Cost and Effectiveness of Search-based Techniques for Model-based Testing: An Empirical Analysis

Aneesa Saeed
Software Engineering, Faculty of Computer Science and Information Technology, University of Malaya, Kuala Lumpur, Malaysia, aneesa.saeed@siswa.um.edu.my

Siti Hafizah Ab Hamid
Software Engineering, Faculty of Computer Science and Information Technology, University of Malaya, Kuala Lumpur, Malaysia, sitihafizah@um.edu.my

Model-based testing (MBT) gains interest in industry and academia due to its provision of systematic, automated and comprehensive testing. The challenge in MBT is to generate optimal test data to execute the test cases. Recently, researchers have successfully applied search-based techniques (SBTs) by automating the search for an optimal set of test data at reasonable cost compared to other more expensive techniques. In real complex systems, effectiveness and cost of SBTs for MBT in industrial context are little known. The objective of this study is to empirically evaluate the cost and effectiveness of SBTs for MBT on industrial case studies. We applied a model-driven approach and SBTs to automatically generate executable feasible test cases. The results show that the model-driven approach generated high number of infeasible test cases with less time while genetic algorithm (GA) and simulating annealing (SA) outperformed significantly random search (RS) with high generation time. We concluded that local SBTs are more appropriate to generate test data when the type of the constraints is simple. Current work on analyzing cost and effectiveness on SBTs for MBT opens possible enhancement to the model-driven approach to detect the infeasible paths and SBTs to achieve optimal success rate.

Keywords: Software Testing; Model Transformation; Model-Based Testing; Search-Based Techniques; Model Driven ; Test Case Generation

1. Introduction

Model-based testing (MBT) automates the test case generation and execution in order to reduce software testing cost. The lower the cost of test case generation, the lower the cost of the software testing process and software development will be. The cost of software testing consumes almost half of the entire software development cost [1]. Recent industry projects and academic studies focused on MBT [2, 3, 4, 5, 6, 7, 8, 9, 10] and showed successful results. One of various possible input models for MBT is state machine models which widely utilized to model a great number of today's embedded systems [11, 4, 3, 12]. Two challenges in applying MBT
for state machine models are generating concrete test cases and generating optimal
test data for executing concrete test cases.

For the first challenge of concrete test cases generation, several model-based
tools and approaches have been proposed for state machines models [13, 14, 15],
but at least one drawback has been found on each of them. They do not support
well-established modeling, and unable modify them to various needs and contexts.
For instance, practical constraints able to evolve, such as the test-script language a
company works with. These drawbacks motivated researchers to proposed extensible
and configured model-driven approach (model transformations) to generate concrete
test cases [16] and test oracle [17] from UML models. Realistic evaluation of the cost
and effectiveness of model-driven approach on industrial applications was provided
in [18].

For the second challenge of test data generation, the latest endeavor is to deploy
search-based techniques (SBTs) to generate optimal test data that satisfy all the
transitions and constraints in the state model and this recently become a field of
interest as reported in [4, 19]. Studies applied SBTs for MBT show their significant
performance compared to other test data generation techniques. For example, stud-
ies concluded that SBTs outperform model checking when testing dynamic systems
[20] and embedded real-time systems [21]. Another study declared that generating
test data using model checkers is more expensive than using heuristic techniques
[22]. 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 [3].

Extensive empirical studies conducted to evaluate both
MBT and SBTs separately. Regarding the evaluation of MBT, extensive studies
[23, 18, 24, 25, 26, 27, 28, 29] focused on the context of UML state machine models.
In these studies, the used technique for generating test data is simple such as ran-
dom. With respect to evaluate SBTs for code-based test case generation, extensive
research has been conducted [30, 31, 32, 33, 34, 35, 36]. Regarding the evaluation of
SBTs for MBT, study [37] evaluated SBTs for industrial case studies in context
of evaluating different fitness functions using simulation models. Tanja et al. [3]
evaluated the use of SBTs for functional MBT by using EvoTest tool (a code-based
EV tool) with auto-generated C code from SIMULINK models. Study [38] evalu-
ated the hill climbing (HC) for embedded system modeled using auto-generated C
code from Function Block Diagrams. All the research converted the models into
an auto-code and applied SBTs as code-based domain. Although, evaluating MBT
and SBTs have been extensively evaluated separately, no empirical studies focused
on evaluating the cost and effectiveness of both model-based approach for concrete
test case generation with SBTs for test data generation in industrial context.

