TCG
A Model-based Testing Tool for Functional and Statistical Testing

Laryssa Lima Muniz[1,2], Ubiratan S. C. Netto[1] and Paulo Henrique M. Maia[2]
[1]*Fundação Cearense de Meteorologia e Recursos Hídricos (Funceme), Fortaleza, Brazil*
[2]*Universidade Estadual do Ceará, Fortaleza, Brazil*

Keywords: Model-based Testing, Tool, Functional Testing, Statistical Testing.

Abstract: Model-based testing (MBT) is an approach that takes software specification as the base for the formal model creation and, from it, enables the test case extraction. Depending on the type of model, an MBT tool can support functional and statistical tests. However, there are few tools that support both testing techniques. Moreover, the ones that support them offer a limited number of coverage criteria. This paper presents TCG, a tool for the generation and selection of functional and statistical test cases. It provides 8 classic generation techniques and 5 selection heuristics, including a novel one called minimum probability of path.

1 INTRODUCTION

Model-based testing (MBT) is a technique for automatic generation of a test case set using models extracted from software artefacts (Binder, 1999). This approach stands for automating the whole test generation process and for reducing the costs for test design by up to 85 percent (Weissleder and Schlingloff, 2014). As this approach depends on the quality of the used model, a high level of accuracy in the model construction task is necessary (Beizer, 1995).

The automation of an MBT approach depends on three main elements (Dalal et al., 1999): (i) the model that describes the system behaviour; (ii) the test generation algorithm (test criteria); and (iii) tools that provide the necessary infrastructure for the test case generation. Regarding the model, the most used ones are the UML diagrams, like the sequence, use case and state diagrams, finite state machine based formalisms, statecharts, and labelled transition systems (LTS). To manipulate them, many tools have been developed that implement different test criteria, as shown by several surveys in the area (Shirole and Kumar, 2013; Shafique and Labiche, 2013; Aichernig et al., 2008). Since the test cases are generated from models instead of source code, MBT approaches are adequate for functional tests.

In addition, some work have proposed the use of probabilistic models, such as Markov chains, together with MBT techniques for statistical generation of test cases based on the system probability execution (Walton and Poore, 2000)(Feliachi and Le Guen, 2010), that also allows to estimate the reliability of a given system (Trammell, 1995)(Zhou et al., 2012). Supported by traditional software testing techniques, such as functional and structural testing, the statistical test comprises the application of statistics to the solution of software testing problems (Sayre and Poore, 1999).

The statistical test can be seen as an excellent complement to the existing testing techniques. It does not need to be used as a different technique, but as a one that aims at adding reliability to the other ones (Poore and Trammell, 1999). In this context, statistical tests should be supported by tools that promote all that testing process, such as MaTeLo (Feliachi and Le Guen, 2010) and JUMBL.

According to Utting and Legeard (2007), an MBT tool should support several kinds of criterion to give as much control over the generated tests as possible. Furthermore, the user should be able to combine different coverage criteria (Antoniol et al., 2002). However, this is not found in several MBT tools, as we show in Section 5. Only few tools support functional and statistical testing. Moreover, even the tools that allow both types of analysis usually offer few combinations of coverage criteria. This may prevent the user of choosing the most appropriate approach according to the system under test (SUT).

This paper introduces TCG, a free MBT tool for generation and selection of test cases for both functional and statistical tests. This is possible because

DOI: 10.5220/0005398604040411
In *Proceedings of the 17th International Conference on Enterprise Information Systems* (ICEIS-2015), pages 404-411
ISBN: 978-989-758-097-0

it was developed as a plugin for LoTuS [1], a graphical tool for software behavior modeling that allows the user to model both the non-probabilistic behavior using LTS, and the probabilistic one using probabilistic LTSs, which are similar to Discrete Time Markov Chain. The main contribution of this work is twofold: (i) providing a set of techniques for test cases generation and selection addressing both functional and statistical tests, which permits the user to use different criteria when analyzing distinct systems and to exercise the same system with different criteria, and (ii) implementing criteria that have not been found in other tools, including a new selection technique called minimum probability of a path. We describe a case study where TCG was used to generate and select functional and statistical test cases for a distributed system for health remote assistance.

The reminder of this work is organized as follows: in Section 2 we give the background that supports this work. Section 3 introduces the TCG tool and its main features. A case study using the proposed tool is described in Section 4, while a comparison with other tools is discussed in Section 5. Finally, Section 6 brings the conclusions and future work.

