Generating transition probabilities for automatic model-based test generation

Abderrahmane FELIACHI* and Hélène LE GUEN
All4tec GL, 6 rue Léonard de Vinci, BP 0119. 53001 Laval Cedex, France.
abderrahmane.feliachi@lri.fr, helene.leguen@all4tec.net
**Now at: L.R.I., Université Paris-Sud 11 & CNRS, bât. 490, 91405 Orsay Cedex, France.*

***Abstract*—Markov chains with Labelled Transitions can be used to generate test cases in a model-based approach. These test cases are generated by random walks on the model according to probabilities associated with transitions. When these probabilities correspond to a usage profile, reliability may be estimated. However, in early stages of development, such probabilities are not easy to determine, thus default profiles must be considered. In such a case it may be interesting to target some coverage criteria rather to use classical uniform probability generation approach.**

In this paper we enrich an existing industrial tool based on usage profile with 3 possibilities to create default profiles that improve transition coverage. We report experiments that compare the improvement of the coverage rates by our approaches with respect to uniform probabilities on transitions from a given state, which is the current default profile.

***Keywords*-Model-Based Testing; Markov chain; profile generation; transition probabilities; coverage criteria.**

I. Introduction

Model-based testing is an activity that is essential to automate testing. It can also define criteria for selecting test cases. Some probabilistic models are used in Model-Based Testing to represent usage profile and to estimate reliability. It is the case when using MaTeLo, a tool developed by All4tec company. These models can have multiple profiles, all of which are not necessarily indicative of user behaviour.

A first reason is that it is not always obvious in early testing stage to state a realistic usage profile. Thus it is interesting to study default profiles generation. A second issue is that, like any testing approach, it is interesting to ensure coverage of system requirements. For these reasons we are interested in creating profiles for an even coverage of these models by investigating random generation techniques.

In this paper we aim at improving MaTeLo by adding 3 profile generation solutions (based on graph algorithms). We will also rely on this tool to perform experiments.

II. The MaTeLo tool

MaTeLo is a tool developed by All4tec; its purpose is to generate test cases for systems described by a probabilistic model. It consists of two applications, the first one allows modeling of systems, and the second one allows the generation of test cases from models. MaTeLo is used in several domains, and has improved the reliability of the tested systems. As can be seen in [1].

A. Models in MaTeLo

Models are based on the work presented in [2]. A test graph G is defined by (S, Invoke, End, T, V, F) such that:

- S: a finite set of states;
- Invoke ∈ S: the unique initial state;
- End ∈ S: the unique final state;
- V: a set of inputs, outputs and global variables;
- F: a set of "transfer functions": f (V1) = V2 with V1, V2 ∈ V. V1: a set of global variables and / or input vectors; V2: a set of output vectors;
- T: a set of transitions; partitioned in two subsets:
- Tp: a finite set of probabilistic transitions of the form $s \xrightarrow{f,P} s'$, with s ∈ S/{End}, s' ∈ S/{Invoke}, f ∈ F, P ∈ [0, 1]. P is the probability of choosing s' from s.
  o In Tp we distinguish TA: a set of asynchronous transitions, connecting all the states to a state s';
- Tc: a finite set of conditional transitions of the form $s \xrightarrow{C} s'$, with s ∈ S/{End}, s' ∈ S/{Invoke}; C is a logical expression expressing a condition over V.

This definition is illustrated in Fig. 1.