The objective of this study is to empirically evaluate the cost and effectiveness
of SBTs for MBT within the context of industrial systems. We followed the em-
pirical case study method in [39, 40]. To evaluate SBTs for MBT, we developed
an extensible model-driven approach for automating the test case generation from
UML state machines and applied non-heuristic, global and local SBTs for test data generation for solving OCL constraints. The case study was conducted within the context of using two industrial public case studies which are taken from [41]. We investigated the all round trip coverage criteria(RTP) in model-driven approach, and analyzed the generated data by checking the success rate of solving all the test case constraints. The findings helps 1) the MBT research community wherein this empirical research apply model-driven architecture and constraint solving in MBT, and 2) search-based software testing community wherein it shows the cost and effectiveness of SBTs for MBT in industry context. Finally, the directions for future work are presented on SBTs for MBT.

The rest of this paper is organized as follows: section 2 provides background information about MBT while section 3 illustrates SBTs in test case generation domain, MBT and model-driven approach. The related empirical studies are described in section 9. Section 4 presents the methodology that we follow in this study whereas section 5 describes the execution of the study. Section 6 presents and discusses the results and finding. Threats to validity are addressed in section 8. Finally, section 10 presents summary, conclusion and future work.

Table 1. List of Acronyms.

<table>
<thead>
<tr>
<th>Term</th>
<th>Acronym</th>
<th>Term</th>
<th>Acronym</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model-based Testing</td>
<td>MBT</td>
<td>Search-based Techniques</td>
<td>SBTs</td>
</tr>
<tr>
<td>Unified Modeling Language</td>
<td>UML</td>
<td>Object Constraint Language</td>
<td>OCL</td>
</tr>
<tr>
<td>Hill Climbing</td>
<td>HC</td>
<td>Genetic Algorithm</td>
<td>GA</td>
</tr>
<tr>
<td>Simulating Annealing</td>
<td>SA</td>
<td>Random Search</td>
<td>RS</td>
</tr>
<tr>
<td>All Round-Trip</td>
<td>RTP</td>
<td>All Transitions</td>
<td>AT</td>
</tr>
<tr>
<td>All Transitions Pairs</td>
<td>ATP</td>
<td>Full Predicate</td>
<td>FP</td>
</tr>
<tr>
<td>Ceiling Speed Monitoring</td>
<td>CSM</td>
<td>Turn Indicator System</td>
<td>TIS</td>
</tr>
<tr>
<td>Software Under Test</td>
<td>SUT</td>
<td>Evolutionary Algorithm</td>
<td>EV</td>
</tr>
</tbody>
</table>

2. Model based testing and Model-Driven Approach

Model-based Testing (MBT) produces executable test cases by consistently analyzing the behavioral design models (abstract representation) of a software system by following a test strategy. To fully automate MBT, three tasks are required: constructing models for testing, defining a suitable testing strategy and adequacy criteria, and generating test data for executing test cases as shown in Fig. 1. As one of various possible input models for MBT, state machines are widely utilized to model the behavior of the most critical and complex system components that exhibit state-driven behavior [23], which called state-based testing. Object-oriented methodologies recommend modeling such as components with state models for the purpose of test automation [24]. Coverage criteria are measures used to describe the degree to which the source models of a SUT is covered in the generated test
cases. Four coverage criteria are 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). RTP is a compromise between the weak AT and the more expensive ATP criteria [23, 18]. The RTP strategy requires that all paths in a state machine model that starts and finishes with the same state must be covered. A test tree is elevated by depth-first search of the state machine to cover all such paths. The test tree contains edges and nodes with respect to transitions and states in a state machine model. The test tree obtained by the RTP strategy is called a transition tree. In the transition 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.

