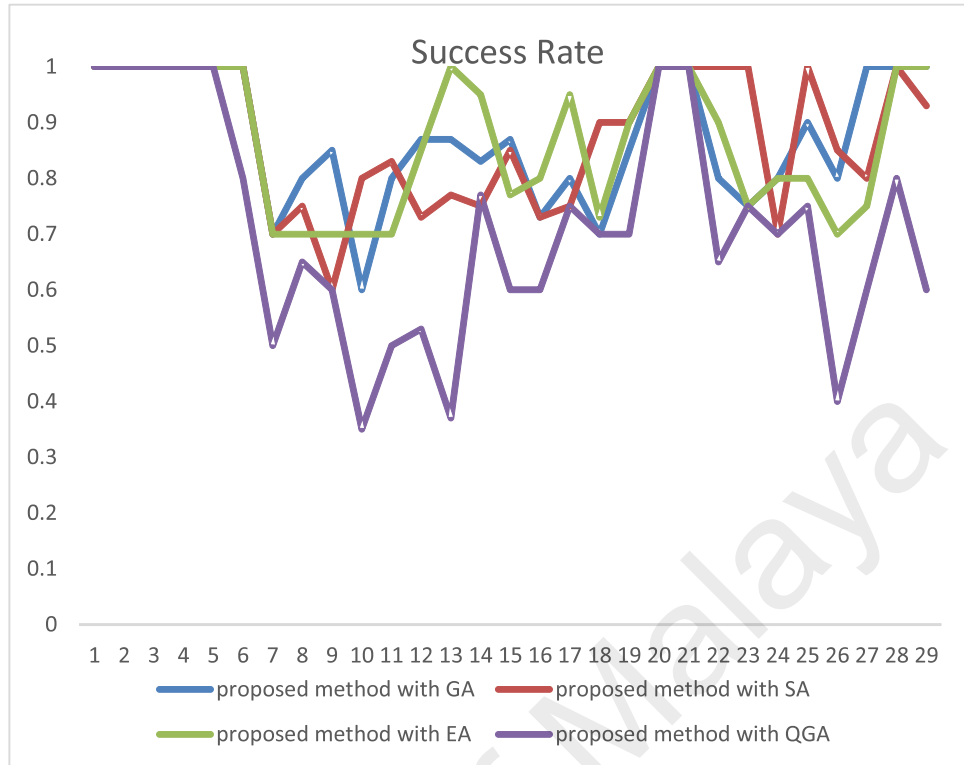


Figure 5.4: The results of success rate of the proposed method with GA, EA, SA, and QGA techniques in case study 2 (TIS).

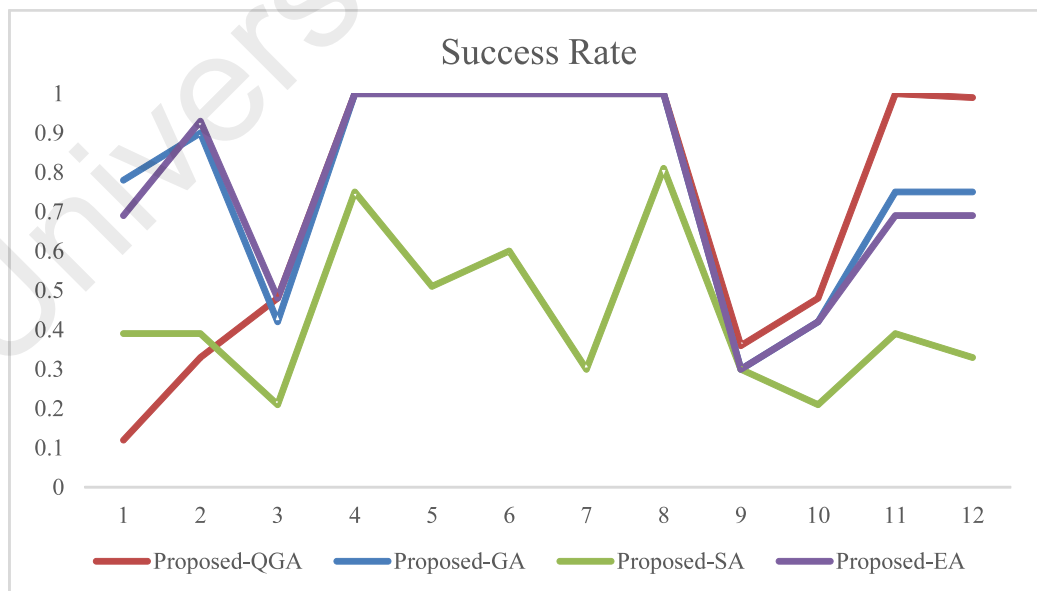


Figure 5.5: The results of success rate of the proposed method with GA, EA, SA, and QGA techniques in case study 3 (EU-Rent).

case 1 whereas GA and QGA generated data for test cases 4 and 9 within almost similar time. In general, local SBTs (SA) needed more time to generate data in most test cases. The taken time by GA is wavering with EA and QGA. Relating cost and effectiveness criteria, it has been observed that GA achieved best success rates with less time in most of test cases in CSM case study. Furthermore, SA achieved roughly same success rate of GA within more time. QGA obtained the lowest success rate within moderate generation time. The reason of this is the low number of global optima in this case study search space.

Table 5.3: The results of generation time of the proposed method with GA, EA, SA, and QGA techniques in case study 1 (CSM).

TC ID	CSM		Generation Time			
	constraints	number	GA	EA	SA	QGA
1	4		146.12	147.81	138.05	154.59
2	5		174.29	184.5	178.51	181.73
3	4		207.43	164.11	152.4	165.33
4	4		149.38	157.11	165.87	150.34
5	5		179.12	196.64	182.16	185.11
6	9		297.94	328.93	307.41	308.95
7	6		150.47	215.72	205.53	213.76
8	2		29.59	35.14	37.57	32.26
9	1		4.67	7.59	6.74	3.54

With respect to TIS case study, Table 5.4 shows that the 1-6 test cases, test cases 20-23 and test case 28 were taken less time than others by all SBTs due to number of constraints which is either 1 or 2 with simple number of clauses (containing 1, 2, and 3). The generation time by the four SBTs was approximately similar in each test case 6-9, and test cases 12,13,16, and 29. Test case 17 clearly cost less time by SA while test case 27 took significantly less time by GA. The taken time by four SBTs were almost similar in the other test cases. Comparing global SBTs, GA and EA almost generated data within the same time in each test case 1,3,5,9,10,13,15-17,20-21,23, and 29 whilst QGA took less time than GA and EA in test cases 2, and 23. Overall, there is no significant difference

between the cost of the four SBTs in the most test cases.

Respecting to cost and effectiveness in TIS case study, the test cases 1-5, 20, and 21 successfully satisfied with very low cost by the four SBTs. Generally, QGA achieved the lowest success rates with high time cost in most test cases. Moreover, the performance of the proposed method with GA, SA and EA roughly similar in most test cases. Specifically, 100 percent success rate was achieved with lowest cost was obtained by SA in test cases 23, by GA in test cases 29, and 27, and by EA in test case 25.

Table 5.4: The results of generation time of the proposed method with GA, EA, SA, and QGA techniques in case study 2 (TIS).

TC ID	TIS		Generation Time			
	constraints number		GA	SA	EA	QGA
1	1		0.4	0.9	0.53	1.24
2	1		6.15	5.09	1.343	0.672
3	2		5.374	12.54	5.925	5.668
4	1		5.02	5.06	12.18	6.73
5	2		5.624	12.67	5.725	6.307
6	1		9.82	11.18	11.56	10.48
7	2		224.42	220.87	231.1	232.26
8	2		91.85	95.54	97.75	97.21
9	2		112.75	115.91	113.59	118.49
10	2		185.34	156.62	188.24	194.03
11	2		168.25	189.06	157.16	162.12
12	3		188.88	186.06	194.37	197.37
13	3		193.63	190.67	196.75	198.69
14	3		187.59	200.94	194.52	210.02
15	2		189.04	194.06	190.9	200.53
16	2		191.16	195.07	190.62	197.65
17	2		189.6	60.29	190.86	203.14
18	2		195.87	179.5	188.18	197.7
19	2		47.82	18.41	58.46	49.716
20	1		10.72	12.3	12.49	10.93
21	1		10.48	12.05	12.49	11.08
22	2		14.11	19.26	17.01	51.63
23	2		18.986	12.18	17.05	15.01
24	2		174.71	160.6	189.9	149.8
25	2		172.53	160.1	158.54	197.26
26	2		164.61	184.2	161.46	164.04
27	2		37.15	115.99	160.25	163.22
28	1		10.63	11.95	39.02	38.51
29	4		15.4011	16.49	17.271	17.404

In Eu-Rent case study, EA generated data with less time in all test cases as compared to the other three SBTs as shown in Table 5.5. Furthermore, test cases 4-8 were needed less time to satisfy all their constraints than others. QGA, GA and SA were different in the taken generation time. Specifically, GA took less time than SA and QGA when the number of test case constraints are high as shown in the test cases 1,3,9-12, while SA needed less time when the number of test case constraints are small. This is because the search space of the test case with high number of constraints is large which GA works efficiently in searching optimal data in this large search space.

Combining cost and effectiveness criteria, QGA obtained the best success rates and SA obtained the worst success rates with various generation time. Test cases 4-8 took less time to full successfully generate data. The test case 9 took longer time while the success rates using four SBTs were low. QGA obtained highest success rate (100 percent) with highest cost in test cases 11-12 compared to GA,SA and EA. Generally, the complex constrains (containing many clauses) in the test cases needed more time and achieved less success rates by all SBTs, while the constraints with lowest clauses needed less time and got 100 percent success rates.

Table 5.5: The results of generation time of the proposed method with GA, EA, SA, and QGA techniques in case study 3 (EU-Rent).

TC ID	EU-Rent		Generation Time			
	No of constraints		GA	SA	EA	QGA
1	8		143.642	160.305	71.039	151.402
2	9		151.246	151.183	101.332	146.043
3	9		145.589	175.436	97.418	182.152
4	2		24.049	6.533	2.63	10.239
5	3		32.458	7.254	4.924	12.963
6	4		35.979	17.983	6.688	24.069
7	5		36.316	23.214	8.235	27.726
8	6		43.869	27.118	9.029	31.12
9	6		111.06	140.625	62.63	147.869
10	7		108.454	149.353	78.08	150.383
11	8		116.312	142.271	79.788	158.318
12	8		161.674	233.848	102.132	251.286

5.3.3 Comparison Results

This section presents the comparison results between the proposed infeasible path detection method with Kalaji approach and the proposed test data generator with EsOCL solver.

5.3.3.1 Infeasible Path Detection

Figure 5.6 presents the comparison result between the proposed infeasible path detection method and infeasible path detection method by Kalaji (A. S. Kalaji et al., 2011). From the figure, the proposed method is superior than Kalaji approach. The proposed method achieved 100 percent in both CSM and TIS case studies while Kalaji approach detected only 29 and 13 percent of the infeasible paths in them. Non of the infeasible paths in the Eu-Rent case study has been detected by Kalaji approach while 90 percent of them have been detected by the proposed approach. The reason of the low detection rate of Kalaji approach in the three case studies is that Kalaji approach covered only basic relations of integer data type while the OCL constraints in the case studies include other complex data types with their operations. The proposed infeasible path detection method detected almost all the infeasible paths in the three case studies because all the OCL data types and their operations are considered.