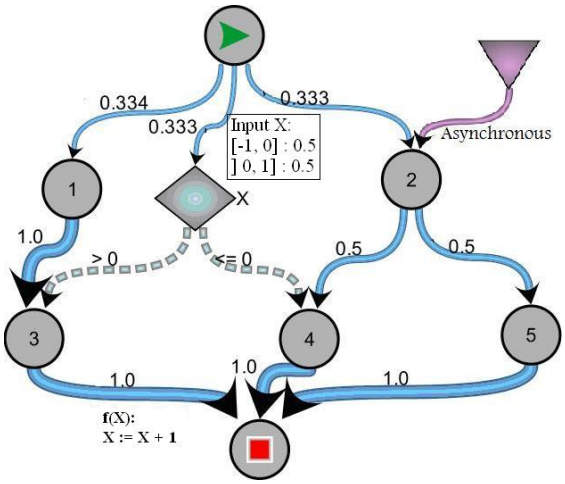

Figure 1. A simple MaTeLo model

This model contains probabilistic and conditional transitions, one input named X with probability distribution over 2 intervals; X can also be modified after crossing state 3 by the function f(X).

We further impose that:

- The unique initial state has no incoming transition and the unique final state has no outgoing transition ;

- All states except the final one have at least one outgoing conditional or probabilistic transition ;
- All states except the initial one have at least one incoming conditional or probabilistic transition ;
- A state can have either outgoing probabilistic transitions or outgoing conditional transitions, but not both ;
- Asynchronous transitions probabilities are fixed on the model but don't appear in probability sum. In test generation phase model probabilities are recalculated in order to support these probabilities.

During the tests generation phase, when firing a transition the function associated with is executed. Function parameters can be variables, constants or inputs generated randomly according to some distribution described with the transition. The function performs (arithmetic or logic) computations on inputs, update variables and / or produce some output.

Currently, MaTeLo deals with models with up to 10000 states or more. Even in large models, only few conditional transitions are used. Therefore, we will mainly focus on probabilistic transitions, and consider in this paper only models without conditional transitions.

In order to simplify modeling, MaTeLo uses a hierarchical composition of models. A state will be either a simple state or a "macro state" which itself refers to another model called "sub model". This composition improves readability, modularity and facilitates reuse.

B. Tests generation in MaTeLo

MaTeLo currently uses three tests generation algorithms:

1) Random walk: From Invoke state, all global variables are initialized to default values. An outgoing transition is chosen randomly according to its probability. Once the transition is selected, inputs (if any) are generated according to the domain distribution and the associated function is executed. Generation of a test ends when reaching the final state. A test case is built as a succession of information (values generated for inputs) associated with transitions traversed during the generation.

2) Most probable: A single test case is generated by choosing at every state with probabilistic transitions, the transition with the highest probability.

3) Chinese Postman: Generates a set of test cases covering all transitions (ignoring transition probabilities).

C. Operational Profile

An operational profile is a model equipped with probabilities for all its probabilistic transitions and inputs. Each profile corresponds to a future use (or type of use) of the system, and describes its behavior through probabilities. Usually a given "raw" model will be associated with several operational profiles provided by the designer.

In practice, probabilities are not easy to estimate and it could be comfortable to have default profiles generated automatically. The easiest way to automatically generate profiles is to locally fix probabilities by uniform distribution among the transitions leaving a state. This is the current solution in MaTeLo. Often, tests generated using such profiles do not cover the system well (see Fig. 2).

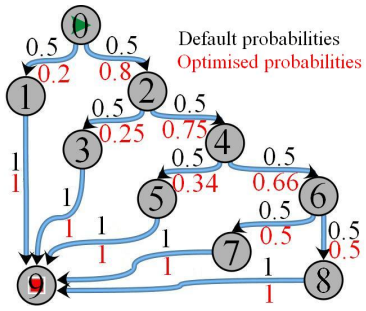

Figure 2. Default and optimised probabilities

The example of Fig. 2 is very simple according to real model and it doesn't contain cycles. Let call "visiting probability" v_i the probability to visit state i during a random walk from the initial state, then the vector (v_s, s = 1..8) of the visiting probabilities is (0.5, 0.5, 0.5^2, 0.5^2, 0.5^3, 0.5^3, 0.5^4, 0.5^4).

In this example the expected number of test cases to obtain a full state coverage is greater than $1/0.5^4$ = 16 test cases (expected number of test needed to cover states 7 or 8).