One of the MBT approaches concentrate on generating test model directly from source drawn models using model-driven architecture concept, called model-driven approach. Using a model-driven approach, the abstract test cases are automatically generated using models extracted from software models through serial of model transformations. Model transformations process contains a source metamodel, a target metamodel, and a set of transformation rules that characterize how the el-

---

**Fig. 1.** The process of model-based testing.
lements from the source metamodel are matched into elements of the target metamodel as seen in Fig2. Therefore, the advantage of the MD 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. Specifically, it can extensible to take any type of input models and produce any type of output files based on scripting language and the used metamodels. The challenge of applying this approach is that new effort is needed to develop the transformation rules.

Fig. 2. The process of model-driven approach.

3. Search-based Software Testing

Search based software engineering (SBSE) tries to solve various problems in the software engineering domain by reformulating these problems as search problems [42]. Search-based software testing (SBST); 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 [42]. Due to the possible generated test data is 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 [34, 43], functional testing [44], stress testing [45], and non-functional properties testing [46].

For more information on SBST, extensive review studies conducted as summarized in the following: Anand et al. [47] conducted a 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 SBSTs separately, as a part of the survey. Comprehensive studies [48, 49] have reviewed generally SBSTs for test case generation and found that most studies on SBST focused on code-based testing. Haraman et al. [50] presented the achievement, challenges and open problems in SBST domain and concluded that there is a rapid growth of interest in SBST. Increasing SBST interest for practitioners due to the results are comparable to human competence [51], and are real-world results. For example, EvoSuite SBST tool [52] has been successfully utilized to automatically generate test case for open source projects randomly selected from open source repositories [53]. SBST is presently adequately mature that it has used for industrial application rather than laboratory study to, for example at Daimler [54, 3], and Microsoft [55].

4. Case Study Design

This analysis follows the Empirical Case Study method in software engineering [39, 40].

4.1. Research Objectives

The main purpose of this study was to evaluate the cost and effectiveness of SBSTs and model-driven approach. The cost and effectiveness were chosen due to they are common in test case generation domain and also to give an evidence of applicability of using SBSTs and MBT in industrial context. We followed the GQM template in [56] 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 cost and effectiveness of generating test data using SBSTs?

4.2. Case and Subject Selection

To ensure that the cases studies were real-world complex industrial systems, this paper used case studies by Daimler and European Vital Computer, TIS and CSM [41]. 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 most important factor for selection was that these case studies described concurrent real-time behavior of automotive applications and directly derived from
industrial applications. The two case studies are publicly available as benchmark of real time embedded systems [41]. These models applied UML state machines, class diagrams, and C language for constraint.

4.2.1. Daimlers System: Turn Indicator

Turn Indicator System (TIS) is the system that controls 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 test 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 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 [41].
4.2.2. European Train Control System: Ceiling Speed Monitoring

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 presence of an 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 [57]. Speed and distance monitoring is covered in one of the functional category of the EVC, 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.” [58] (3.13.1.1). 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 [58] (3.13.10.1.2), and only one out of these three functions is active at a point in time. The case study 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.

4.3. Data collection procedures

Six metrics were used to measure the cost and effectiveness of model-driven approach and SBTs from [24, 18]. These metrics are common in test case generation domain. We followed the GQM template in [56] to derive proper metrics that embody the study objective and questions:

We measure the cost in terms of:

- 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.

Effectiveness is measured by:

- State and transition coverage: The number of covered states and transitions 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.

We collected timing data 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.
5. Case Study Execution

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

5.1. Preparation of Input Models

The CSM case study was modeled using Papyrus while the TIS case study was modeled using EnterpriseArchitect. We flattened one state machine in the TIS case study manually because the flattening component did not implement. The summary of the CSM and TIS case studies are shown in Table 2. Note that the functionality as originally modeled has not been affected by the remodeling. The constraints in the both case studies were written in C++ language. We wrote the constraint in OCL. The number of guards in CSM case study is 12 while guards in TIS case study are 119. The number of clauses in each guard varies from one to four. The characteristics of the constraints are summarized in Table 3, wherein we provide 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>
<thead>
<tr>
<th>State-machine feature</th>
<th>CSM case study</th>
<th>TIS case study</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum level of hierarchy</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Number of submachine</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Number of simple states</td>
<td>12</td>
<td>118</td>
</tr>
<tr>
<td>Number of transitions (guarded)</td>
<td>17</td>
<td>164</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Conjuncted Clauses</th>
<th>Frequency (CSM)</th>
<th>Frequency (TIS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>-</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>38</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>51</td>
</tr>
</tbody>
</table>

5.2. SBTs for 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. We selected GA and SA SBTs due to they are the most commonly used global and local search algorithms in SBSE[48]. This section provides information about using GA and SA to generate test data for solving OCL constraints.