2 BACKGROUND

2.1 Model-based Testing

MBT is an approach that takes software specification as the base for the formal model creation and, from it, enables the test case extraction. MBT is composed by several activities, as depicted by Figure 1.

Firstly, the user takes the specification or requirement document and encodes that into an expected system behaviour model which the test generation tool can understand. This format can be either graphical, like UML diagrams and finite state machine formalisms, or textual, like the Aldebaran format (AUT)[2]. The formal model represents the high-level test objectives: it expresses the aspects of the system behavior that the engineer wants to test. In this work, we represent the system behaviour model as an LTS (Keller, 1976), which is a 4-tuple L = (S, A, T, q_0), where S is a non-empty finite set of states, A is a non-empty finite set of labels that denote the alphabet of L, T $\subseteq$ (S x A x S) defines the set of labelled transitions between states, and q_0 is the initial state.

The next step consists of selecting the most appropriate test case generation (or coverage) criteria. The choice of coverage criteria determines the algorithms that the tool uses to generate tests, how large a test suite it generates, how long it takes to generate them, and which parts of the model are tested (Utting and Legeard, 2007).

There are “families” of test generation criteria (Utting and Legeard, 2007), such as: (i) *Structural model coverage criteria*, which deal with coverage of the control-flow through the model, based on ideas from control-flow through programs; (ii) *Data coverage criteria*, which deal with coverage of the input data space of an operation or transition in the model; (iii) *Fault-model criteria*, which generate test suites that are good at detecting certain kinds of fault in the model; (iv) *Requirements-based criteria*, which aim to generate a test suite that ensures that all the informal requirements are tested; (v) *Explicit test case specifications*, which allow a validation engineer to explicitly say that a certain test, or set of tests, should be generated from the model; and (vi) *Statistical test generation methods*, which use random generation as a easy way to generate tests that explore a wide range of system behaviors. Currently, the proposed tool supports criteria (i) and (vi) as generation criteria, while (v) is used as test case selection criterion.

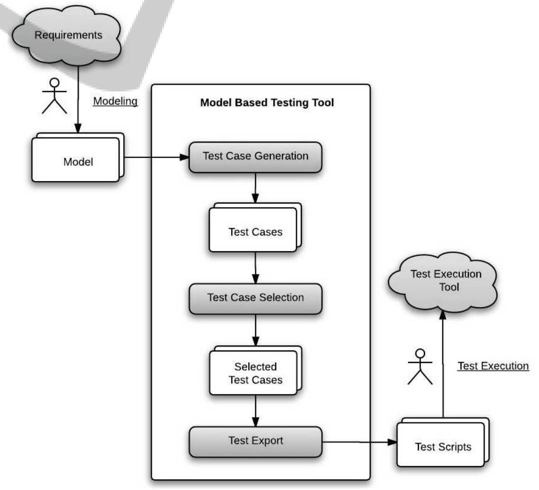


Figure 1: Model-based testing approach.

The output of the test generation is a collection of abstract test cases, called test suite, which are sequences of traces from the model. Each trace represents a possible usage scenario that needs to be tested.

Before the tests cases are returned, an alternative step is to choose a test selection heuristic. This is useful, and sometimes necessary, since the generation technique may produce a large number of test cases, which can turn the tester’s work impractical. This work implements selection heuristics that reduce the test suite size by both discarding similar test cases

[1]http://jeri.larces.uece.br/lotus

[2]http://www.inrialpes.fr/vasy/cadp/man/aldebaran.html

using a similarity degree function or letting the user select the scenarios that should be tested.

The next step of the MBT process is to export and concretize the abstract test cases into executable and/or human readable formats. Often this happens via some translation or transformation tool. Currently, this work does not support this phase.

Finally, the test execution happens using a test execution environment. In the case of manual execution, the abstract test cases are turned into manual test plans and detailed test steps for manual test execution.

2.2 Functional Tests

The functional test, also known as *black box* test, sees the SUT as a closed box in which there is no knowledge about how it is implemented or about its internal behaviour. This approach has the advantage of generating test cases before the implementation is ready, since it is based on the specification. On the other hand, it depends on the model, i.e., the test cases generated are the ones specified in the system behavior. If there are possible behaviors that are missing from the software specification, then the generated test cases will not cover all possible behaviors of the SUT (Cartaxo et al., 2008).