5.3.3.2 Search-based Test data Generation

Tables 5.6, 5.7, and 5.8 show the comparison between the results of the proposed test data generator with the results of EsOCL solver with three industrial case studies in term of effectiveness. Generally, the proposed method significantly outperformed the EsOCL and improved the performance of the SBTs in generating test data that satisfy all constraints in each test cases in the three case studies. Furthermore, the proposed method with all SBTs achieved the same success rate (100 percent) as EsOCL solver in the test cases numbers 9 in CSM and 2-5 in TIS, while the proposed method only achieved 100 percent in Eu-Rent

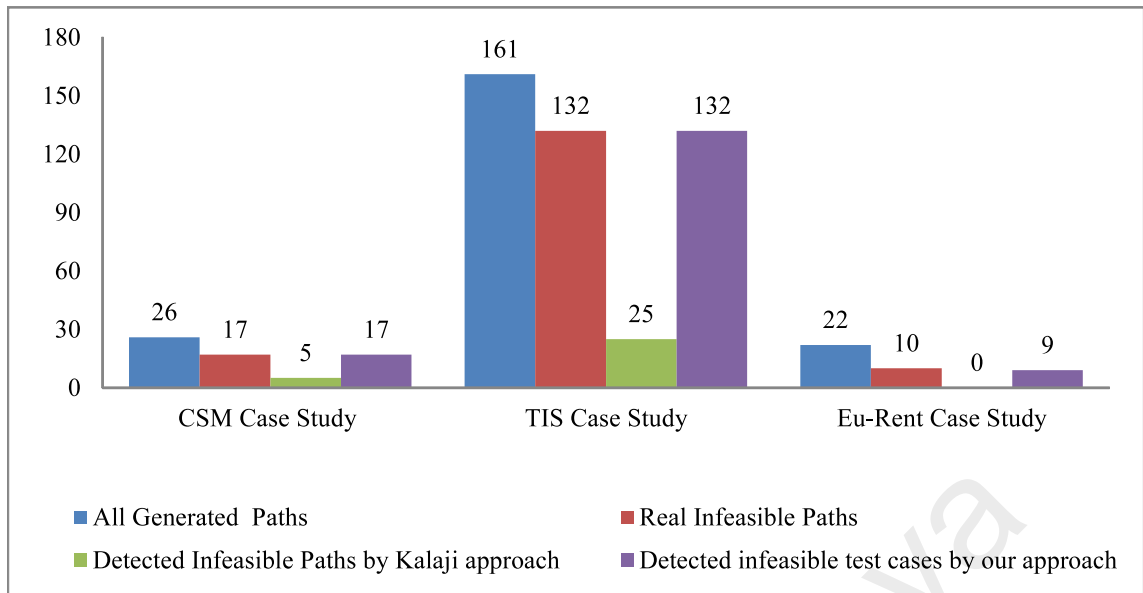


Figure 5.6: The results of the proposed infeasible path detection method with Kalaji approach.

case study. The Superior performance of the proposed method is due to the proposed method optimized each variable separately to satisfy all the constraints at the same time compared to EsOCL solver which optimized the variables of one constraint at each time. In CSM case study, both SA and GA with the proposed method achieved the best success rates whereas SA with EsOCL solver achieved the best success rates. In TIS case study, QGA achieved the worst success rates with longer generation time with both the proposed method and EsOCL solver. In Eu-Rent case study, the proposed method with QGA was superior than others techniques (GA, SA and EA) while QGA with EsOCL solver did not perform well. Furthermore, SA with the proposed method achieved the least success rates but it performed better with EsOCL solver. The difference in the performance of SBTs with the proposed method and EsOCL solver is due to that the proposed OCL analyzer simplifies the search space of case studies. When the search space contains few local optima, the global SBTs (GA, EA, and QGA) perform well in finding the best solution.

The detailed comparison between the proposed method and EsOCL solver with GA, SA, EA and QGA within three case studies is presented in Figure 5.7. In CSM case

Table 5.6: The results of success rate of the proposed method and OCL solver with GA, EA, SA, and QGA techniques in case study 1 (CSM).

CSM TC ID	EsOCL Solver				Proposed Method			
	GA	SA	EA	QGA	GA	SA	EA	QGA
1	0.21	0.7	0.32	0.33	1	1	0.98	0.98
2	0.19	0.75	0.24	0.3	1	1	0.98	0.92
3	0.31	0.8	0.31	0.43	1	0.95	0.98	0.93
4	0.31	0.83	0.33	0.23	1	0.98	0.95	0.95
5	0.25	0.76	0.26	0.22	0.98	1	1	0.86
6	0.24	0.75	0.1	0.1	1	0.98	0.99	0.98
7	0.16	0.82	0.2	0.22	0.67	1	0.97	0.93
8	0.45	0.67	1	1	1	1	0.5	0.6
9	1	1	1	1	1	1	1	1

study, it has been noticed from Figure 5.7 (a) that clear drop in the performance of the proposed method with GA in test case 7 while achieved 100 percent in the others. The proposed method with SA achieved almost 100 percent of success rate in all test cases as illustrated in Figure 5.7 (b). EsOCL solver with both EA and QGA outperformed the proposed method in test case 8 as presented in Figure 5.7 (c) and (d). Generally, SA with the proposed method obtained stable success compared to others SBTs in CSM case study due to SA apply local search mechanism which is suitable for low optimum search space.

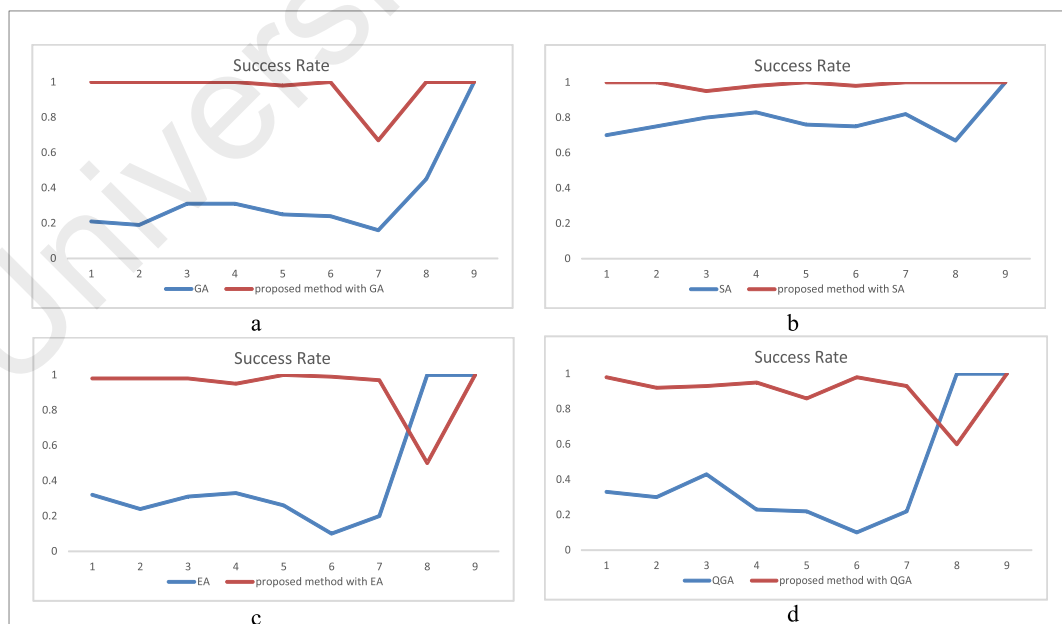


Figure 5.7: The results of success rate of the proposed method and EsOCL solver with four SBTs in case study 1 (CSM) from Table 5.6.

Table 5.7: The results of success rate of the proposed method and EsOCL solver with GA, EA, SA, and QGA techniques in case study 2 (TIS).

TIS	EsOCL Solver				Proposed Method			
TC ID	GA	SA	EA	QGA	GA	SA	EA	QGA
1	0.81	1	1	0.5	1	1	1	1
2	1	1	0.8	1	1	1	1	1
3	1	1	1	1	1	1	1	1
4	1	1	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1
6	0.73	1	1	0.8	1	1	1	0.8
7	0.15	0.28	0.4	0.01	0.7	0.7	0.7	0.5
8	0.42	0.56	0.3	0.01	0.8	0.75	0.7	0.65
9	0.31	0.35	0.5	0.1	0.85	0.6	0.7	0.6
10	0.31	0.35	0.3	0.1	0.6	0.8	0.7	0.35
11	0.92	0.32	0.2	0.1	0.8	0.83	0.7	0.5
12	0.99	0.35	0.3	0.01	0.87	0.73	0.85	0.53
13	0.18	0.31	0.1	0.1	0.87	0.77	1	0.37
14	0.21	0.31	0.3	0.3	0.83	0.75	0.95	0.77
15	0.17	0.31	0.2	0.5	0.87	0.85	0.77	0.6
16	0.2	0.32	0.3	0.2	0.73	0.73	0.8	0.6
17	0.18	0.31	0.1	0.2	0.8	0.75	0.95	0.75
18	0.2	0.26	0.3	0.5	0.7	0.9	0.73	0.7
19	0.65	0.25	0.8	0.4	0.85	0.9	0.9	0.7
20	0.72	0.97	1	1	1	1	1	1
21	0.69	0.98	1	1	1	1	1	1
22	0.79	0.55	0.8	0.2	0.8	1	0.9	0.65
23	0.79	0.56	0.5	0.4	0.75	1	0.75	0.75
24	0.43	0.57	0.1	0.1	0.8	0.7	0.8	0.7
25	0.16	0.26	0.3	0.6	0.9	1	0.8	0.75
26	0.22	0.3	0.2	0.1	0.8	0.85	0.7	0.4
27	0.22	0.28	0.4	0.1	1	0.8	0.75	0.6
28	0.23	0.32	0.8	1	1	1	1	0.8
29	0.72	0.96	0.5	0.1	1	0.93	1	0.6

In TIS case study, the success rates of the proposed method with all SBTs were different as shown in Figure 5.8. The proposed method with SA and EA significantly outperformed EsOCL solver in all test cases. On the other hand, EsOCL solver with GA slightly obtained better success rates than the proposed method in test cases 11, 12 and 23 and almost similar success rates in test cases 10-22. The proposed method with SA obtained roughly similar test data with EsOCL solver in test cases 20,21 and 29. Test cases 1, 3-6, and 20-21 got same success rates by EA with both the proposed method and

Table 5.8: The results of success rate of the proposed method and EsOCL solver with GA, EA, SA, and QGA techniques in case study 3 (Eu-Rent).

Eu-Rent TC ID	EsOCL Solver				Proposed Method			
	GA	SA	EA	QGA	GA	SA	EA	QGA
1	0.18	0.05	0.24	0.19	0.78	0.39	0.69	0.12
2	0.16	0.08	0.21	0.16	0.9	0.39	0.93	0.33
3	0.14	0.13	0.21	0.16	0.42	0.21	0.48	0.48
4	0.09	0.1	0.1	0.01	1	0.75	1	1
5	0.08	0.67	0.06	0.01	1	0.51	1	1
6	0.05	0.75	0.01	0.09	1	0.6	1	1
7	0.16	0.6	0.2	0.15	1	0.3	1	1
8	0.1	0.67	0.17	0.12	1	0.81	1	1
9	0.1	0.23	0.1	0.09	0.3	0.3	0.3	0.36
10	0.1	0.43	0.14	0.1	0.42	0.21	0.42	0.48
11	0.19	0.5	0.25	0.18	0.75	0.39	0.69	1
12	0.19	0.75	0.25	0.17	0.75	0.33	0.69	0.99

EsOCL solver, While test cases 2-5 and 20-21 obtained same data by QGA with both the proposed method and EsOCL solver. In general, the proposed method with all SBTs is superior than EsOCL in most of the test cases due to the proposed method heuristically optimized each variable separately rather than the variables of the constraint.

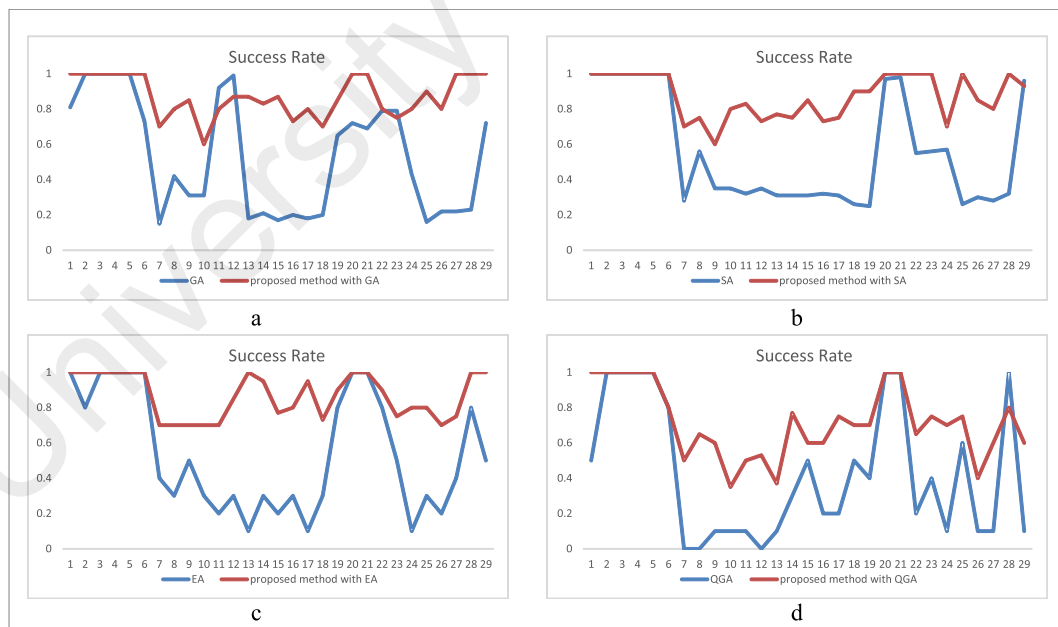


Figure 5.8: The results of success rate of the proposed method and EsOCL solver with four SBTs in case study 2 (TIS) from Table 5.7.