5.2.1. Representation of Problem

Each abstract test case has number of transition guards and state invariants as OCL constraints. One OCL constraint (P) represents as a set of Boolean clauses \( (C_1, C_2, \ldots, C_N) \) linked by Boolean operations \( (\text{and}, \text{or}, \text{implies}, \text{xor}, \text{not}) \). Each Boolean clause \( C_i \) is defined over a number of variables \( (V_{Ci1}; V_{Ci2}; \ldots; V_{Cij}) \). 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: \( \bigcup_{x=1}^{n} \bigcup_{y=1}^{m} (C_{xy}) \) 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.

5.2.2. Fitness Function Definition

We utilized the fitness function proposed in [4], which inspired from code-based testing in [49]. The fitness function is based on measuring the distance (so-called branch distance \( d() \)) as shown in equation (1). The \( d() \) returns 0 if the constraint is solved 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 [4].

\[
F(\text{TestCase}) = 1 - 1.05^{-d()} \tag{1}
\]

5.3. Test Case Generation and Execution

We implemented model-driven approach [16] as shown in Fig. 3. It shows the model-driven approach based on model-to model and model-to text transformations. First, transforming the input models into transition tree model based on RTP criterion using ATL model-to model transformation language. The developed transformation ATL rules need the source design UML state machine models and two meta models, including transition tree metamodel and UML 3.0 metamodel. The two metamodels are ecore files, in which the transition tree metamodel is taken from study [16] and developed again using plugin Eclipse Modeling Framework. The UML 3.0 metamodel is retrieved from eclipse. The output of this step is the generated test model (transition tree) as XML file, in which 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. Second, transforming the test model into executable test cases using MOFScript language, including traversing the test model (the transition tree) to get all paths
in the transition tree, and transforming each one path into one java based abstract
test case. This step needs MOFscript rules, 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 is one abstract test case. The generated test cases are then analyzed in order to
remove 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.

![Model Transformation Approach Architecture](image)

**Fig. 3.** The model transformation approach architecture used for model-based testing

The test data generator is based on three techniques, including random (non-
heuristic SBTs), genetic algorithm (GA) (global SBTs) and simulating annealing
(SA) (local SBTs) as aforementioned in the previous section. The configuration
of the SBTs is shown in Table 4. The used OCL evaluator is EyeOCL Software
to evaluate the generated data by satisfying the OCL constraints. To use OCL
evaluator, the class and object diagrams are 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 is used at runtime to check the
constraints in the test case script. For execution, each test case was invoked on a
new instance of the SUT.
Table 4. Configuration of search-based techniques.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fitness evaluations</td>
<td>2000 times</td>
</tr>
<tr>
<td>Number of iterations</td>
<td>1000</td>
</tr>
<tr>
<td>Crossover rate</td>
<td>0.75</td>
</tr>
<tr>
<td>Population size</td>
<td>100</td>
</tr>
<tr>
<td>Mutation</td>
<td>1/n, where n is the number of variables</td>
</tr>
</tbody>
</table>

6. Results and Discussion

Table 5 illustrates 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. Tables 6 and 7 present 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. Statistical t-test has been applied to analyze statistically the significant differences between the three techniques in Tables 8 and 9. Table 8 shows the result of the t-test on the success rate and Table 9 shows the result of the t-test on the generation time of the three SBTs.

6.1. Model-Driven approach

The result of this section is to answer RQ1. The evaluation metric related to the effectiveness of model-driven approach is 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 are measured by number of all generated abstract test cases and how much from them are feasible and not feasible. Table 5 shows significant performance 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 is taken minimum cost, the number of generated infeasible test cases is high in both case studies. Due to, the model-driven approach does not has any infeasibility detection approach. After the data has been generated by using GA, the time needed for executing the executable test cases is more in the TIS case study due to its size is large.