2.3 Statistical Tests

Statistical tests are based on the development of a system behavior model and, from it, the specification of a usage model that represents it, such as the probability of use of system functions. In (Walton et al., 1995), the authors state that usage models can be represented by Markov chains, and propose an eight-step methodology for creating those models, among which we highlight: review the software specification; identify the use of the software and set environmental parameters; develop the structure of the usage model and see if it conforms to the specification; develop and verify the probability distribution for the model. It is noteworthy that the usage model defines only functional aspects of a software, not considering non-functional or structural aspects of the modeled software.

According to (Poore and Trammell, 1999), among the benefits of statistical software testing using usage models are: (i) automatic test generation: the use of usage models is ideal for automatic test generation, e.g, using Markov chains for performing statistical software testing; (ii) efficient testing: testing become very structured and contribute to achieve more efficient results; and (iii) quantitative management of testing: quantitative results of the tests end up helping in management and decision making. As limitations, the authors cites the possibility of explosion of states, if the model is very detailed, or loss of information if the model is more abstract.

In this work we use the probabilistic LTS (PLTS) as the formalism for modeling the probabilistic behavior of the SUT. A PLTS extends an LTS adding, for each transition, a probability of occurrence between 0 and 1 such that the sum of the probabilities of the outgoing transitions from the same state is always 1. When removing the labels of the PLTS transitions, the resulting model is a Discrete Time Markov Chain (DTMC). We assume that the model is a proper representation of the system being modelled and that the transition probabilities are correct.

3 THE TCG TOOL

LoTuS is an open source lightweight tool for graphical modeling of the system behavior using LTS. It provides an interactive and simple way for the creation, composition and manipulation of LTSs. Furthermore, it allows the user to associate the LTS transitions to guard conditions and probabilities of execution, which makes possible modeling PLTSs. One of the main benefits of LoTuS is that it is extensible, providing a Java API with which a developer can create plugins to add new features to the tool. Some plugins have been developed and are available on the tool's website, such as a plugin for generation of execution traces and a plugin for annotation of probability of occurrence in the model's transitions from a file containing a bag of traces. The idea, therefore, is to permit the users to choose the plugins that they want to work with and to allow them to create their own plugins that meet their needs.

TCG (Test Case Generation) is a plugin to generate and select test cases for probabilistic and non-probabilistic models created by the user in LoTuS. The plugin is available as a jar file such that, to be loaded by LoTuS, it needs simply to be placed in the tool's *extensions* folder. When launched, LoTuS automatically loads all plugins that are in that folder. As LoTuS, the tool is free to download [3] and use.

The plugin creates the menu item "TCG" in the LoTuS' main menu. From it, two sub-menus are displayed, Functional and Statistical, that contain the available criteria for the generation and selection of test cases for non-probabilistic and probabilistic models, respectively.

[3]http://jeri.larces.uece.br/lotus/plugins/tcg

3.1 Test Case Generation

As discussed in Section 2, TCG addresses two kinds of test generation families: Structural model coverage criteria and Statistical test generation methods. Regarding the former, Utting and Legeard (2007) highlight four main categories: Control-flow-oriented coverage criteria, Data-flow-oriented coverage criteria, Transition-based coverage criteria, and UML-based coverage criteria. As TCG deals with state machine based formalisms (LTS and PLTS) with no information about control flow condition (guards) or input data, it provides only Transition-based coverage criteria.

More specifically, TCG implements 8 transition-based coverage criteria, which are:

- *All States*: returns a set of paths in which the union of them cover all the states of the LTS.
- *All Transitions*: returns a set of paths in which the union of them cover all transitions of the LTS.
- *All paths*: performs an exhaustive search, generating a large number of paths. In order to avoid an infinite execution, the user can set a timer as the *stop condition*, meaning that the algorithm will terminate after that time slot has passed and the test cases produced within that period will be returned.
- *All-loop-free paths*: returns all paths that do not contain any cycle, consequently, each generated path will not contain any repeated state.
- *All-one-loop paths*: returns all paths that contain at most one cycle, consequently, each generated path will contain at most one and only one repeated state.
- *Shortest path*: returns the shortest complete path.
- *Random path*: also known as *random walk*, returns a number of random paths specified by the user. For each path, the choice on which state to go is made non-deterministically.
- *Probabilistic random path*: similar to its non-probabilistic counterpart, with the proviso that the random choice is based on the probability distribution of the current state's outgoing transitions.