In Eu-Rent case study, Figure 5.9 shows that three global SBTs (GA,EA, and QGA) were significantly improved by the proposed method. Performance of global SBTs were

almost similar in all the test cases while their performance with EsOCL solver were very low. EsOCL solver and the proposed method with SA achieved roughly similar success rates. Specifically, EA and GA with the proposed method obtained the same or almost similar success rates in all test case. There is a clear drop in the performance of GA, EA and QGA in test cases 1,3, and 9-11. On the other hand, the success rates of SA with both the proposed method and EsOCL solver were approximate. Test cases 4-9 with the proposed method achieved the optimal success rate by the three global SBTs while they got different success rates by SA.

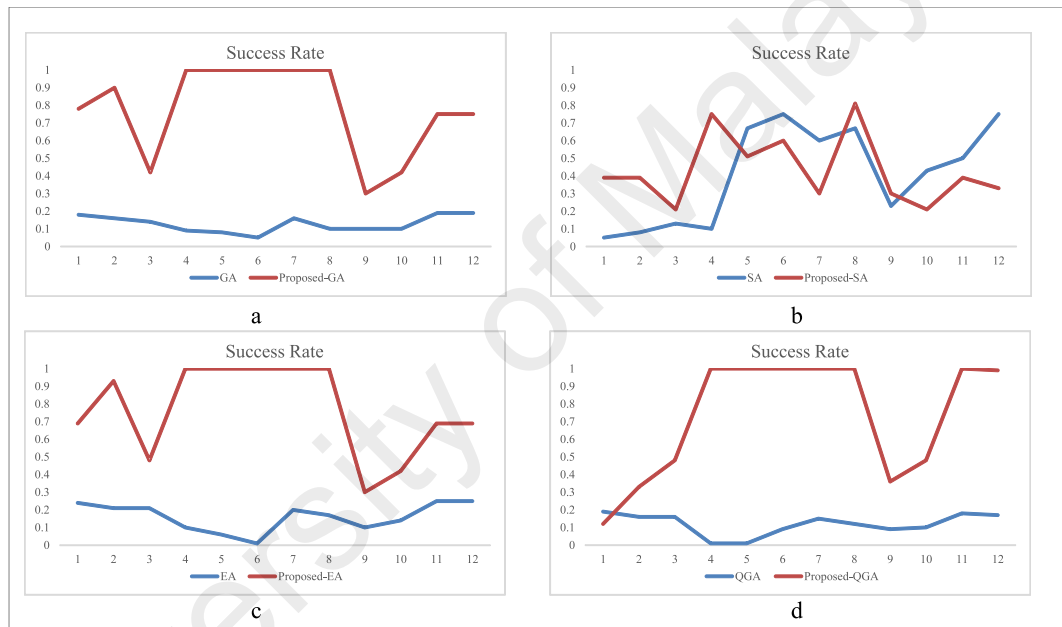


Figure 5.9: The results of success rate of the proposed method and EsOCL solver with four SBTs in case study 3 (EU-Rent) from Table 5.8.

Table 5.9 illustrates the time taken to generate the data by the proposed method and EsOCL solver in CSM case study. From the table, the proposed method cost more than EsOCL solver with all SBTs. In particular, test case 6 was the most costly while test case 9 was the most costless by all techniques. The other test cases took different time. Cost with respect to the effectiveness, the proposed method superior than EsOCL solver with all techniques. This high cost of the proposed method is due to the number of the variables is high and more time was needed to analyze these variables and generate the data.

Table 5.9: The results of generation time of the proposed method and EsOCL solver with GA, EA, SA, and QGA techniques in case study 1 (CSM).

TC ID	EsOCL Solver				The proposed Method			
	GA	EA	SA	QGA	GA	EA	SA	QGA
1	95.12	110.52	63.659	74.04	146.12	147.81	138.05	154.59
2	64.12	70.43	67.839	80.69	174.29	184.5	178.51	181.73
3	102.35	71.412	99.611	111.82	207.43	164.11	152.4	165.33
4	98.33	106.96	99.488	109.42	149.38	157.11	165.87	150.34
5	101.82	103.04	99.068	111.91	179.12	196.64	182.16	185.11
6	101.26	103.76	118.25	120.25	297.94	328.93	307.41	308.95
7	75.29	103.02	72.508	84.29	150.47	215.72	205.53	213.76
8	1.08	3.72	0.642	5.32	29.59	35.14	37.57	32.26
9	1	1.55	0.532	1.65	4.67	7.59	6.74	3.54

It has been concluded from Table 5.10 that, the proposed method increased the cost of the techniques in most test cases in TIS case study. In specific, the proposed method took significantly less time than EsOCL solver in test case 28 because of the number of variables in the constraints is very small compared to the constraints. Furthermore, the time of the proposed method was slightly less by GA in test cases 1, 12, and 27, by EA in test cases 1, 7, 13, 17, 19 and, SA in test cases 2, 3, 5 and 20 and by QGA in test cases 2, 21, and 23 while the rest of the test cases took more time compared to EsOCL solver. The test case with high number of variables compared to the number of constraints takes more time by the proposed method because it needs time to analyze the variables, to extract the constraints of the variables and then generate the data for the new constraints. Table 5.11 shows that the proposed method with global SBTs (GA, EA and QGA) has more cost than EsOCL solver in the Eu-Rent case study while the proposed method efficiently reduced the cost of SA. In particular, GA, and EA with the proposed method cost less in the test cases that have constraints with a few number of variables such as test cases 4-8 whereas the proposed method with QGA has high cost in all test cases except test cases 4-5. To summarize, the proposed method with QGA is more efficient with high cost in the Eu-Rent case study while the proposed method improved the effectiveness and the cost of

Table 5.10: The results of generation time of the proposed method and EsOCL solver with GA, EA, SA, and QGA techniques in case study 2 (TIS).

TIS TC ID	EsOCL Solver				The proposed Method			
	GA	EA	SA	QGA	GA	EA	SA	QGA
1	0.5	1.25	0.452	1.5	0.4	0.9	0.53	1.24
2	0.7	4.18	4.21	2.97	6.15	5.09	1.343	0.672
3	1.28	6.69	6.71	4.67	5.374	12.54	5.925	5.668
4	0.72	3.7	7.48	2.65	5.02	5.06	12.18	6.73
5	1.31	6.58	7.48	4.56	5.624	12.67	5.725	6.307
6	4.537	7.91	10.94	8.44	9.82	11.18	11.56	10.48
7	107.74	364.99	111.57	119.49	224.42	220.87	231.1	232.26
8	62.85	66.4	71.05	71.88	91.85	95.54	97.75	97.21
9	110.32	86.57	85.9	91.25	112.75	115.91	113.59	118.49
10	98.73	96.98	104.8	108.6	185.34	156.62	188.24	194.03
11	136.37	112.32	101.72	106.79	168.25	189.06	157.16	162.12
12	190.97	126.28	132.9	139.12	188.88	186.06	194.37	197.37
13	124.9	122.73	131.7	139.54	193.63	190.67	196.75	198.69
14	125.65	122.18	133.02	140.2	187.59	200.94	194.52	210.02
15	128.73	124.9	132.43	139.3	189.04	194.06	190.9	200.53
16	128.95	122.73	132.08	143.31	191.16	195.07	190.62	197.65
17	128.15	124.82	133.12	139.33	189.6	60.29	190.86	203.14
18	123.84	125.06	132.17	143.92	195.87	179.5	188.18	197.7
19	33.74	36.46	42.16	49.1	47.82	18.41	58.46	49.716
20	7.67	5.5	12.8	8.57	10.72	12.3	12.49	10.93
21	8.13	5.56	9.51	12.68	10.48	12.05	12.49	11.08
22	13.7	9.55	12.65	12.55	14.11	19.26	17.01	51.63
23	6.85	9.56	13.41	16.22	18.986	12.18	17.05	15.01
24	125.43	134.8	133.86	110.43	174.71	160.6	189.9	149.8
25	125.71	130.13	109.94	139.94	172.53	160.1	158.54	197.26
26	96.88	98	107.51	110.63	164.61	184.2	161.46	164.04
27	103.76	100.09	105.83	118.27	37.15	115.99	160.25	163.22
28	124.16	103.59	132.53	140	10.63	11.95	39.02	38.51
29	9.45	9.37	8.35	9.1	15.401	16.49	17.271	17.404

SA.

5.3.4 Overall Discussion

For general overview, Figure 5.10 illustrates the success rate of four SBTs based on the 50 test cases; 9 test cases of CSM case study, 29 test cases of TIS case study and 12 test cases of Eu-Rent case study for the proposed method. The Figure shows that GA and EA achieved better results than the others with average success rate of 87 and 85 percent respectively. SA outperformed QGA with average success rate of 79 percent and

Table 5.11: The results of generation time of the proposed method and EsOCL solver with GA, EA, SA, and QGA techniques in case study 3 (EU-Rent).

EU-Rent TC ID	EsOCL solver				The proposed Method			
	GA	SA	EA	QGA	GA	SA	EA	QGA
1	123.76	145.52	133.15	143.64	160.31	71.04	151.40	465.21
2	144.35	174.71	176.95	151.25	151.18	101.33	146.04	455.45
3	141.76	179.36	156.18	145.59	175.44	97.49	182.15	514.14
4	12.98	69.72	27.09	24.05	6.53	2.63	10.24	22.18
5	42.11	52.52	41.98	32.46	7.25	4.92	12.96	27.15
6	35.24	62	47.76	35.98	17.98	6.69	24.07	56.82
7	37.16	59.03	51.16	36.32	23.21	8.24	27.73	66.11
8	43.85	73.91	62.99	43.87	27.12	9.03	31.12	85.87
9	108.91	142.59	135.79	111.06	140.63	62.63	147.87	497.70
10	125.51	129.36	137.19	108.45	149.35	78.08	150.38	451.42
11	111.36	139.49	135.95	116.31	142.27	79.79	158.32	484.77
12	161.41	191.77	205.07	161.67	233.85	102.13	251.29	808.97

75 percent respectively. It can be observed that, all the techniques achieved 100 percent for at least one test case. With the upper limit of 1000 iterations, both GA and SA achieve almost similar median success rates (90 and 85 percent) followed by QGA (75 percent) and EA exceeds better median success rate of 95 percent. It has been also concluded that the lowest success rate of SA is approximately 21, whereas the tendency of the lowest success rates of QGA is towards 1 percent. Both GA and EA at least solve 30 percent of the constraints. The highest 25 percent of the GA, SA, EA, and QGA success rates are 100 percent, while the lowest 25 percent of their success rates are 71 percent for both SA and EA while GA has the best lowest 25 percent and QGA has the worst percent. Overall, all the values in the figure refers to the preference of the proposed method with GA as compared to the others SBTs.

With respect to statistical check, the statistical paired t-test is carried out on the distributions of the success rates and generation time of all the four SBTs with the proposed method and EsOCL solver. Table 5.12 shows the statistical difference based on success rates and Table 5.13 depicts the differences of the cost.