6.2. Test Data Generation

This section is to answer RQ2. The evaluation metrics for the cost is the generation time and for the effectiveness is the successful rate. From the Tables 6 and 7, we conclude SA and GA achieved better success rate than RS in both case studies. SA significantly outperformed the RS and GA in CSM while it’s 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 case study 2. GA and RS closely achieved the same success rate in CSM. GA greatly outperformed SA in test
Table 5. Result of model-driven approach.

<table>
<thead>
<tr>
<th>Evaluation Metrics</th>
<th>CSM Case Study</th>
<th>TIS Case Study</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generated abstract test cases</td>
<td>26</td>
<td>161</td>
</tr>
<tr>
<td>Generated feasible test cases</td>
<td>9</td>
<td>29</td>
</tr>
<tr>
<td>Generated infeasible test cases</td>
<td>17</td>
<td>132</td>
</tr>
<tr>
<td>State coverage</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Transition coverage</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Preparation time (Seconds)</td>
<td>0.5</td>
<td>1.5</td>
</tr>
<tr>
<td>Execution time (Seconds)</td>
<td>97</td>
<td>191</td>
</tr>
</tbody>
</table>

case numbers 12-17 and 28 with higher generation time. SA achieved 100 percent in the top seven test cases in both case studies, and GA obtained 100 percent in five test cases. From the results, we observed that SA performed efficiently when

table 6. The results of successful rate and generation time for each GA, SA and RS techniques in case study 1 (CSM).

<table>
<thead>
<tr>
<th>CSM Case Study</th>
<th>Successful Rate</th>
<th>Generation Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Case ID</td>
<td>No of constraints</td>
<td>GA RS GA RS SA</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0.21 0.28 0.7 95.12 10.36 110.52</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>0.19 0.2 0.75 64.12 22.56 70.43</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>0.31 0.5 0.8 102.35 14.16 71.41</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0.31 0.25 0.83 98.33 9.31 106.96</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>0.25 0.4 0.76 101.82 9.97 103.04</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>0.24 0.2 0.75 101.26 11.88 103.76</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>0.16 0.17 0.82 75.29 12.57 103.02</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>0.45 0.1 0.67 1.08 5.78 5.72</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>1.00 0.5 1.00 0.001 1.55</td>
</tr>
</tbody>
</table>

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). This is the situation of test cases of CSM case study. The test cases with higher number of constraints are the most difficult to solve by GA and RS as shown in tables 6 and 7. Specifically, the increasing number of conjuncted clauses in each test is, the decreasing success rate is. With respect to the cost, RS is the fastest SBTs comparing to the GA and SA, taking few seconds for generating data. In CSM case study, SA costs approximately more than GA in all the test cases except the test case number 3. However, the taken time by SA is waver with GA in the TIS case study. In conclusion, 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 the CSM case study and it has approximately the same performance and generation time of GA in TIS case study. Generally, the better success rate is, the more taken generation time is.

With respect to statistical check, the statistical paired t-test is carried out on
Table 7. The results of successful rate and generation time for each GA, SA and RS techniques in case study 2 (TIS)

<table>
<thead>
<tr>
<th>Test Case ID</th>
<th>No of constraints</th>
<th>Successful Rate</th>
<th>Generation Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>GA</td>
<td>RS</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0.81</td>
<td>0.2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0.99</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0.99</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>0.99</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>1</td>
<td>0.99</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>0.73</td>
<td>0.48</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>0.15</td>
<td>0.14</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>0.68</td>
<td>0.24</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>0.56</td>
<td>0.12</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>0.56</td>
<td>0.12</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>0.46</td>
<td>0.23</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>0.99</td>
<td>0.25</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>0.88</td>
<td>0.27</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>0.89</td>
<td>0.25</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>0.87</td>
<td>0.26</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>0.89</td>
<td>0.24</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>0.88</td>
<td>0.23</td>
</tr>
<tr>
<td>18</td>
<td>2</td>
<td>0.85</td>
<td>0.22</td>
</tr>
<tr>
<td>19</td>
<td>2</td>
<td>0.82</td>
<td>0.99</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>0.72</td>
<td>0.48</td>
</tr>
<tr>
<td>21</td>
<td>1</td>
<td>0.69</td>
<td>0.52</td>
</tr>
<tr>
<td>22</td>
<td>2</td>
<td>0.79</td>
<td>0.25</td>
</tr>
<tr>
<td>23</td>
<td>2</td>
<td>0.79</td>
<td>0.26</td>
</tr>
<tr>
<td>24</td>
<td>4</td>
<td>0.81</td>
<td>0.24</td>
</tr>
<tr>
<td>25</td>
<td>2</td>
<td>0.86</td>
<td>0.24</td>
</tr>
<tr>
<td>26</td>
<td>2</td>
<td>0.47</td>
<td>0.26</td>
</tr>
<tr>
<td>27</td>
<td>2</td>
<td>0.46</td>
<td>0.24</td>
</tr>
<tr>
<td>28</td>
<td>2</td>
<td>0.96</td>
<td>0.5</td>
</tr>
<tr>
<td>29</td>
<td>4</td>
<td>0.72</td>
<td>0.02</td>
</tr>
</tbody>
</table>