The result of the application of the test generation technique is one or a set of paths, each one representing a test case. For the statistical tests, each path is shown with its probability of occurrence, which is calculated by multiplying the probability of each transition between to consecutive states in the path.

3.2 Test Case Selection

Sometimes the number of test cases generated by the tool can be very large, which makes testing all test cases not applicable in most practical cases. Thus, it is necessary to reduce the set of test cases. We pay particular attention to strategies for selecting the parts (functionalities) of the software on which the testing must concentrate in order to avoid loss of time and effort by testing all possible test cases.

TCG provides 5 techniques for test case selection: Test purpose, Similarity of Paths, Weight similarity of Paths, Most Probable Path and Minimum Probability of Path.

3.2.1 Test Purpose

A test purpose is a specific objective (or property) that the tester would like to test, and can be seen as a specification of a test case. Test purposes can be used to select test cases. This approach is also referred to as "user guided" test selection. The test purposes approach was formally elaborated by (de Vries, 2001).

The notation to express test purposes may be the same as the notation used for the model, such as a statechart or an LTS, or it may be a different one (e.g., in UML, a behavior model based on class diagrams and state machines, and explicit test case specifications formalized with message sequence chart diagrams) (Utting and Legeard, 2007). In TCG, the test purpose follows the same approach used in (Cartaxo et al., 2008), which consists of using regular expressions composed by a sequence of model labels, possibly with a * among the labels. In that case, the * represents any transition label or simply no label. The sequence can end either with the word "ACCEPT", which means that all the test cases that satisfy the test purpose are desirable, or "REJECT", which means that we want the counterpart, i.e., all the test cases that do not satisfy the test purpose.

To specify the test purpose, the user must obey to one of the following rules: (i)*&ACCEPT: returns all possible paths; (ii)*a*&ACCEPT: returns all paths that have the label *a*; (iii) *a&ACCEPT: returns all paths that end with the label *a*; (iv) *a*b&ACCEPT: returns all paths that have the label *a* and end with label *b*; *a,b&ACCEPT: returns all paths that end with label *a* followed by label *b*; (vi) a*&ACCEPT: returns all paths that start with label *a*. Although the rules presented here consider only one or two labels, test purpose rules for three or more labels can be easily derived from those rules.

The same formation rules are valid for the sequences ending with "REJECT". However, the returned test cases are exactly the ones that do not satisfy the same sequence ending with "ACCEPT". For instance, the sequence *&REJECT returns no paths, while the sequence *a*&REJECT returns all paths that do not contain the label *a*.

3.2.2 Similarity of Paths

The idea is that whenever full coverage on model-based test case generation is unfeasible and/or test cases are mostly redundant, a similarity function can be automatically applied to discard similar test cases, keeping the most representative ones that still provides an adequate coverage of the functional model (Cartaxo et al., 2011). Given an LTS and a path coverage percentage as goal, the strategy reduces the size of the general test suite (according to the percentage), assuring that the remaining test cases are the less similar and covers the LTS transitions as much as possible.

The similarity degree between each pair of test cases (model paths) is defined by the number of equal actions that both paths have. This similarity degree indicates how close the test cases are. The choice is made by finding the smallest test case, i.e, the one with the smallest number of actions. In case both test cases have the same length, then the tool chooses one of them randomly.

3.2.3 Weighted Similarity of Paths

This approach is similar to the Similarity of Paths, with the proviso that the path probability is taken into account for the selection of the remaining test cases (Bertolino et al., 2008). The main difference is in the calculation of the similarity degree.

Weighted Similarity of Paths foresees the test cases selection by prioritizing accordingly with the probability of each test case. The idea is that when choosing between two similar test cases to be discarded, the one that has a greater probability of occurrence is kept. In addition, if the probabilities of the two test cases are the same, then the algorithm follows the Similarity of Path's steps. Therefore, we aim at testing suites that have the most different test cases and yet these are also the most important ones.

3.2.4 Remaining Techniques

The two other selection techniques are the *most probable paths* and the *minimum probability of a path*. The former consists of showing the N most probable paths, where N is informed by the user (the default value is 1, which means that only the most probable path is returned). The latter allows the user to inform the minimum occurrence probability expected for a test case. The tool returns only the test cases that satisfy that boundary. This selection technique complements the previous one and helps the user prioritize the most important test cases. Although intuitive, the minimum probability of a path has not been found in other MBT statistical testing tools.