Table 5.12 shows that the p-values are very close to 0 in all distributions compressions

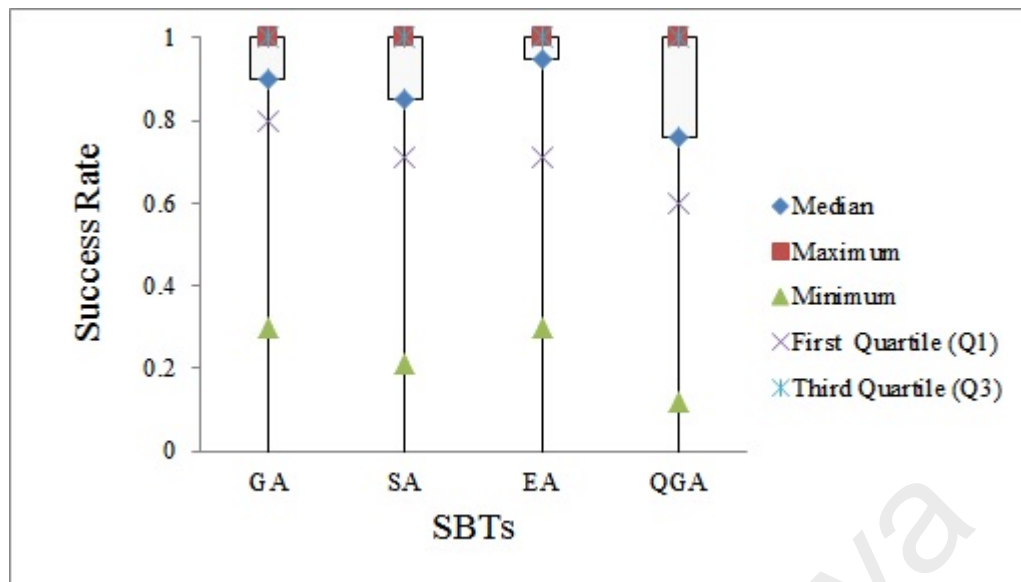


Figure 5.10: Box plot graph of success rate for the proposed method with four SBTs in all case studies.

in all case studies which refers to there is a significant statistical difference between the performance of the proposed method and EsOCL solver with the four techniques. In all case studies, the performance of the proposed method is statistically superior than the EsOCL solver due to the p-values of all techniques are close to 0. In comparing the performance of the four techniques with the proposed method, there is no difference between the performance of GA and SA in the first case studies while QGA, EA and GA are significantly better than SA in the third case study. Moreover, The performance of EA with both the proposed method and EsOCL solver is almost similar in all the case studies while GA is superior than QGA in the second case study. In TIS case study, the p-values of both SA, EA against QGA show that there is significant difference between the performances. In conclusion, the performance of the proposed method with four techniques almost similar in the CSM case study while there are differences between the techniques performance in the TIS and Eu-Rent case studies.

From Table 5.13, the p-values of the proposed method with all SBTs versus the EsOCL solver clearly indicate that EsOCL solver is faster than the proposed method, while SBTs

Table 5.12: The results of the paired t-test based on success rate.

Pairs of Techniques	p-value		
	CSM	TIS	EU-Rent
proposed-GA vs GA	4.12302E-05	3.50222E-07	2.28069E-06
proposed-EA vs EA	0.004961409	1.60809E-07	2.80473E-06
proposed-SA vs SA	0.00012836	9.52449E-08	0.000318955
proposed-QGA vs QGA	0.004006036	2.19692E-07	5.68935E-05
proposed-GA vs proposed-SA	0.235612884	0.487000852	2.31176E-05
proposed-GA vs proposed-EA	0.318893885	0.36114344	0.208745608
proposed-GA vs proposed-QGA	0.176627296	1.78485E-06	0.287976781
proposed-SA vs proposed-EA	0.145838268	0.381824312	2.76662E-05
proposed-SA vs proposed-QGA	0.040290512	2.05661E-06	0.002537812
proposed-EA vs proposed-QGA	0.164658629	3.19697E-06	0.329425517

with the proposed method take approximately the same time for finding solutions in CSM and TIS case studies. In the Eu-Rent case study, SA are significantly faster than others while EA and GA clearly took less time than QGA only. From these statistical analyzing, the proposed method is more efficient. The explanation of the difference between SBTs

Table 5.13: The results of the paired t-test based on generation time.

Pairs of Techniques	p-value		
	CSM	TIS	EU-Rent
proposed-GA vs GA	0.001622358	0.004236085	0.099107253
proposed-SA vs SA	0.002263924	0.04442145	1.79827E-09
proposed-EA vs EA	0.000880012	0.000144315	0.427345499
proposed-QGA vs QGA	0.001757743	0.000189329	0.001332036
proposed-GA vs proposed-SA	0.141719339	0.332352298	0.000628752
proposed-GA vs proposed-EA	0.344953505	0.076884107	0.021667511
proposed-GA vs proposed-QGA	0.250159687	0.024051135	0.000660532
proposed-SA vs proposed-EA	0.025868599	0.051680345	0.000464657
proposed-SA vs proposed-QGA	0.052212542	0.020046861	0.000634442
proposed-EA vs proposed-QGA	0.242698336	0.098660396	0.000716497

performances with the proposed method and EsOCL solver is that SA does local search and EA, QGA, and GA applies global search. Moreover, if the fitness landscape (search space) gives an obvious tendency into the global optima (best solutions in all search space), then SA concentrates on one of them, as compared to global SBTs (explore all the landscape). The search landscape of the case studies consists only a few local optima with EsOCL solver; a type of landscape in which local search enable to give efficient results.

However the proposed method makes the search landscape contains high number of local optima that need global search to obtain efficient results.

5.4 Threats to Validity

The same threats that presented in chapter 3 section 3.4 were migrated in this chapter.

5.5 Conclusion

This chapter presented the methodology used for evaluation the performance of the proposed method in two terms cost and effectiveness. Empirical case study was the method used to evaluate the cost and effectiveness for the proposed method in industrial context. This chapter also presented the results of this study performance evaluation. Case study method was followed for evaluating the cost and effectiveness of the proposed method with the Kalaji approach and EsOCL solver. The three open source industrial case studies were used in this evaluation. Four SBTs have been involved in this evaluation, including global SBTs (GA, EA and QGA), and local SBTs (SA). Four evaluation criteria were considered to measure the cost and effectiveness of both infeasible path detection approach and test data generator: size of test cases, detection rate, success rate, and generation time. The results of the four SBTs to solve the OCL constraints have been statistically tested using t-test.

The result showed that the proposed method was superior than the existing methods. The proposed infeasible path detection method significantly outperformed Kalaji detection approach and achieved 100 detection rates in two case studies and 90 percent in the third one. Due to the proposed method can detect the infeasible paths with complex constraints with various OCL data types while Kalaji approach can detect the infeasible paths containing simple constraint with integer data type only. From the results of the proposed search-based test data generator, the proposed method was significantly

efficient more than EsOCL solver with more cost. In specific, GA, SA, and EA achieved approximately same success rate and outperformed QGA in most test cases of TIS case study. Furthermore, the proposed method with SA obtained the best results while the proposed method with QGA achieved the lowest success rate in CSM case study. QGA outperformed the others techniques in EU-Rent case study. Generally, global SBTs with the proposed method achieve better results than local search (SA) in the proposed search-based test data generator for models, this findings aligns to the findings in (Harman & McMinn, 2010), which reported that global SBTs tends to achieve better results than local SBTs. Furthermore, higher success rate was often associated with longer generation time.

University of Malaysia

CHAPTER 6: CONCLUSIONS AND FUTURE WORK

It has been presented and demonstrated that the proposed method is able to efficiently and effectively generate feasible test cases from state machine models with OCL constraints. This chapter presents the conclusions of this thesis and identifies the future works. This chapter also describes how the aim and objectives of this research (in chapter 1) are fulfilled. This chapter also presents the contributions of this thesis and highlights the significance of the work undertaken in this study. To note that, the result of this study has been published in various peer reviewed ISI journal articles, conference paper as listed at the end of this chapter. Finally, the limitations of this study and possible future works are concluded.

The outline of this chapter are as follows: Section 6.1 describes how aim and objectives of this study are attained and section 6.2 presents contributions of this thesis. Significance of this work is presented in section 6.3. The publications of this study are listed in section 6.5. Limitations and future works are appeared in section 6.4.

6.1 Restatement of Research Aim

In this research, the aim is to achieve efficient feasible test case generation from UML state machine models with OCL constraints. The explanation of how the research aim could be attained is by realizing each objective stated in section 1.3.

- **To review the current state-based testing and the applications of search-based techniques for model-based testing to generate executable test cases.**

The objective was fulfilled and the most credible works reported in articles and conferences were reviewed. The recent works in state-based testing and SBTs for MBT were analyzed and synthesized. The applications of SBTs for MBT were categorized into four main categories, compressing problems, solutions, evaluation

and purpose. The taxonomy of state-based testing is based on two activity test data generation and path generations. Each main category has sub-categories. From the analysis, seven open challenges were identified, including hybrid technique, systematic guidelines, empirical case studies, model transformation, improving optimization process, infeasible paths and optimizing whole constraints in each test case. The infeasible path detection and optimizing the whole constraints were considered as the most significant and related problems in test case generation from UML state-based testing to be addressed in this research. This was presented in chapter 2.

- **To analyze the existing model-driven approach with infeasible path detection for generating abstract test cases and search-based test data generation for satisfying all OCL constraints in each abstract test case.**

In fulfilling this objective, the identified research problems of infeasible path detection and satisfying whole constraints were verified to be addressed in this research. In order to analyze the significance of this research problems, an analytical analysis was carried out using empirical case study on two industrial systems. In order to gain insight into the problems, the recent model-driven approach with Kalaji infeasible path detection and EsOCL solver were implemented. The methods were executed with the two case studies to harvest their cost and effectiveness. The results showed that the model-driven with infeasible path detection produced 79 percent of infeasible test cases because the Kalaji approach is limited to integer data type only. Also, the results showed that the EsOCL solver generated insufficient data to satisfy all the constraints by time. Chapter 3 presented this analysis.

- **To develop a method with an infeasible path detection and a fitness function**

that evolves itself using error feedback for satisfying whole OCL constraints in each abstract test case.

The objective was realized by proposing an efficient test case generation method to increase infeasibility detection rate and the success rates. The main novel principles employed in this method are that developing a model-driven path generator with infeasible path detection and search-based test data generator with whole constraints analyzer and fitness function evolves itself. The infeasible path detection is based on penalty estimation for the relations and equations of all OCL data types including primitive (String, Integer, Real, and Boolean) and complex (Set,OrderedSet,Bag,Sequence,Enumerations, and Tuple). The whole constraints analyzer creates a constraint related to a certain variable from the existing constraints. The new constraint is used by the proposed fitness function that evolves itself based on error feedback. The fitness function was used by four SBTs (GA, Quantum GA, EA and SA). The optimum data was generated by this method that able to satisfy all the constraints at the same time. This was presented in chapter 4.

- **To evaluate the ability of the proposed method to detect infeasible paths and generate optimal data using three industrial case studies of embedded systems**

This objective was attained by evaluating the proposed method via empirical case study using three open source industrial systems. The ability of the infeasible path detection was evaluated with different complex OCL constraints. The proposed test data generator with four SBTs were run and repeated 1000 times for the sake of reducing random variation. Cost and effectiveness of the proposed method were measured and analyzed. The performance analysis results unveiled that utilizing the proposed method to perform feasible test case generation improved the infeasibility

detection rate by 96.7 percent and success rates by 87 percent for GA, 85 percent for EA, percent 79 for SA and 75 percent for QGA in average of the three case studies compared with Kalaji approach and the EsOCL solver. The results of performance evaluation were validated via statistical testing. T-test analysis was used as a common approach to derive accurate statistical overview of success rates. Statical results confirmed that leveraging the proposed method to generate feasible test cases increased the effectiveness compared with the Kalaji and EsOCL solver. This was presented in chapter 5.

6.2 Contributions

In this research, several contributions have been produced to the body of knowledge as following.

- **Taxonomy of the applications of search-based techniques for model-based testing.**

The taxonomy of the applications of SBTs for MBT was produced. The proposed taxonomy presents four major concepts (purpose, problem, solution and evaluation) and sub-categories of each main category. The proposed taxonomy can be considered as a basic framework to classify existing work in this

field. The taxonomy also can be useful in developing and evaluating future work.