The distributions of the success rates and generation time of all the three. Table 8 shows the statistical difference based on success rates and Table 9 depicts the differences of the cost. Table 8 presents that p-values were very close to 0 in some distributions compressions in both case studies which refers to there is a intense statistical significant difference between the performance of three techniques. In CSM case study, the performance of SA is statistically superior than the others due to p-values of SA distributions comparisons are close to 0. In TIS case study, the p-value of SA vs GA shows that there is no significance between the performances of them while p-values of their comparisons with RS show that SA and GA are better than RS. From Table 9, the p-values of RS versus the others proofs that RS is faster than others, while SA and GA take approximately the same time for finding the solutions in both case studies.

For general overview, box plot diagram has been drawn for representing the success rates of RS, GA and SA. Fig. 4 illustrates the success rate of three SBTs based
Table 8. The results of the paired t-test based on successful rate.

<table>
<thead>
<tr>
<th>Pairs of Techniques</th>
<th>CSM Case study</th>
<th>TIS Case study 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>GA vs SA</td>
<td>9.02743E-05</td>
<td>0.237301152</td>
</tr>
<tr>
<td>GA vs RS</td>
<td>0.233288588</td>
<td>1.18483E-08</td>
</tr>
<tr>
<td>SA vs RS</td>
<td>5.64267E-07</td>
<td>1.73242E-09</td>
</tr>
</tbody>
</table>

Table 9. The results of the paired t-test based on generation time.

<table>
<thead>
<tr>
<th>Pairs of Techniques</th>
<th>CSM Case study</th>
<th>TIS Case study 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>GA vs SA</td>
<td>0.244773935</td>
<td>0.309263749</td>
</tr>
<tr>
<td>GA vs RS</td>
<td>0.000819209</td>
<td>6.40878E-07</td>
</tr>
<tr>
<td>SA vs RS</td>
<td>0.000752768</td>
<td>1.03843E-05</td>
</tr>
</tbody>
</table>

on the 38 test cases, 9 test cases of CSM case study, and 29 test cases of TIS case study. The figure shows that SA achieved better results than the others with average success rate 79 percent. GA outperformed RS with average success rate 67 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 SA and GA achieve roughly the same median success rates (77 and 76 percent) and RS exceeds a median success rate of 25 percent. We can also conclude that the lowest success rate of SA is roughly 40, whereas the tendency of the lowest success rates of GA and RS are towards to 0. The highest 25 percent of the SA, GA, and RS success rates are respectively 98, 88, and 48 percent, while the lowest 25 percent of their success rates are 69, 46, and 21 percent sequentially. Overall, all the values in the figure refers to the preference of SA compared to the RS and GA. The explanation of the difference between GA and SA performances is that SA is local search and GA is 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. Furthermore, 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.
7. Concluded Problems and Proposed Solutions

From the results, we concluded that the model-driven approach suffers from generating infeasible paths and SBTs still generating unsufficient data. In this section, we describe briefly proposed solutions that we are working in for detecting infeasible paths with complex construct of OCL constraints and for improving the performance of SBTs.