The most favourable profile is the one that gives equal visiting probabilities to states 1, 3, 5, 7, 8 i.e. a visiting probability equal to 0.2, since other nodes are covered when these ones are. If we note p_{ij} the probability to choose state j from state i, the optimal solution is obtained by p_{01}=1/5, p_{02}=4/5, p_{23}=1/4, p_{24}=3/4.. p_{67}=1/2 and p_{68}=1/2. In this case v =(0.2, 0.8, 0.2, 0.6, 0.2, 0.4, 0.2, 0.2) and the expected number of test needed to cover any given state is 5.

We see on this very simple example the impact of probabilities and the interest to have a default profile adapted to a generation that takes into account the shape of the graph.

In this paper we propose some ways to compute profiles. When used in test generation, these profiles improve system coverage by tests. Structural coverage criteria on states and transitions will be used to compare solutions. Path and data criteria being difficult to reach in practice (sometimes impossible in presence of cycles), states and transitions coverage will be sufficient for our purposes.

III. Profile generation

Recall that we consider models with only probabilistic transitions, so all paths are "executable".

Transitions coverage was studied in directed graphs as Chinese postman problem; we use this as a key idea for our first solution. Transition coverage was also studied in test domain; T.S. Chow presented a solution that ensures transition coverage by computing a "test tree" covering all transitions. Tests are generated from this tree by exploring different paths. We finally propose a third solution that generates a profile by a simple model exploration.

A. Solution 1: adapting the Chinese Postman Problem

The Chinese postman problem (hereafter CPP) in a directed graph [3] consists in finding a minimum cost closed circuit that goes at least once over every transition. A fictitious arc (of weight 0) is added between the final and the initial states to ensure the connectivity of the graph. The weights of all other arcs are 1.

One minimal circuit from model in Fig. 3 could be:
$0 \rightarrow 2 \rightarrow 3 \rightarrow 0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 0 \rightarrow 1 \rightarrow 3 \rightarrow 0$.

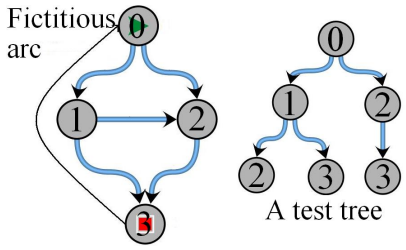

Figure 3. A simple model and its test tree

To extract probabilities from the circuit, we follow it and count the number of times we pass by a state or a transition. Let N[s] be the number of occurrences of a state s in the circuit, T[s, s'] the number of occurrences of transition s→s' in the circuit, the probability of a transition s→s' is fixed as pss' = T[s, s'] / N[s]. Each path up to the fictitious arc corresponds to a separate test case to generate. By the way the algorithm works, we usually obtain a small number of test cases each of rather great length.

B. Solution 2: adapting Chow's algorithm

The solution proposed by Chow [4] (hereafter CH) constructs a test tree covering all transitions. The idea is to build a tree from the model by doing a breadth-first traversal and adding arcs to the tree if they are in the model. The traversal stops when one reaches the final state, or if the transition was already traversed. We usually obtain more test sequences than with CPP, each of relatively short length. Fig. 3 presents a test tree calculated from the last model.

Probabilities are obtained as above; the calculation of N and T is done by traversing the tree from root to leaves, and counting the passes over states and transitions.

Since such a tree is not unique, the solution can be improved by minimizing the number of leaves, which will allow us to cover more transitions with fewer paths.

C. Solution 3: our proposed solution

Model with cycles are more difficult to process; first two solutions make no provision for cycles. In this solution we will explicitly handle them.

To limit the number of cycles crossing, one possible solution (CL) is to apply an algorithm that traverses the graph in-depth first to count the weight (the number of passes) of each state and transition. Transitions generating a cycle will have a fixed weight that can be given as a parameter to the algorithm, allowing the designer to better fit the test of cycles for the generation phase.