This study should be of interest to several stakeholders. First, the MBT research community since this review indicates a discrepancy between MBT aspects (such as model transformation, constraint, and coverage criteria) and SBTs. Second, search-based testing community since it shows how far SBTs investigated in MBT. Finally, individuals and academia community which interested in applying SBTs for MBT because this study provides a comprehensive overview and open a gateway for new

research opportunities. This study, appeared in chapter 2 and published in (Saeed, Hamid, & Mustafa, 2016).

- **Taxonomy for state-based testing.**

The taxonomy of state-based testing was devised. The taxonomy differentiates test case generation steps in state-based testing into two categories: path generation and data generations. The proposed taxonomy presents four major aspects of path generation (including coverage criteria, test model, all path generation, and infeasible path detection) and six major data generation techniques (comprising Symbolic execution, model checking, random search, constraint solving, theorem proving and SBTs). The taxonomy helps researchers in distinguishing all the concepts related to the test case generation from state models and exploring the existing methods and techniques that achieved these concepts. The findings of this contribution were presented in chapter 2 and will be published in the literature.

- **Empirical analysis of the cost and effectiveness of existing search-based techniques for model-based testing.**

A contribution was added to the body of knowledge by empirically analyzing performance of model-driven approach with Kalaji infeasible detection for generating feasible test cases and EsOCL solver for generating test data satisfying OCL constraints. With the help of empirical case study, the cost and effectiveness of these methods were analyzed in term of seven evaluation metrics. The results showed noticeable insufficient detecting rates of infeasible paths when the OCL constraints contain non-integer data types and low success rates of generating optimal data satisfying whole constraints in each test case. The results of this analysis were presented in chapter 3 and published in the literature (Saeed, Hamid, & Sani, 2016).

This study is a powerful evidence about the suitability of SBTs and model-driven approach for generating test case.

- **Efficient method for generating feasible test cases from state machine models with OCL constraints.**

A research was achieved that generating feasible executable test cases from UML state machine models with all types of OCL constraints which are lacking in the literature. While focus in traditional solutions was on detecting infeasible paths containing integer data type with its operations and on generating data to satisfy one constraint only, this is the first effort that proposed infeasible path detection for all OCL data types and search-based test data generator for whole constrains at one time. The proposed method is able to be used in industry for testing systems because the proposed method showed an effective performance on the industrial open source systems. The proposed method was reported in chapter 4 and will be published in the literature.

- **Empirical analysis for the proposed method.**

The empirical and analytical evaluation of the proposed method was produced using empirical case study method. Performance evaluation using empirical case study was performed on three industrial open source systems. The method also was validated by comparing its results with the recent existing method (Kalaji and EsOCL solver). The performance evaluation is reported in chapter 5 and the results are presented in chapter 6. Statistical test of the results shows high performance of the proposed method and advocate that objectives of this study are fulfilled and the aim is realized. This study, is a prove of the effectiveness of the proposed method compared to the recent research, which lined with the idea of evidence-based

software engineering (Dyba et al., 2005).

6.3 Significance of the Work

The significant of the proposed method have been described in section 4.2. These features were demonstrated in chapter 3.4.5. The features are as follows.

Feasibility: The most significant feature of the proposed method is that the generated test cases are feasible and all the generated data able to satisfy all the OCL constraints in the test case at least one time. This feature was demonstrated by detecting the infeasible paths in chapter 4 (section 4.1.1.1).

Robustness: One of the significant features of the proposed method is the robustness of the generated test cases which obtains from solving complex OCL constraints on the properties of the systems emulating faulty situations. These constraints were solved at the same time by the proposed search-based test data generator 4.1.2 during test case generation in order to set the system properties in such a way as to trigger faulty situations. This feature was demonstrated by the proposed search-based test data generator in chapter 4.

Extensible: One of the significant feature of the proposed method is the extensibility of the method for various contexts which inherits from applying model transformations. In specific, the proposed method can be extensible to various UML models, various output scripting languages such as C++, test models such as testing flow graph, and coverage criteria such as all transition for different application domains and systems. These are examples of useful extensions for the proposed method. This feature was demonstrated by using ATL model-to-model transformation and MOFScript model-to-text transformation. This transformations based on developed rules which need metamodels as inputs. Therefore, to extend to other contexts, new rules and

metamodels can be added.

Configurable: The proposed method is easily configurable to satisfy varying requirements such as input model, coverage criteria and scripting language. High configurability enables testers to experiment with various techniques without significant effort and changes in the implementation. This is of practical importance as different test models, coverage criteria, and scripting language helps in targeting different types of faults. This feature is because the method was implemented by Eclipse and MDE and can be extensible for different contexts.

Coverage: Path coverage (includes transitions and states) measures the number of path traversed during the generation process as compared to the actual number of path that exist in the subject. High coverage was achieved by the proposed method, where in all aspect of the state models were sufficiently considered in the generation process. Thus, test cases that has a good coverage statistic were generated as presented in the results in chapter 3 and 5 (100 percent stat and transition coverage).

Independence: The proposed method generated test cases with no interaction between them, in which there is no test case depends on output of other test case. The test cases were run in any order without having any impact on the overall evaluation of the test suite. This is because each path was transformed as one abstract test case (java file) by the MOFScript language as shown in section 5.2.2.

Costless: The proposed method was implemented and developed with open source languages and softwares. Eclipse with its modeling plugins were used for developing test case generation and open source library for search-based test data generator. This free cost of developing reduced the cost of the entire testing cost.

Easy to use: The proposed method is easy to use because it is configurable and flexible enough to cater to a variety of different contexts. Different systems have various testing needs and the proposed method is able to be adapted to different testing processes.

Other features are inherited from benefiting of applying automatic test generation.

Compression testing cost: The proposed auto generating test cases method significantly contributes to reduced the cost of software testing. The reasons are the followings: A big portion of testing cost goes toward writing test cases. The proposed method generates a comprehensive, current set of feasible test cases covering all possible outcomes which saves time and money that can be reallocated.

Quality improvement: Creating test cases manually brings with it the risk of human error. The proposed method creates automatically from high quality requirements which diminishing that risk. Therefore, generated test cases are more accurate with complete test coverage, so no need for running a series of tests only to discover later the human errors. Furthermore, its also simple to regenerate test cases as models change, keeping requirements and test cases in synchronization.

6.4 Limitation and Future Work

One of the limitations of this study is that error feedback concept in the fitness function did not applied for the non-numerical data types such as String. Flattening and checking the consistency of the models was not covered in this study. The future work of this study is to extend the proposed fitness function for non-numerical data types. Develop the fattening part into the proposed method. Additional SBTs, also will be investigated, that are commonly used in the SBSE community, such as particle swarm optimization and alternative variable method, on an additional case studies.

6.5 International Scholarly Publications

The list of publications related (in whole or part) to the research undertaken in this thesis is as follows.

- **Saeed A.**, Hamid S. H., & Mustafa M. The Applications of Search-Based Techniques for Model Based Testing: Taxonomy and Systematic Literature Review. *Applied Soft Computing*. 2016 . (Published) (Q1 ISI-Cited Publication).
- **Saeed A.**, Hamid S. H., & Mustafa M. Software Testing Research : Past Research and Future Direction. *e-informatica software engineering journal*. 2015 . (Under review) (2015) (Scopus-Cited Publication)
- **Saeed A.**, & Hamid S. H. The Applications of Search-Based Techniques for Extended Finite State Machines- based Testing: Issues and Open Challenges. *Malaysian Conference on Software Engineering*. 2015 . (Presented)
- **Saeed A.**, Hamid S. H., & Sani A. Search-based Techniques for Test Case Generation from UML State Machine Models, Post Graduate Research Excellence Symposium (PGRES) 2016, University Malaya. (Best PhD Presentation Award)
- **Saeed A.**, Hamid S. H., & Sani A. Empirical Analysis of Search-based Techniques and Model Driven Approach in Model-based Testing: Industrial Case Studies. *International Journal of Software Engineering and Knowledge Engineering*. 2016 . (ISI-Cited Publication) (Accepted).
- **Saeed A.**, Hamid S. H., & Sani A. An Effective Test case Generation from UML State Machine Models with OCL Constraints Using Search-based Techniques. *Applied Soft Computing*. 2016 (Q1 ISI-Cited Publication) (First Draft).

- **Saeed A.**, Hamid S. H., & Sani A. State-based Testing: Taxonomy, Systematic Review. Information and Software Technology. 2016 (Q1 ISI-Cited Publication) (First Draft).

University of Malaya

REFERENCES

- Adamopoulos, K., Harman, M., & Hierons, R. M. (2004). How to Overcome the Equivalent Mutant Problem and Achieve Tailored Selective Mutation using Co-Evolution. In *Proceedings of the 2004 conference on genetic and evolutionary computation (gecco 04)* (pp. 1338–1349). Seattle, Washington, USA: Springer Berlin / Heidelberg.
- Ahmed, B. S., Zamli, K. Z., & Lim, C. P. (2012, April). Application of Particle Swarm Optimization to uniform and variable strength covering array construction. *Applied Soft Computing*, 12(4), 1330–1347.
- Aleti, A., Buhnova, B., Grunske, L., Koziolok, A., & Meedeniya, I. (2013, May). Software Architecture Optimization Methods: A Systematic Literature Review. *IEEE Transactions on Software Engineering*, 39(5), 658–683.
- Ali, S. (2011). *Scalable Model-based Robustness Testing : Novel Methodologies and Industrial Application* (Unpublished doctoral dissertation). University of Oslo.
- Ali, S., Briand, L. C., Arcuri, A., & Walawege, S. (2011). An Industrial Application of Robustness Testing Using Aspect-Oriented Modeling , UML / MARTE , and Search Algorithms. In *Model driven engineering languages and systems* (pp. 108–122).
- Ali, S., Briand, L. C., Hemmati, H., & Panesar-Walawege, R. K. (2010). A systematic review of the application and empirical investigation of search-based test case generation. *IEEE Transactions on Software Engineering*, 36, 742–762.
- Ali, S., Hemmati, H., Holt, N. E., Arisholm, E., & Briand, L. C. (2010). *Model Transformations as a Strategy to Automate Model-Based Testing A Tool and Industrial Case Studies* (Tech. Rep.).
- Ali, S., Iqbal, M. Z., & Arcuri, A. (2013). Generating Test Data from OCL Constraints with Search Techniques. *IEEE Transactions on Software Engineering*, 39(10), 1376–1402.
- Ali, S., Iqbal, M. Z., Arcuri, A., & Briand, L. (2011). A search-based OCL constraint solver for model-based test data generation. In *Proceedings - international conference on quality software* (pp. 41–50).

- Ali, S., Iqbal, M. Z., Khalid, M., & Arcuri, A. (2015, June). Improving the performance of OCL constraint solving with novel heuristics for logical operations: a search-based approach. *Empirical Software Engineering*, 1–44.
- Alshahwan, N., & Harman, M. (2012). State aware test case regeneration for improving web application test suite coverage and fault detection. *Proceedings of the 2012 International Symposium on Software Testing and Analysis - ISSTA 2012*, 45.
- Alsmadi, I., Alkhateeb, F., Maghayreh, E. A., Samarah, S., & Doush, I. A. (2010). Effective Generation of Test Cases Using Genetic Algorithms and Optimization Theory. *Journal of Communication and Computer*, 7(11), 72–82.
- Ammann, P., & Offutt, J. (2008). *Introduction to software testing*. Cambridge University Press.
- Anand, S., Burke, E. K., Chen, T. Y., Clark, J., Cohen, M. B., Grieskamp, W., . . . McMinn, P. (2013, August). An orchestrated survey of methodologies for automated software test case generation. *Journal of Systems and Software*, 86(8), 1978–2001.
- Arcuri, A., & Fraser, G. (2013, February). Parameter tuning or default values? An empirical investigation in search-based software engineering. *Empirical Software Engineering*, 18(3), 594–623.
- Arcuri, A., Iqbal, M., & Briand, L. (2010). Black-Box System Testing of Real-Time Embedded Systems using Random and Search-Based Testing. In *International conference on testing software and systems* (pp. 95–110). Springer Berlin Heidelberg.
- Arora, A., & Sinha, M. (2014, February). State Based Test Case Generation using VCL-GA. In *2014 international conference on issues and challenges in intelligent computing techniques (icict)* (pp. 661–665). IEEE.
- AviZienis, A., Laprie, J.-C., Randell, B., & Landwehr, C. (2004). Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1), 11–33.
- Banerjee, I., Nguyen, B., Garousi, V., & Memon, A. (2013, October). Graphical user interface (GUI) testing: Systematic mapping and repository. *Information and Software Technology*, 55(10), 1679–1694.