**Model-driven approach with infeasible path detection** : The method is based on model-to model and model-to text transformations and static analysis. First, transforming the input models into transition tree model based on RTP criterion using ATL model-to model transformation language. The developed transformation ATL rules need the source design UML state machine models and two meta models, including transition tree metamodel and UML 3.0 metamodel. The output of this step is the generated test model (transition tree) as XML file, in which 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. Second, transforming the test model into executable test cases using MOFScript language, including traversing the test model (the transition tree) to get all paths in the transition tree, and check the feasibility of each path using our proposed static analysis method. To detect infeasible paths that contain OCL constraint, a set of rules is proposed. These rules are based on penalty values proposed for different data types and relations in OCL. The proposed penalty values cover the relations of Integer, String, Real, Boolean, collection, tuples, and enumerations data types. The distance between the penalty values of OCL clauses is
calculated. The path is feasible when the distance is zero. Each feasible path is transformed into one java based abstract test case. This step needs MOFscript rules, feasibility detection java code, and the two metamodels (the transition tree models and class diagram of the SUT). The output is a set of java files (test cases) and each java file is one abstract test case.

Improving SBTs: The data is generated to satisfy whole constraints in each test case. The OCL constraints in the test case are analyzed to get the dependency between the constraints clauses and the variables. All the clauses related to one variable are gathered as a new constraint. A fitness function evolves itself based on error feedback is proposed to improve the performance of test data generator. The fitness function utilizes the concept of neural network, in which optimal data that satisfies all constraints is generated and regarded as desired output, and then the difference between the new generated test data and desired output is calculated as error feedback. The fitness function evolves itself based on the error feedback. The fitness function is utilized by SBTs to generate data to satisfy all the constraints.

8. Threats to Validity
The developing of the model-driven approach and the test data generation SBTs may be not professional as the original one. To address this, we used the same tools and transformation language as they used. Also, we used the existing library of SBTs (jmetal library) and the developed fitness function is the same (distance function) provided in the study [4].

The reliability threat defined in [59] that affects the robustness of this study is that unclear, detailed description of data collection may give different results in case of repeating the study experiments. We mitigated this threat 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, we reiterated the experiments 1000 times to decrease the likelihood that the obtained results caused by chance. In addition, statistical test has been conducted to analyze statistically the results.

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

9. Related Work
With respect of evaluating MBT, study [25] compared RTP with random testing, finding that the RTP is rationally effective at revealing faults, where in 11 percent of the faults were did not detect while 31 percent for random testing. To enhance the fault detection, Briand et al. [24] augmented RTP with category-partition testing
with increasing cost, while Mouchawrab et al. [26, 27] combined RTP with white-box testing. For improving cost and effectiveness of RTP, Briand et al. [28] investigated data-flow analysis when more than one transition tree could be generated. The finding of the study is that data-flow information was beneficial for selecting the best transition tree. Briand et al. [29] studied the RTP criterion with a concentration on how to enhance the cost and effectiveness of the fault-detection. They investigated data flow analysis based on OCL guard conditions and operation contracts with a view to revise the selection of a cost and effective test cases among alternatives. The results showed that using data-flow information can help in selecting a transition tree with the highest fault detection ability. Recent empirical analysis by Holt et al. [18] evaluated extensively the cost and effectiveness by studying the combined impact of four aspects: six coverage criterion (FP, ATP, RTP, AT, and path of length (LN2, LN3, and LN4), test oracle, test model and unspecified behavior (sneak paths). Their result was aligned with result of Briand et al study [23] regarding that RTP is a compromise coverage criterion between the poor criterion AT and the more expensive criterion ATP in terms of cost and effectiveness.

For evaluating SBTs for code-based test case generation field, empirical studies [31, 32, 33] compared EV with non-SBTs (random) and found that EV is more efficiently and obtains the highest coverage. It is worthy of study that the chosen SBTs should are persuasively better than random search. On other hand, empirical studies for comparing SBTs with other SBTs are summarized in the following: comparison between EV, HC, and SA for generating test data based on path coverage was done by [35] and founded that HC took less time than EV and SA while EV and SA covered more paths. Their results depended on eight functions of fewer than 86 lines of code with integer inputs. Xiao et al. [36] compared EV with SA based on condition-decision coverage, concluding that EV obtained steadily the best performance. The study evaluated based on small examples with limited complexity. Harman et al. [34] were analyzed theoretically and experimentally the performance of global SBTs (EV), local SBTs (HC) , and hybrid SBTs (memetic ) and concluding that local SBTs outperformed global SBTs and both pure local and global SBTs performed better that hybrid SBTs. This surprising results were because the search space of the case studies contained only a few local optima, a type of landscape in which local search can fulfill very efficiently. Recent empirical evaluation in 2015, interactive search-based software testing system was designed by [30] to operate in an industrial setting and also allow domain specialists to use their experience and intuition to interactively guide the search. This system used by a domain specialist unfamiliar with search-based techniques and required an effort in terms of training and generating test cases that complement existing approaches. From the reviewed empirical studies on SBTs for code-based test case generation field, non of them provides an analysis of the success rate of the generated data for industrial systems.