The probability of a transition s→s' is: $p_{ss'} = T[s, s'] / N[s]$. N[s] gives for each state s, the number of possible paths from this state to the final state. T[s, s'] gives for a transition s→s', the number of possible paths to the final state. The number of possible paths from a state (transition) is equal to the sum of possible paths from its successors. For cycles, i paths are counted ("i" is the algorithm parameter).

The algorithm is presented below.

```
Algorithm CL
In:  graph G ; Int i ;
Out: weight matrix T ; weight vector N, E
Var: State s; list of State Ch;
Begin
  s := initial state ;
  Ch: = empty list;
  T and E are initialized to 0;
  N [final state] :=1;
  Generate(s, Ch, i);
End
```

```
Function Generate (s, Ch, i)
Begin
  Add s to Ch;
  For each s' successor of s do
    If Ch doesn't contains s' then
      If E[s'] = 0 then // Unprocessed paths
        Generate(s', Ch, i);
      End if
      N [s] := N[s] + N[s'];
      T[s, s'] := N[s'] ;
    Else // cycle
      N[s] := N[s] + i; // i : weight given to cycles
      T[s, s'] := i;
    End if
  End for
End
```

D. Adapting solutions to support macro states

Solutions described above are applicable to simple models (without macro states); we implemented a generalization for these methods to support complete models by:

- Generating probabilities for all sub-models and adding paths number to the corresponding macro state weight ;
- Recalculating weights and probabilities in order to spread macro states weights in the whole model.

IV. Experiments

Experiments consists in profile generation and then test generation using these profiles. We used a model given by a ckient with 1500 states and 2500 transitions, made of several sub-models. The current MaTeLo profile generation method (let's call it A) is compared with the new solutions. The profile generation time difference is not very important between the three solutions. It is moreover negligible w.r.t. the time for generating tests.

We used the profiles to generate "k" tests, and then computed the coverage rate. "k" starts from 1 and is incremented until a coverage rate up to 95%. A test length is its number of transitions and the test suite length (generated "k" tests) is the sum of the lengths of the "k" tests. Since we use random walk techniques, the generation is done 10 times for every given "k", and we consider the average of coverage rate.

Fig. 4 and 5 show states and transitions coverage for the 4 algorithms (CPP, CH, CL and A) depending on the length of the test suite. These results show that the three new profiles allow better coverage than current solution A. For instance, the three algorithms cover 80% of transitions with only 15,000 tests, while solution A reaches this rate with not less than 30,000. The difference between the solutions is not very significant, but the CPP solution converges rapidly because of the great probability given to cycles.