- Baresel, A., Pohlheim, H., & Sadeghipour, S. (2003). Structural and Functional Sequence Test of Dynamic and State-based Software with Evolutionary Algorithms. In *Genetic and evolutionary computation gecco 2003* (pp. 2428–2441). Springer Berlin Heidelberg.
- Bate, I., & Fairbairn, M. (2013). Searching for the minimum failures that can cause a hazard in a wireless sensor network. In *the fifteenth annual conference on genetic and evolutionary computation- gecco '13* (p. 1213). Amsterdam, Netherlands: ACM.
- Bianchi, L., Dorigo, M., Gambardella, L. M., & Gutjahr, W. J. (2008, September). A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing*, 8(2), 239–287.
- Binder, R. (2000). *Testing object-oriented systems: models, patterns, and tools*. Addison-Wesley Professional.
- Blanco, R., Tuya, J., & Adenso-Díaz, B. (2009, April). Automated test data generation using a scatter search approach. *Information and Software Technology*, 51(4), 708–720.
- Blum, C., Puchinger, J., Raidl, G. R., & Roli, A. (2011). Hybrid metaheuristics in combinatorial optimization: A survey. *Applied Soft Computing*, 11(6), 4135–4151.
- Bouchachia, A., Mittermeir, R., Sielecky, P., Stafiej, S., & Zieminski, M. (2010, June). Nature-inspired techniques for conformance testing of object-oriented software. *Applied Soft Computing*, 10(3), 730–745.
- Briand, L., Labiche, Y., & Lin, Q. (2010). Improving the coverage criteria of UML state machines using data flow analysis. *SOFTWARE TESTING, VERIFICATION AND RELIABILITY*(April 2009), 177–207.
- Briand, L. C., Labiche, Y., & Shousha, M. (2005). Stress testing real-time systems with genetic algorithms. In *Proceedings of the 2005 conference on genetic and evolutionary computation - gecco '05* (p. 1021). New York, USA: ACM.
- Briand, L. C., Labiche, Y., & Wang, Y. (2004). Using Simulation to Empirically Investigate Test Coverage Criteria Based on Statechart. In *Proceedings of the 26th international conference on software engineering*. IEEE.

- Briand, L. C., Society, I. C., Penta, M. D., & Labiche, Y. (2004). Assessing and Improving State-Based Class Testing : A Series of Experiments. *IEEE Transactions on Software Engineering*, 30(11), 770–793.
- Brucker, A. D., Krieger, M. P., & Longuet, D. (2011). A Specification-Based Test Case Generation Method for UML / OCL. In *International conference models in software engineering* (pp. 334–348).
- Brucker, A. D., & Wolff, B. (2013). On theorem prover-based testing. *Formal Aspects of Computing*, 25(5), 683–721.
- Buehler, O., & Wegener, J. (2003). Evolutionary Functional Testing of an Automated Parking System. In *Proceedings of the international conference on computer, communication and control technologies (ccct'03) and the 9th. international conference on information systems analysis and synthesis (isas'03)*. Florida, USA.
- Buehler, O., & Wegener, J. (2005). Evolutionary Functional Testing of a Vehicle Brake Assistant System. In *6th metaheuristics international conference* (pp. 157–162).
- Bühler, O., & Wegener, J. (2004, March). Automatic Testing of an Autonomous Parking System Using Evolutionary Computation. In *Sae 2004 world congress*.
- Bühler, O., & Wegener, J. (2008a, October). Evolutionary functional testing. *Computers & Operations Research*, 35(10), 3144–3160.
- Bühler, O., & Wegener, J. (2008b, October). Evolutionary Functional Testing. *Computers & Operations Research*, 35(10), 3144–3160.
- Cabot, J., Claris, R., & Riera, D. (2008). Verification of UML / OCL Class Diagrams using Constraint Programming. In *Ieee intl conf. software testing verification and validation workshop*.
- Cabot, J., & Gogolla, M. (2012). Object Constraint Language (OCL): a Definitive Guide. In *Formal methods for model-driven engineering* (pp. 1–33). Springer Berlin Heidelberg.
- Cantenot, J., Ambert, F., & Bouquet, F. (2014). Test generation with Satisfiability Modulo Theories solvers in model-based testing. *Software Testing Verification and Reliability*, 24(May), 499–531. doi: 10.1002/stvr

- Cheng, Xichao, Yong Cheng, & Zhao, R. (2011). Efficient Path-Oriented Test Data Generation Algorithm for EFSM With Simulated Annealing. In *International conference on computer engineering and technology*. ASME Press.
- Chimisliu, V., & Wotawa, F. (2013a, July). Improving Test Case Generation from UML Statecharts by Using Control, Data and Communication Dependencies. *2013 13th International Conference on Quality Software*, 125–134. doi: 10.1109/QSIC.2013.48
- Chimisliu, V., & Wotawa, F. (2013b, July). Using Dependency Relations to Improve Test Case Generation from UML Statecharts. In *2013 IEEE 37th Annual Computer Software and Applications Conference Workshops* (pp. 71–76). IEEE. doi: 10.1109/COMPSACW.2013.24
- Clarke, J., Dolado, J. J., Harman, M., Hierons, R., J. B., Lumkin, M., . . . Roper, M. (2003). Reformulating software engineering as a search. In *Iee software* (pp. 161–175).
- Coello, C. A. C. (2002). Theoretical and Numerical Constraint-Handling Techniques used with Evolutionary Algorithms : A Survey of The State of The Art. *Computer Methods in Applied Mechanics and Engineering*, 191(11), 1245–1287.
- Dabney, J. B., & Harman, T. L. (n.d.). *Mastering Simulink* (Tech. Rep.). University of HoustonClear Lake.
- Davies, E., McMaster, J., & Stark, M. (1994). *Use of Genetic Algorithms for Flight Test and Evaluation of Artificial Intelligence and Complex Software Systems* (Tech. Rep.). Technical Report AD-A284824, Naval Air Warfare Center, Patuxent River.
- Delahaye, M., Botella, B., & Gotlieb, A. (2015, February). Infeasible path generalization in dynamic symbolic execution. *Information and Software Technology*, 58, 403–418.
- Derderian, K., Hierons, R. M., & Harman, M. (2006). Automated Unique Input Output Sequence Generation for Conformance Testing of FSMs. *The Computer Journal*, 49(3), 331–344.
- Derderian, K., Hierons, R. M., Harman, M., & Guo, Q. (2009, November). Estimating the Feasibility of Transition Paths in Extended Finite State Machines. *Automated Software Engineering*, 17(1), 33–56.

- Derderian, K. A. (2006). *Automated test sequence generation for Finite State Machines using Genetic Algorithms* (Unpublished doctoral dissertation). Brunel University.
- de Souza, J. T., Maia, C. L., de Freitas, F. G., & Coutinho, D. P. (2010, September). The Human Competitiveness of Search Based Software Engineering. *2nd International Symposium on Search Based Software Engineering*, 143–152.
- D-MINT. (n.d.). *Deployment of Model-Based Technologies to Industrial Testing*. Retrieved 2014, from <http://www.elvior.com/motes/d-mint>
- Doganay, K., Bohlin, M., & Sellin, O. (2013, March). Search Based Testing of Embedded Systems Implemented in IEC 61131-3: An Industrial Case Study. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops* (pp. 425–432). IEEE.
- Duale, A. Y., & Uyar, M. U. (2004). A method enabling feasible conformance test sequence generation for EFSM models. *IEEE Transactions on Computers*, 53(5), 614–27.
- Dyba, T., Kitchenham, B. A., & Røgnesen, M. J. (2005). Evidence-Based Software Engineering for Practitioners. *IEEE Software*, 22(1), 58–65.
- Enoiu, E. P., Doganay, K., Bohlin, M., Sundmark, D., & Pettersson, P. (2013, May). MOS: An integrated model-based and search-based testing tool for Function Block Diagrams. In *2013 1st International Workshop on Combining Modelling and Search-Based Software Engineering (CMSBSE)* (pp. 55–60). IEEE.
- Ensan, F., Bagheri, E., & Ga, D. (2012). Evolutionary Search-Based Test Generation for Software Product Line Feature Models. In *Advanced Information Systems Engineering* (pp. 613–628).
- European Railway Agency. (2015). *ERTMS-System Requirements Specification-UNISIG SUBSET-026*. Retrieved August 2015, from http://www.era.europa.eu/Document-Register/Pages/ERA_TSI_Dev_for_HS.aspx
- European Union. (2014). *ARTEMIS Embedded Computing Systems*. Retrieved from http://www.artemis-ju.eu/embedded_systems
- Farooq, U., & Lam, C. P. (2009). A Max-Min Multiobjective Technique to Optimize Model Based Test Suite. *2009 10th ACIS International Conference on Software*

Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing, 569–574.

- Farooq, U., & This. (2011). *Model Based Test Suite Minimization Using Metaheuristics* (Unpublished doctoral dissertation). Edith Cowan University.
- Feldstein, J. (2005). *Model-Based Testing using IBM Rational Functional Tester* (Tech. Rep.). Developer Works, IBM.
- Fischer, M., & Tonjes, R. (2012, September). Generating test data for black-box testing using genetic algorithms. In *Proceedings of 2012 IEEE 17th international conference on emerging technologies & factory automation (etfa 2012)* (pp. 1–6). IEEE.
- Fraser, G., & Arcuri, A. (2011). EvoSuite : Automatic Test Suite Generation for Object-Oriented Software. In *Acm symposium. foundations of software engineering* (pp. 11–14).
- Fraser, G., & Arcuri, A. (2013a, November). 1600 faults in 100 projects: automatically finding faults while achieving high coverage with EvoSuite. *Empirical Software Engineering*, 20(3), 611–639.
- Fraser, G., & Arcuri, A. (2013b, February). Whole Test Suite Generation. *IEEE Transactions on Software Engineering*, 39(2), 276–291.
- Freitas, F. G. D., & Souza, J. T. D. (2011). Ten Years of Search Based Software Engineering : A Bibliometric Analysis. In *3rd international symposium on search based software engineering* (pp. 18–32). Springer-Verlag Berlin Heidelberg.
- Frías, L., Queralt, A., & Ramon, A. O. (2003). *EU-Rent car rentals specification* (Tech. Rep.).
- Friske, M., & Schlingloff, B.-H. (2007). Improving Test Coverage for UML State Machines Using Transition Instrumentation. In *Computer safety, reliability, and security* (pp. 301–314). Springer-Verlag Berlin Heidelberg.
- Galeotti, J. P., Fraser, G., & Arcuri, A. (2013, November). Improving search-based test suite generation with dynamic symbolic execution. In *2013 IEEE 24th international symposium on software reliability engineering (issre)* (pp. 360–369). IEEE.

- Garousi, V. (2010, November). A Genetic Algorithm-Based Stress Test Requirements Generator Tool and Its Empirical Evaluation. *IEEE Transactions on Software Engineering*, 36(6), 778–797.
- Girgis, M. R. (2005). Automatic Test Data Generation for Data Flow Testing Using a Genetic Algorithm. *Journal of Universal Computer Science*, 11(6), 898–915.
- Gogolla, M., Büttner, F., & Richters, M. (2007, December). USE: A UML-based specification environment for validating UML and OCL. *Science of Computer Programming*, 69(1-3), 27–34. Retrieved from <http://linkinghub.elsevier.com/retrieve/pii/S0167642307001608> doi: 10.1016/j.scico.2007.01.013
- Gorev, N. B., Kodzheshirova, I. F., Kovalenko, Y., Álvarez, R., Prokhorov, E., & Ramos, A. (2011, February). Evolutionary Testing of Hydraulic Simulator Functionality. *Water Resources Management*, 25(8), 1935–1947.
- Grosan, C., & Abraham, A. (2007). Hybrid Evolutionary Algorithms : Methodologies , Architectures , and Reviews. In *Hybrid evolutionary algorithms* (Vol. 17, pp. 1–17). Springer Berlin Heidelberg.
- Guglielmo, G. D., Guglielmo, L. D., Fummi, F., & Pravadelli, G. (2011, March). Efficient Generation of Stimuli for Functional Verification by Backjumping Across Extended FSMs. *Journal of Electronic Testing*, 27(2), 137–162.
- Guo, Q., Hierons, R. M., Harman, M., & Derderian, K. (2004). Computing Unique Input / Output Sequences Using Genetic Algorithms. In *Third international workshop on formal approaches to testing of software* (pp. 164–177). Springer Berlin Heidelberg.
- Guo, Q., Hierons, R. M., Harman, M., & Derderian, K. (2005). Constructing Multiple Unique Input / Output Sequences Using Metaheuristic Optimisation Techniques FSMs based Testing. *IEE Proceedings-Software*, 152(3), 1–24.
- Hamon, G., De Moura, L., & Rushby, J. (2004). Generating efficient test sets with a model checker. In *Proceedings of the second international conference on software engineering and formal methods. sefm 2004* (pp. 261–270). IEEE.
- Hänsel, J., Rose, D., Herber, P., & Glesner, S. (2011, March). An Evolutionary Algorithm for the Generation of Timed Test Traces for Embedded Real-Time Systems. In *2011 fourth ieee international conference on software testing, verification and validation* (pp. 170–179). IEEE.