4 CASE STUDY

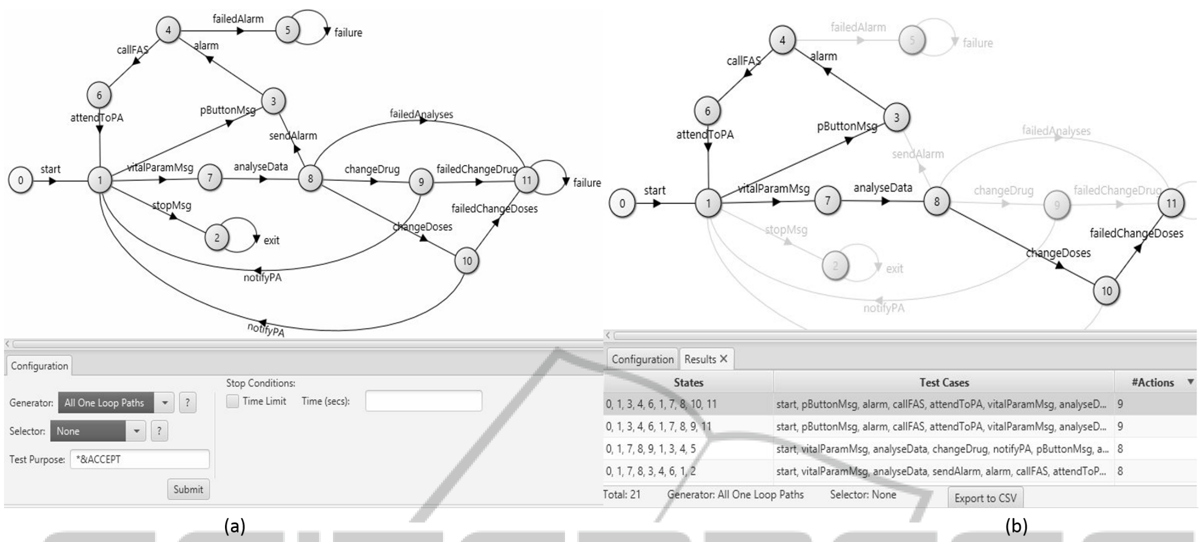
The case study has been carried out with an application called TeleAssistance (TA), a distributed system for remote assistance of patients (Epifani et al., 2009). We set up two scenarios: in the first one, we considered the non-probabilistic model, while the second one we assumed that the user annotated the model with execution probabilities, turning it into a PLTS.

The TeleAssistance offers three choices: sending the patient's vital parameters, sending a panic alarm by pressing a button and stopping the application. The data of the first message is forwarded to a Medical Laboratory service (LAB) to be analyzed. The LAB replies by sending one of the following results: *change drug*, *change doses* or *send an alarm*. The latter message triggers the intervention of a First-Aid Squad (FAS) whose task is to attend to the patient at home in case of emergency. When the patient presses the panic button, the application also generates an alarm sent to the FAS. Finally, the patient may decide to stop the TA service. TA can fail in the following situations: sending an alarm to the FAS, receiving the data analysis from the LAB, or sending a change dose or change drug notification to the patient. In all cases, the system goes to a final state indicating that a failure has occurred.

Figure 2(a) shows an excerpt of the LoTuS' main screen. At the top there is the TA behaviour model, and at the bottom the TCG main panel, from which the user can select the test case generation and selection techniques. In that example, the user selected the All-one-loop paths generator and no selector technique. Figure 2(b) depicts the result of the application of the technique previously chosen. Note that when a test case is selected, LoTuS highlights the corresponding path in the model.