For evaluating SBTs for MBT, study [37] evaluated SBTs for industrial case studies in context of evaluating different fitness functions. Their result showed that
the area criterion function is better suited to directing the search towards collision
maneuvers than the distance criterion function. For evaluating the scalability and
applicability of SBTs in industrial context, Tanja et al. [3] evaluated the use of
SBTs for functional MBT by using EvoTest tool (an code-based SBT tool). The
auto-generated C code was generated from SIMULINK models. They compared the
performance of SBTs with manual test, and random test in term of the ability of
detecting errors. They concluded that the developing of fitness function required
long time and experience but the SBTs are scalable and applicable in an industrial
setting. Moreover, recent study [38] evaluated the SBTs for embedded system mod-
eled by Function Block Diagrams. They compiled the Function Block Diagrams into
C code. The modified condition/decision coverage criterion was used as basis of the
evaluation. For this particular system, HC performed well in some software units.

To conclude, non of the available evaluation studies on SBTs for MBT applied
model-based approach, an auto-code generated from the source models was utilized
as a concrete test case and applying SBTs for generating test data. Furthermore,
some reported research has evaluated SBTs for MBT, but the focus has mostly been
directed towards coverage and fault-detection effectiveness and was not concerned
with the cost of such testing. In addition, the used models did not develop using
widely accepted standard modeling language (e.g UML state machines with OCL
constraint). The results of evaluating SBTs for MBT are based on comparing SBTs
with RS. The majority of the results found in existing SBTs research are based on
non public available industrial case studies. Our study complements and extends
existing research on evaluating the cost and effectiveness of SBTs for MBT by:

- using public industrial embedded systems as subject.
- using model-driven approach for test case generation
- comparing three type of techniques global SBTs (GA), local SBTs (SA), and
  non-heuristic SBTs (random).
- studying the effectiveness and cost of using model-driven approach.
- studying the effectiveness and cost of using SBTs for test data generation

The contribution of our study is to provide an empirical analysis of both SBT and
model-driven approach using public industrial benchmark which is lacking in the
literature, in which the existing analysis studies were conducted for either SBTs
[31, 32, 33, 36, 34, 30] or MBT [25, 24, 26, 27, 28, 18] using non public industrial case
study.

10. Summary and Conclusion

This study provides empirical evaluation of the cost and effectiveness of MBT for
generating executable test cases and SBTs for generating test data based on two
industrial case studies. The MBT approach was 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). This study was motivated
by the lack of empirical evaluation studies in SBTs for MBT domain. Furthermore, the available evaluation studies on SBTs for MBT were based on an auto-code generated from the models. These studies primarily focused on coverage and the fault-detection effectiveness and ignored testing cost. The main goal was to achieve maximum realism and understand the interplay between MBT and SBTs. The comparison is exclusively based on real industrial case studies modeled using UML modeling language with its extension OCL. In our study, we considered six evaluation criteria that measured the cost and effectiveness of both model-driven approach and SBTs, comprising state, and transition coverage criteria, size of test cases, 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. From the results of SBTs, we 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. Our result of SA aligns to the result of SA in study [35], 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, we recommend further research to improve the existing model-driven approach for UML state machines with OCL by generating only feasible test cases and also to improve SBTs investigating on SBTs by enhancing the fitness function.

Acknowledgments
The authors acknowledge the support provided by University of Malaya Research Grant; RP006 B14HNE, ”Quantum Computing for Designing and Validation Procedure”.

References


2007).