- Harman, M. (2007). The Current State and Future of Search Based Software Engineering. In *2007 future of software engineering, ieee computer society* (pp. 342–357). IEEE Computer Society.
- Harman, M., Hu, L., Hierons, R., Wegener, J., Sthamer, H., Roper, M., & Society, I. C. (2004). Testability Transformation. *IEEE Transactions on Software Engineering*, *30*(1), 3–16.
- Harman, M., Jia, Y., Krinke, J., Langdon, W. B., Petke, J., & Zhang, Y. (2014). Search based software engineering for software product line engineering : a survey and directions for future work. In *Proceedings of the 18th international software product line conference-volume 1* (pp. 5–18). ACM.
- Harman, M., Jia, Y., & Zhang, Y. (2015). Achievements , open problems and challenges for search based software testing. In *Software testing, verification and validation* (pp. 1–12). IEEE.
- Harman, M., & Jones, B. F. (2001, December). Search-based software engineering. *Information and Software Technology*, *43*(14), 833–839.
- Harman, M., Lakhota, K., & McMinn, P. (2007). A Multi Objective Approach To Search Based Test Data Generation. In *The 9th annual conference on genetic and evolutionary computation (gecco 07)* (pp. 1098–1105). London, England, United Kingdom: ACM.
- Harman, M., Mansouri, S. A., & Zhang, Y. (2009). *Search Based Software Engineering : A Comprehensive Analysis and Review of Trends Techniques and Applications* (Tech. Rep.). Department of Computer Science, Kings College London.
- Harman, M., & McMinn, P. (2010, March). A Theoretical and Empirical Study of Search-Based Testing: Local, Global, and Hybrid Search. *IEEE Transactions on Software Engineering*, *36*(2), 226–247.
- Hemmati, H., Arcuri, A., & Briand, L. (2010). Reducing the Cost of Model-Based Testing through Test Case Diversity Similarity-based Test Case Selection. In *International conference on testing software and systems* (pp. 63–78).
- Hermadi, I., Lokan, C., & Sarker, R. (2014, April). Dynamic stopping criteria for search-based test data generation for path testing. *Information and Software Technology*, *56*(4), 395–407.

- Holt, N. E., Briand, L. C., & Torkar, R. (2014, August). Empirical evaluations on the cost-effectiveness of state-based testing: An industrial case study. *Information and Software Technology*, 56(8), 890–910.
- Huang, R., Liu, H., Xie, X., & Chen, J. (2015, November). Enhancing mirror adaptive random testing through dynamic partitioning. *Information and Software Technology*, 67, 13–29.
- IEEE Standard Glossary of Software Engineering Terminology*. (n.d.). IEEE.
- Ipate, F., & Lefticaru, R. (2008). Genetic Model based Testing : a Framework and a Case Study. *Romanian Journal of Information Science and Technology*, 11(3), 209–227.
- Iqbal, M. Z., Arcuri, A., & Briand, L. (2012a). Combining Search-Based and Adaptive Random Testing Strategies for Environment Model-Based Testing of Real-Time Embedded Systems. In *International conference on testing software and systems* (pp. 136–151).
- Iqbal, M. Z., Arcuri, A., & Briand, L. (2012b). Empirical Investigation of Search Algorithms for Environment Model-Based Testing Of Real-Time Embedded Software. In *the 2012 international symposium on software testing and analysis - issta 2012* (p. 199). Minneapolis, MN, USA: ACM.
- Jin, C., & Jin, S.-W. (2016, March). Parameter optimization of software reliability growth model with S-shaped testing-effort function using improved swarm intelligent optimization. *Applied Soft Computing*, 40, 283–291.
- Joachim Wegener, O. B. (2004). Evaluation of Different Fitness Functions for the Evolutionary Testing of an Autonomous Parking System. In *Genetic and evolutionary computation* (pp. 1400–1412).
- Kalaji, A., Hierons, R. M., & Swift, S. (2009). A Search-Based Approach for Automatic Test Generation from Extended Finite State Machine (EFSM). In *2009 testing: Academic and industrial conference - practice and research techniques* (pp. 131–132). IEEE.
- Kalaji, A. S., Hierons, R. M., & Swift, S. (2009, April). Generating Feasible Transition Paths for Testing from an Extended Finite State Machine (EFSM). In *2009 international conference on software testing verification and validation* (pp. 230–239). IEEE.

- Kalaji, A. S., Hierons, R. M., & Swift, S. (2011, December). An integrated search-based approach for automatic testing from extended finite state machine (EFSM) models. *Information and Software Technology*, 53(12), 1297–1318.
- Kao, Y.-T., & Zahara, E. (2008, March). A hybrid genetic algorithm and particle swarm optimization for multimodal functions. *Applied Soft Computing*, 8(2), 849–857.
- Kapfhammer, G. M., McMinn, P., & Wright, C. J. (2013, March). Search-Based Testing of Relational Schema Integrity Constraints Across Multiple Database Management Systems. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation* (pp. 31–40). IEEE.
- Khurana, N., & Chillar, R. (2015). Test Case Generation and Optimization using UML Models and Genetic Algorithm. *Procedia Computer Science*, 57, 996–1004. Retrieved from <http://linkinghub.elsevier.com/retrieve/pii/S1877050915020311> doi: 10.1016/j.procs.2015.07.502
- Kim, Y.-H., & Moon, B.-R. (2002). Visualization of the Fitness Landscape, a Steady-State Genetic Search, and Schema Traces. In *Proceedings of the genetic and evolutionary computation conference* (p. 686). San Francisco: MorganKaufmann Publishers.
- Kitchenham, B. (2004). *Procedures for Performing Systematic Reviews* (Vol. 33; Tech. Rep.). Keele, UK, Keele University.
- Kitchenham, B. A., Pfleeger, S. L., Pickard, L. M., Jones, P. W., Hoaglin, D. C., Emam, K. E., & Rosenberg, J. (2002). Preliminary Guidelines for Empirical Research in Software Engineering. *IEEE Transactions on Software Engineering*, 28(8), 721–734.
- Korel, B. (1990). Automated Software Test Data Generation. *IEEE Transactions on Software Engineering*, 16(8), 870–879.
- Kruse, P. M., Wegener, J., & Wappler, S. (2009). A highly configurable test system for evolutionary black-box testing of embedded systems. In *the 11th annual conference on genetic and evolutionary computation - gecco '09* (pp. 1545–1552). Montreal, Quebec, Canada: ACM.
- Kruse, P. M., Wegener, J., & Wappler, S. (2010, May). A Cross-Platform Test System for Evolutionary Black-Box Testing of Embedded Systems. *ACM Sigevolution*, 5(1), 3–9.

- Kumari, A. C., Srinivas, K., & Gupta, M. P. (2013). Software Requirements Optimization Using Multi-Objective Quantum-Inspired Hybrid Differential Evolution. In *Evolve-a bridge between probability, set oriented numerics, and evolutionary computation ii* (pp. 107–120). Springer Berlin Heidelberg.
- Lamancha, B. P., Polo, M., Caivano, D., Piattini, M., & Visaggio, G. (2013, February). Automated generation of test oracles using a model-driven approach. *Information and Software Technology*, 55(2), 301–319.
- Lefticaru, R., & Ipatu, F. (2007, September). Automatic State-Based Test Generation Using Genetic Algorithms. In *Ninth international symposium on symbolic and numeric algorithms for scientific computing (synasc 2007)* (pp. 188–195). IEEE.
- Lefticaru, R., & Ipatu, F. (2008a). A Comparative Landscape Analysis of Fitness Functions for Search-Based Testing. In *2008 10th international symposium on symbolic and numeric algorithms for scientific computing* (pp. 201–208). IEEE.
- Lefticaru, R., & Ipatu, F. (2008b, April). Functional Search-based Testing from State Machines. In *2008 international conference on software testing, verification, and validation* (pp. 525–528). IEEE.
- Li, B.-L., Li, Z.-s., Qing, L., & Chen, Y.-H. (2007, December). Test Case Automate Generation from UML Sequence Diagram and OCL Expression. In *2007 international conference on computational intelligence and security (cis 2007)* (pp. 1048–1052). Ieee. Retrieved from <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4415508> doi: 10.1109/CIS.2007.150
- Li, J., Bao, W., Zhao, Y., Ma, Z., & Dong, H. (2009, June). Evolutionary Generation of Unique Input/Output Sequences for Class Behavioral Testing. *Computers & Mathematics with Applications*, 57(11-12), 1800–1807.
- Li, N., Li, F., Offutt, J., & Va, F. (2012). Better Algorithms to Minimize the Cost of Test Paths. In *2012 ieee fifth international conference on software testing, verification and validation better* (pp. 280–289). IEEE.
- LI, Y.-l., LI, R., LI, R.-f., & ZHANG, Y. (2009). Method for generating test suite based on user-defined faults in EFSM model [J]. *Application Research of Computers*, 9, 055.
- Lindlar, F., & Windisch, A. (2010, September). A Search-Based Approach to Functional Hardware-in-the-Loop Testing. In *2nd international symposium on search based*

software engineering (pp. 111–119). IEEE.

Lindlar, F., Windisch, A., & Wegener, J. (2010, April). Integrating Model-Based Testing with Evolutionary Functional Testing. In *2010 third international conference on software testing, verification, and validation workshops* (pp. 163–172). IEEE.

Luo, L. (2001). *Software testing techniques* (Tech. Rep.). Pittsburgh, USA: Institute for Software Research International Carnegie Mellon University.

Mani, N., Garousi, V., & Far, B. H. (2010, August). Search-Based Testing of Multi-Agent Manufacturing Systems for Deadlocks Based on Models. *International Journal on Artificial Intelligence Tools*, 19(04), 417–437.

Mansour, N., & Salame, M. (2004). Data Generation for Path Testing. *Software Quality Journal*, 121–136.

Marchetto, A., & Tonella, P. (2009, May). Search-Based Testing of Ajax Web Applications. In *2009 1st international symposium on search based software engineering* (pp. 3–12). IEEE.

Marchetto, A., & Tonella, P. (2010, December). Using search-based algorithms for Ajax event sequence generation during testing. *Empirical Software Engineering*, 16(1), 103–140.

Mark Harman, L. H., Hierons, Robert, A. B., Sthamer, & Harmen. (2002). Improving Evolutionary Testing by Flag Removal. In *Proceedings of the 2002 conference on genetic and evolutionary computation* (pp. 1359–1366). New York, USA: Morgan Kaufmann Publishers.

McMinn, P. (2004). Search-based software test data generation: A survey. *Software Testing Verification and Reliability*, 14, 105–156.

McMinn, P., Shahbaz, M., Stevenson, M., & Court, R. (2006). Search-Based Test Input Generation for String Data Types Using the Results of Web Queries. *Software Testing Verification and Reliability*, 16, 175–203.

Miller, W., & Spooner, D. L. (1976). Automatic Generation of Floating-Point Test Data. *IEEE Transactions on Software Engineering*(3), 223–226.