4.1 Scenario 1: Using a Non-probabilistic Model

The first analysis consisted of generating the abstract test cases using the main coverage criteria. Table 1 shows the result. As expected, the All Paths criterion generated the greatest number of test cases (760),



Configuration

Generator: All One Loop Paths

Selector: None

Test Purpose: *&ACCEPT

Submit

Stop Conditions:

Time Limit   Time (secs):

Configuration | Results

States	Test Cases	#Actions
0, 1, 3, 4, 6, 1, 7, 8, 10, 11	start, pButtonMsg, alarm, callFAS, attendToPA, vitalParamMsg, analyseD...	9
0, 1, 3, 4, 6, 1, 7, 8, 9, 11	start, pButtonMsg, alarm, callFAS, attendToPA, vitalParamMsg, analyseD...	9
0, 1, 7, 8, 9, 1, 3, 4, 5	start, vitalParamMsg, analyseData, changeDrug, notifyPA, pButtonMsg, a...	8
0, 1, 7, 8, 3, 4, 6, 1, 2	start, vitalParamMsg, analyseData, sendAlarm, alarm, callFAS, attendToP...	8

Total: 21   Generator: All One Loop Paths   Selector: None   Export to CSV

Figure 2: (a)LoTuS main screen and TA behaviour model and (b)Result of the application of All-one-loop path generator.

Table 1: Comparison among the generation criteria.

Criterion	#test cases
All Paths	760
All One-loop paths	21
All Transitions	9
All Free-loop paths	6
All states	4

since it is an exhaustive algorithm, followed by All-one-loop paths (21), All transitions (9), All-free-loop paths (6), and All states (4).

The second analysis aimed at showing the application of different selection techniques. Then, to illustrate the results, let us assume again that the user chose the All-one-loop paths generation criteria.

For non-probabilistic models, two selection techniques are available: similarity of paths and test purposes. To exercise the former, we considered 3 scenarios, each one with a different similarity degree. After applying a path coverage of 25%, 50%, and 75%, TCG reduced the original number test cases to 6, 11, and 16, respectively.

Although the similarity of paths can considerably decrease the test suite size, the user cannot select the situations that he/she desires to test. In that case, the test purpose is more adequate. For instance, supposing the user desires to select only the test case where a successful termination of the TA happens (the system performs the *stopMsg* message). To do this, the search string *stopMsg&ACCEPT is used. Note that to use the test purpose, it is necessary first to generate the test cases using a particular criteria. This influences the result of the test purpose.

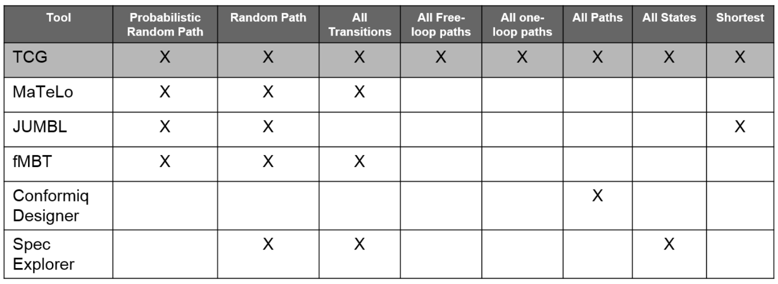
Applying that test purpose to the original test suite generated by All-one-loop paths criterion, it resulted in 5 test cases. If the user, on the other hand, chooses the All Path coverage criterion and subsequently applies that test purpose, the resulting test suite size jumped to 65 test cases.

Therefore, the choice on which generation algorithm to use impacts the number of test cases. The user should know the pros and cons of each criterion. For instance, using All Paths may be very expensive due to its large result set, but it covers practically all possible system behaviours. Each selection technique also affects the test case size. So, for instance, if the user chooses a very low degree of similarity, the number of test cases tend to be very small, which may not be enough to test the main software features.

4.2 Scenario 2: Using a Probabilistic Model

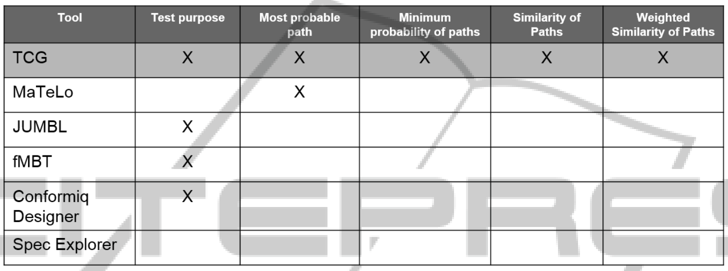
In this scenario, we use the same probabilistic model of the TA system used in (Epifani et al., 2009), which represents in the model the probability of execution of ordinary transitions and the probability of failure for failure transitions. As the generation techniques are the same used in the non-probabilistic counterpart, we focused on applying selection criteria that are specific of probabilistic models: the weight similarity of paths and minimum probability of path.