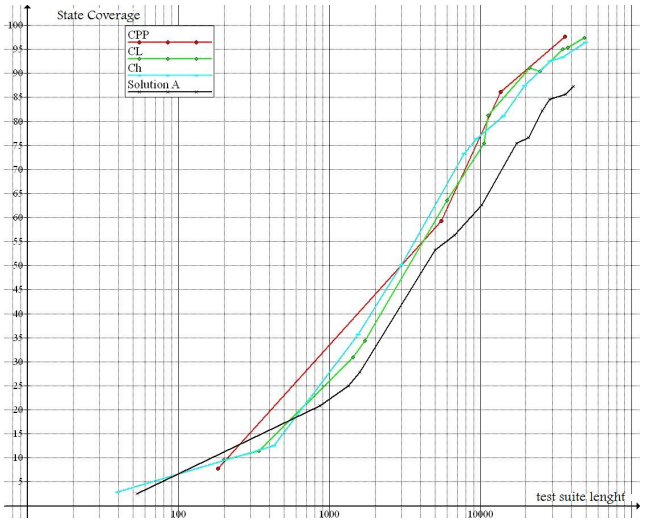

Figure 4. State coverage

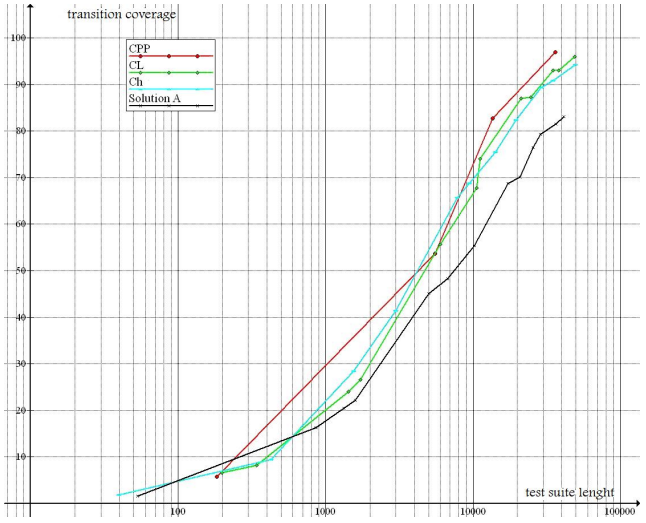

Figure 5. Transition coverage

Tests generated using CPP are longer but fewer than with CH and CL. In practice, tests are stored and executed several times. For some clients, for material reasons, long tests are not practicable; the CPP profile is not the best in this case. But if the lengths of the tests are acceptable CPP is preferable because it converges more rapidly.

V. Related works

Probabilistic models based on Markov chains have been the subject of numerous works, in particular the problem of the generation of transitions probabilities.

Walton and Poore [5] process optimal probabilities based on an objective and a number of constraints. Their solution is based on uniform probability generation (as solution A), and then these probabilities are optimized according to some objectives and constraints. Another interesting work is that of Gutjahr [6], its goal is to produce test cases to estimate the risk, safety and reliability. Solutions we proposed are different because they are only based on structural composition of the model and not on its specificities.

The work of Denise, Gaudel et al. [7], [8] proposes a uniform generation of paths of length less than a predefined value, and can be adapted to other criteria like states and transitions coverage. Probabilities transitions are not fixed, but depend on the traversal context (number of remaining steps before reaching a given state). This work is interesting but not applicable to current MaTeLo which gives a unique transition probability independently of traversal context.

VI. Conclusions

In this paper we tackled the transitions probabilities generation problem, in a model-based test generation context. We proposed 3 profile generation solutions, and give some experimental comparison with current solution. First experiments show that all the proposed solutions increase the coverage of the model by the generated tests; but one of the solutions (CPP) has a different behavior, it generates less but longer tests. The proposed solutions will be implemented in the MaTeLo tool to provide users with a wider choice in terms of profile generation.

References

[1] All4tec website: http://www.all4tec.net/.

[2] H. Le Guen, "Validation d'un logiciel par le test statistique d'usage: de la modélisation à la décision de livraison," Ph.D. dissertation, Université de Rennes 1, 2005.

[3] H. W. Thimbleby, "The directed chinese postman problem," *Softw., Pract. Exper.*, vol. 33, no. 11, pp. 1081–1096, 2003.

[4] T. S. Chow, "Testing software design modeled by finite-state machines," *IEEE-TSE*, vol. 4, no. 3, pp. 178–187, 1978.

[5] G. H. Walton and J. H. Poore, "Generating transition probabilities to support model-based software testing," *Softw. Pract. Exper.*, vol. 30, no. 10, pp. 1095–1106, 2000.

[6] W. J. Gutjahr, "Software dependability evaluation based on markov usage models," *Performance Evaluation*, vol. 40, no. 4, pp. 199–222, 2000.

[7] S.-D. Gouraud, A. Denise, M.-C. Gaudel, and B. Marre, "A new way of automating statistical testing methods," in *Proc. of the 16th IEEE int. conf. on Aut. Softw. Eng.*, 2001, p. 5.

[8] A. Denise, M.-C. Gaudel, S.-D. Gouraud, R. Lassaigne, J. Oudinet, and S. Peyronnet, "Coverage-biased random exploration of large models and application to testing," LRI, Tech. Rep. 1494, 2008, http://fortesse.lri.fr/pubs/fortesse.html.