- Mohalik, S., Gadkari, A. A., Yeolekar, A., Shashidhar, K. C., & Ramesh, S. (2014). Automatic test case generation from Simulink/Stateflow models using model checking. *Software Testing, Verification and Reliability*, 24(2), 155–180.
- Mouchawrab, S., Briand, L. C., Labiche, Y., & Penta, M. D. (2011). Assessing , Comparing , and Combining State Machine-Based Testing and Structural Testing : A Series of Experiments. *IEEE Transactions on Software Engineering*, 37(2), 161–187.
- Nguyen, C. D., Miles, S., Perini, A., Tonella, P., Harman, M., & Luck, M. (2011, May). Evolutionary testing of autonomous software agents. *Autonomous Agents and Multi-Agent Systems*, 25(2), 260–283.
- Nilsson, R., Offutt, J., & Mellin, J. (2006, October). Test Case Generation for Mutation-based Testing of Timeliness. *Electronic Notes in Theoretical Computer Science*, 164(4), 97–114.
- Núñez, A., Merayo, M. G., Hierons, R. M., & Núñez, M. (2012, July). Using Genetic Algorithms to Generate Test Sequences For Complex Timed Systems. *Soft Computing*, 17(2), 301–315.
- Offutt, J., Liu, S., Abdurazik, A., & Ammann, P. (2003). Generating test data from statebased specifications. *Software testing, verification and reliability*, 13(1), 25–53.
- Oh, J., Harman, M., & Yoo, S. (2011). Transition Coverage Testing for Simulink/Stateflow Models using Messy Genetic Algorithms. In *Proceedings of the 13th annual conference on genetic and evolutionary computation - gecco '11* (p. 1851). Dublin, Ireland: ACM.
- Peleska, J., Honisch, A., Lapschies, F., & Helge, L. (2011). A Real-World Benchmark Model for Testing Concurrent Real-Time Systems in the Automotive Domain. In *Testing software and systems* (pp. 146–161). Springer Berlin Heidelberg.
- Pender, T. (2003). *UML Bible*. John Wiley & Sons.
- Perry, D. E., Sim, S. E., & Easterbrook, S. M. (2004). Case Studies for Software Engineers. In *26th international conference on software engineering*. IEEE Computer Society.

- Prelguskas, J., & Bareisa, E. (2012). Generating Unit Tests for Floating Point Embedded Software using Compositional Dynamic Symbolic Execution. *Elektronika ir Elektrotechnika*, 6(6), 19–22.
- Pretschner, A., & Philipps, J. (2005). 10 Methodological Issues in Model-Based Testing. In *Model-based testing of reactive systems*. (pp. 281–291). Springer Berlin Heidelberg.
- Roger, F., & Korel, B. (1995). Software Test Data Generation using the Chaining Approach. In *Proceedings of the IEEE international test conference on driving down the cost of tes* (pp. 703–709). IEEE Computer Society.
- Runeson, P., & Höst, M. (2008, December). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2), 131–164.
- Saeed, A., Hamid, S. H., & Sani, A. A. (2016). Empirical Analysis of Search-based Techniques and Model Driven Approach in Model-based Testing: Industrial Case Studies. *International Journal of Software Engineering and Knowledge Engineering*.
- Saeed, A., Hamid, S. H. A., & Mustafa, M. B. (2016). The experimental applications of search-based techniques for model-based testing: Taxonomy and systematic literature review. *Applied Soft Computing*.
- Sarma, M., & Mall, R. (2009, February). Automatic generation of test specifications for coverage of system state transitions. *Information and Software Technology*, 51(2), 418–432. Retrieved from <http://linkinghub.elsevier.com/retrieve/pii/S0950584908000700> doi: 10.1016/j.infsof.2008.05.002
- Schoenauer, M., & Xanthakis, S. (1993). Constrained GA optimization. In *the 5th international conference on genetic algorithms (icga 93)* (pp. 573–580).
- Sharma, C., Sabharwal, S., & Sibal, R. (2013). A Survey on Software Testing Techniques using Genetic Algorithm. *International Journal of Computer Science Issues*, 10(1), 381–393.
- Shelburg, J., Kessentini, M., & Tauritz, D. R. (2013). Regression Testing for Model Transformations : A Multi-objective Approach. In *Search based software engineering* (pp. 209–223).

- Shirole, M. (2011). Generation of Improved Test Cases from UML State Diagram Using Genetic Algorithm. In *Proceedings of the 4th india software engineering conference* (pp. 125–134). Thiruvananthapuram, Kerala, India: ACM.
- Shirole, M., & Kumar, R. (2010). A Hybrid Genetic Algorithm Based Test Case Generation. In *Contemporary computing* (pp. 53–63).
- Sthamer, H., & Morgannwg, P. (1995). *The Automatic Generation of Software Test Data Using Genetic Algorithms* (Unpublished doctoral dissertation). University of Glamorgan.
- Straub, J., & Huber, J. (2013, May). A Characterization of the Utility of Using Artificial Intelligence to Test Two Artificial Intelligence Systems. *Computers*, 2(2), 67–87.
- Swarup Mohalik, Ambar A. Gadkari, Anand Yeolekar, K. S., & Ramesh, S. (2014). Automatic test case generation from Simulink/Stateflow models using model checking. *Software Testing, Verification and Reliability*, 24(2), 155–180.
- Tasharofi, S., Ansari, S., & Sirjani, M. (2006). Generating Test Cases for Constraint Automata. In *Formal methods and software engineering* (pp. 478–493). Springer-Verlag Berlin Heidelberg.
- Tillmann, N., & Halleux, J. D. (2014). Transferring an Automated Test Generation Tool to Practice : From Pex to Fakes and Code Digger. In *29th acm/ieee international conference on automated software engineering (ase)* (pp. 385–396).
- Timo Mantere, J. A. (2005). Evolutionary software engineering, a review. *Applied Soft Computing*, 5(3), 315–331.
- Tonella, P. (2004). Evolutionary Testing of Classes. In *Proceedings of the 2004 acm sigsoft iinternational symposium on software testing and analysis (issta 04)* (pp. 119–128). Boston, Massachusetts, USA.: ACM.
- UNISIG. (2012). *ERTMS/ETCS SystemRequirements Specification* (Vol. Subset-026).
- Utting, M., & Legiard, B. (2010). *Practical model-based testing: a tools approach*. Morgan Kaufmann.
- Utting, M., Pretschner, A., & Legiard, B. (2011). A taxonomy of model-based testing

approaches. *Software Testing, Verification and Reliability*, 22(5), 297–312.

Varshney, S., & Mehrotra, M. (2013, July). Search based software test data generation for structural testing. *ACM SIGSOFT Software Engineering Notes*, 38(4), 1.

Victor, R. B. (1992). *software modeling and measurment.pdf*.

Vishal, V., Kovacioglu, M., Kherazi, R., & Mousavi, M. R. (2012, November). Integrating Model-Based and Constraint-Based Testing Using SpecExplorer. In *2012 IEEE 23rd international symposium on software reliability engineering workshops* (pp. 219–224). IEEE.

Vos, T. E. J., Lindlar, F. F., Wilmes, B., Windisch, A., Baars, A. I., Kruse, P. M., . . . Wegener, J. (2012, February). Evolutionary functional black-box testing in an industrial setting. *Software Quality Journal*, 21(2), 259–288.

Wegener, J., & Kruse, P. M. (2009, May). Search-Based Testing with in-the-loop Systems. In *2009 1st international symposium on search based software engineering* (pp. 81–84). IEEE.

Weiß leder, S., & Schlingloff, B.-h. (2008). Deriving Input Partitions from UML Models for Automatic Test Generation. In *International conference on model driven engineering languages and systems* (pp. 151–163). Springer Berlin Heidelberg.

Wenzel, I., Kirner, R., Rieder, B., & Puschner, P. (2008). Measurement-Based Timing Analysis . In *Leveraging applications of formal methods, verification and validation* (pp. 430–444). Berlin and Heidelberg: Springer.

White, D. R., Arcuri, A., & Clark, J. a. (2011, August). Evolutionary Improvement of Programs. *IEEE Transactions on Evolutionary Computation*, 15(4), 515–538.

Wilmes, B., & Windisch, A. (2010, April). Considering Signal Constraints in Search-Based Testing of Continuous Systems. In *2010 third international conference on software testing, verification, and validation workshops* (pp. 202–211). IEEE.

Wilson, B. (1994). *EU-Rent car rentals case study* (Tech. Rep.). Model Systems & Brian Wilson Associates.

Windisch, A. (2010). Search-Based Test Data Generation from Stateflow Statecharts. In

Proceedings of the 12th annual conference on genetic and evolutionary computation (pp. 1349–1356). Portland, Oregon, USA: ACM.

- Windisch, A., & Al Moubayed, N. (2009). Signal Generation for Search-Based Testing of Continuous Systems. In *2009 international conference on software testing, verification, and validation workshops* (pp. 121–130). IEEE.
- Woehrle, M. (2012, April). Search-Based Stress Testing of Wireless Network Protocol Stacks. In *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation* (pp. 794–803). IEEE.
- Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslé, A. (2012). *Experimentation in Software Engineering*. Springer Science & Business Media.
- Wong, S., Ooi, C. Y., Hau, Y. W., Marsono, M. N., & Shaikh-husin, N. (2013). Feasible Transition Path Generation for EFSM-based System Testing. In *2013 IEEE International Symposium on Circuits and Systems (ISCAS)* (pp. 1724–1727).
- Xanthakis, S., Ellis, C., Skourlas, C., Le Gall, A., Katsikas, S., & Karapoulios, K. (1992). Application of genetic algorithms to software testing. In *Proceedings of the 5th international conference on software engineering and applications* (pp. 625–636). Toulouse, France.
- Xiao, M., El-Attar, M., Reformat, M., & Miller, J. (2006, November). Empirical evaluation of optimization algorithms when used in goal-oriented automated test data generation techniques. *Empirical Software Engineering*, 12(2), 183–239.
- Yang, R., Chen, Z., Xu, B., Wong, W. E., & Zhang, J. (2011, November). Improve the Effectiveness of Test Case Generation on EFSM via Automatic Path Feasibility Analysis. In *2011 IEEE 13th International Symposium on High-Assurance Systems Engineering* (pp. 17–24). IEEE.
- Yano, T., Martins, E., & de Sousa, F. L. (2010, April). Generating Feasible Test Paths from an Executable Model Using a Multi-objective Approach. In *2010 Third International Conference on Software Testing, Verification, and Validation Workshops* (pp. 236–239). IEEE.
- Yano, T., Martins, E., & de Sousa, F. L. (2011, March). MOST: A Multi-objective Search-Based Testing from EFSM. In *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops* (pp. 164–173). IEEE.

- Yano, T., Martins, E., & Sousa, F. L. D. (2011). A Multi-Objective Evolutionary Algorithm to Obtain Test Cases With Variable Lengths. In *13th annual conference on genetic and evolutionary computation* (pp. 1875–1882).
- Zhan, Y., & Clark, J. (2004). Search Based Automatic Test-Data Generation at an Architectural Level. In *Genetic and evolutionary computation gecco 2004* (pp. 1413–1424). Springer Berlin / Heidelberg.
- Zhan, Y., & Clark, J. A. (2005). Search-based Mutation Testing for Simulink Models. In *Proceedings of the 2005 conference on genetic and evolutionary computation - gecco '05* (p. 1061). Washington,DC, USA: ACM.
- Zhan, Y., & Clark, J. A. (2006). The State Problem for Test Generation in Simulink. In *In proceedings of the 8th annual conference on genetic and evolutionary computation (gecco 06)* (pp. 1941–1948).
- Zhan, Y., & Clark, J. A. (2008, February). A Search-based Framework for Automatic Testing of MATLAB/Simulink Models. *The Journal of Systems and Software*, 81(2), 262–285.
- Zhang, J. (2005). Constraint Solving and Symbolic Execution. In *Working conference on verified software: Theories, tools, and experiments* (pp. 539–544). Springer Berlin / Heidelberg.
- Zhang, J., Chen, X., & Wang, X. (2004). Path-oriented test data generation using symbolic execution and constraint solving techniques. In *Proceedings of the second international conference on software engineering and formal methods*. IEEE.
- Zhao, R., Harman, M., & Li, Z. (2010, April). Empirical Study on the Efficiency of Search Based Test Generation for EFSM Models. In *2010 third international conference on software testing, verification, and validation workshops* (pp. 222–231). IEEE.