Considering again that the user has already generated the test cases using the All-one-loop paths technique, he/she desired to apply the weight similarity of paths to reduce the test suite size. After applying the same path coverage used in the Similarity of Paths



Tool	Probabilistic Random Path	Random Path	All Transitions	All Free-loop paths	All one-loop paths	All Paths	All States	Shortest
TCG	X	X	X	X	X	X	X	X
MaTeLo	X	X	X					
JUMBL	X	X						X
fMBT	X	X	X					
Conformiq Designer						X		
Spec Explorer		X	X				X	

Figure 3: Comparison regarding generation criteria.



Tool	Test purpose	Most probable path	Minimum probability of paths	Similarity of Paths	Weighted Similarity of Paths
TCG	X	X	X	X	X
MaTeLo		X			
JUMBL	X				
fMBT	X				
Conformiq Designer	X				
Spec Explorer					

Figure 4: Comparison regarding selection criteria.

example (25%, 50%, and 75%), we obtained the test suites with sizes 6, 13, and 19, respectively. Although the results seems equal, they are indeed different. The probabilistic approach selected some test cases that were removed in the previous one, since they had a greater probability of execution then its similar path. This is the case of, for instance, the path 0, 1, 3, 4, 6, 1, 2 (probability 4.78%).

In the second analysis, we used the minimum probability of path technique, in which the user can select only the test cases that satisfy the minimum probability informed by him/her. Supposing that the user is interested in testing firstly the scenarios that have probability of occurrence of at least 10%, TCG returns only one test case, which is the situation where the TA user stars and stops the device, without interacting with it (path 0,1,2 - probability 13.72%. The next most probable path is the one with probability 4,78% mentioned before. This occurred because the probability of the test cases are low, since the probability of the last action of each test case is very small (0.14 for the success transition and 0.04, 0.02, and 0.01 for the failure transitions).

5 RELATED WORK

There are several papers in the area (Shirole and Kumar, 2013)(Shafique and Labiche, 2013)(Aichernig et al., 2008) that show systematic reviews or *surveys* describing various approaches, techniques and MBT tools, comparing them using varied criteria. In this section we will focus the discussion on work closer to ours that also propose tools for generation and selection of functional and statistical test cases.

We have selected a few tools that deal with both types of models. These tools are shown in Figures 3 and 4. Note that they are the ones that could be downloaded and tested.

Figure 3 shows a comparison table among TCG and the other tools regarding the generation techniques. All transition and (probabilistic) random path are the most supported criteria and that are provided by all tools, except Conformiq Designer[4] . On the other hand, that tool is the only one which implements the All path criterion. In addition, Spec Explorer[5] and JUMBL[6] also provide criteria that are not supported by the other tools, namely All States and Shortest path, respectively. TCG is the only one which implements all discussed generation criteria.

Figure 4 complements the comparison regarding the selection techniques. Fours implement only one technique: MaTeLo (Most probable path) (Feliachi and Le Guen, 2010) and JUMBL, fMBT[7], ConformiqDesigner (Test purpose). Spec Explorer provides none of the selection techniques. One more time, TCG is the only one which supports the user

[4]http://www.conformiq.com/products/conformiq-designer/

[5]http://research.microsoft.com/en-us/projects/specexplorer/

[6]http://sqrl.eecs.utk.edu/esp/jumbl.html

[7]https://01.org/fmbt

with all discussed selection techniques, being this a great benefit of the tool.

6 CONCLUSION

This paper introduced TCG, an MBT tool for functional and statistical tests. It uses as input both LTS and probabilistic LTS models and provides 8 classic test case generation techniques along with 5 selection heuristics. The main contributions are the implementation of a variety of techniques in only one tool, including ones that have not been implemented in similar tools, and the introduction of a new selection criteria, the minimum probability of path. We showed a case study where the tool was used to generate and select test suites for a distributed system.

Currently we are increasing the tool features by allowing the automatically generation of test scripts from the produced test suites. We are also working on defining and implementing a probabilistic criteria to compare the whole test suite, and not only individual test cases. As future work, we intend to use the tool in other contexts, particularly real cases from the industry, to assess its usability and applicability, and implement some prioritization techniques.

REFERENCES

Aichernig, B., Krenn, W., Eriksson, H., and Vinter, J. (2008). D 1.2 - state of the art survey - part a: Model-based test case generation. Technical report, AIT Austrian Institute of Technology GmbH.

Antoniol, G., Briand, L., Di Penta, M., and Labiche, Y. (2002). A case study using the round-trip strategy for state-based class testing. In *Software Reliability Engineering, 2002. ISSRE 2003. Proceedings. 13th International Symposium on*, pages 269–279.

Beizer, B. (1995). *Black-box Testing: Techniques for Functional Testing of Software and Systems*. John Wiley & Sons, Inc., New York, NY, USA.

Bertolino, A., Cartaxo, E., Machado, P., and Marchetti, E. (2008). Weighting influence of user behavior in software validation. In *Database and Expert Systems Application, 2008. DEXA '08. 19th International Workshop on*, pages 495–500.

Binder, R. V. (1999). *Testing Object-oriented Systems: Models, Patterns, and Tools*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Cartaxo, E. G., Andrade, W. L., Neto, F. G. O., and Machado, P. D. L. (2008). Lts-bt: A tool to generate and select functional test cases for embedded systems. In *Proceedings of the 2008 ACM Symposium on Applied Computing*, SAC '08, pages 1540–1544. ACM.

Cartaxo, E. G., Machado, P. D. L., and Neto, F. G. O. (2011). On the use of a similarity function for test case selection in the context of model-based testing. *Softw. Test. Verif. Reliab.*, 21(2):75–100.

Dalal, S. R., Jain, A., Karunanithi, N., Leaton, J. M., Lott, C. M., Patton, G. C., and Horowitz, B. M. (1999). Model-based testing in practice. In *Proceedings of the 21st International Conference on Software Engineering*, ICSE '99, pages 285–294. ACM.

de Vries, R. G. (2001). In Tretmans, G. J. and Brinksma, H., editors, *Formal Approaches to Testing of Software 2001 (FATES'01), Aarhus, Denmark*, volume NS-01-4 of *BRICS Notes Series*, pages 61–76, Aarhus, Denkmark.

Epifani, I., Ghezzi, C., Mirandola, R., and Tamburrelli, G. (2009). Model evolution by run-time adaptation. In *ICSE '09: Proceedings of the 31st International Conference on Software Engineering*, pages 111–121, Vancouver, Canada. ACM.

Feliachi, A. and Le Guen, H. (2010). Generating transition probabilities for automatic model-based test generation. In *Software Testing, Verification and Validation (ICST), 2010 Third International Conference on*, pages 99–102.

Keller, R. M. (1976). Formal verification of parallel programs. *Commun. ACM*, 19:371–384.

Poore, J. and Trammell, C. (1999). Application of statistical science to testing and evaluating software intensive systems. In *Science and Engineering for Software Development: A Recognition of Harlan D. Mills' Legacy, 1999. Proceedings*, pages 40–57.

Sayre, K. and Poore, J. H. (1999). Partition testing with usage models. In *Proceedings of the Science and Engineering for Software Development: A Recognition of Harlan D. Mills' Legacy*, SESD '99, pages 24–. IEEE Computer Society.

Shafique, M. and Labiche, Y. (2013). A systematic review of state-based test tools. *International Journal on Software Tools for Technology Transfer*, pages 1–18.

Shirole, M. and Kumar, R. (2013). Uml behavioral model based test case generation: A survey. *SIGSOFT Softw. Eng. Notes*, 38(4):1–13.

Trammell, C. (1995). Quantifying the reliability of software: Statistical testing based on a usage model. In *Proceedings of the 2Nd IEEE Software Engineering Standards Symposium*, ISESS '95, pages 208–, Washington, DC, USA. IEEE Computer Society.

Utting, M. and Legeard, B. (2007). *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Walton, G. H. and Poore, J. H. (2000). Generating transition probabilities to support model-based software testing. *Softw. Pract. Exper.*, 30(10):1095–1106.

Walton, G. H., Poore, J. H., and Trammell, C. J. (1995). Statistical testing of software based on a usage model. *Softw. Pract. Exper.*, 25(1):97–108.

Weissleder, S. and Schlingloff, H. (2014). An evaluation of model-based testing in embedded applications. In *Software Testing, Verification and Validation (ICST), 2014 IEEE Seventh International Conference on*, pages 223–232.

Zhou, K., Wang, X., Hou, G., and Jie Wang, S. A. (2012). Software reliability test based on markov usage model. *Journal of Software*, 7(9):2061–2068.